ANALISIS ESTRUCTURADO MODERNO

ANALISIS ESTRUCTURADO MODERNO

PRENTICE HALL

ANALISIS ESTRUCTURADO MODERNO

ANALISIS ESTRUCTURADO MODERNO

Edward Yourdon

Traducción:
Física Alexandra Taylor Armitage
Facultad de Informática y Computación
Universidad de Guadalajara

Revisión Técnica:
Guillermo Levine Gutiérrez
Director Facultad de Informática y Computación
Universidad de Guadalajara

PRENTICE-HALL HISPANOAMERICANA, S.A.
México - Englewood Cliffs - Londres - Sydney - Toronto
Nueva Delhil - Tokio - Singapur - Rio de Janeiro

EDICION EN ESPAÑOL

DIRECTOR: Raymundo Cruzado González
EDITOR: José Tómas Pérez Bonilla
GERENTE DE TRADUCCION: Jorge Bonilla Talavera
SUPERVISOR DE TRADUCCION Enrique Palos Báez
GERENTE DE PRODUCCION: Eloy Pineda
SUPERVISOR DE PRODUCCION: Teresa Parra Villafaña

EDICION EN INGLES
Editorial/production supervision: Sophie Papanikolaou
Cover design: Wanda Lubelska Design
Manufacturing buyer: Mary Ann Gloriande

EDWARD YOURDON: ANALISIS ESTRUCTURADO MODERNO 1/E

Traducción de la primera edición en inglés de

MODERN STRUCTURED ANALYSIS

Enrique Jacob 20, Col. Conde
53500 Naucalpan de Juárez, Edo. de México

ISBN 968-880-303-0

Miembro de la Cámara Nacional de la Industria
Editorial, Reg. Núm. 1524

ISBN 0-13-598624-9

IMPRESO EN MEXICO / PRINTED IN MEXICO

MR

LITOGRÁFICA INGRAMEX, S.A. DE C.V.
CENTENO 162-1
MEXICO, D.F.
C.P. 09810

XXX 1996

CONTENIDO

PREFACIO

Lo que es valioso no es nuevo y lo que es nuevo no es valioso.
Henry Peter, Lord Brougham
The Edinburgh Review, 1802

Permítanme comenzar por hacer una pregunta muy importante: *¿realmente necesita el mundo de otro libro de análisis de sistemas?* Esto pudiera parecer una pregunta retórica, pero ha habido muchas ocasiones, usualmente ya entrada la noche, al estar trabajando en este libro, que me he preguntado: "¿Por qué estoy haciendo esto? ¿Qué hay de malo con los demás libros que todo mundo ha estado leyendo durante los últimos diez años? ¿Cómo puedo tener siquiera la esperanza de añadir algo al cuerpo literario existente?"

Obviamente, serán otros los que tengan que juzgar los resultados. Pero sí pienso que es necesario un libro que actualice algo del material clásico de análisis estructurado que se publicó por primera vez a finales de la década de los 70. Cuando Tom DeMarco escribió *Structured Analysis and Systemas Specification*, y Chris Gane y Trish Sarson escribieron *Structured Systems Analysis: Tools and Techniques*, no existían los lenguajes de programación de cuarta generación y no había herramientas de creación de prototipos disponibles para los creadores de sistemas. Las computadoras personales no existían en aquellos días, a excepción de algunas de las máquinas primitivas de Apple y Radio Shack. No había productos de software para estaciones de trabajo que pudieran auxiliar al analista de sistemas en la creación de diagramas de flujo de datos.

El desarrollo en estas áreas ha tenido un gran impacto en la aceptación generalizada del análisis estructurado: muchos discuten sobre si el análisis estructurado es pertinente en un ambiente en el que los usuarios crean sus propias aplicaciones en cuestión de horas o de días. Esto por sí solo es razón para crear un nuevo libro que trate el tema del análisis de sistemas: la disponibilidad de tecnología enormemente más poderosa, tanto para analistas de sistemas como para el usuario, ha cambiado nuestro enfoque y perspectiva.

Además, los creadores de sistemas tuvieron que hacerse cargo de cuestiones como sistemas de bases de datos y sistemas de tiempo real, así como de los sistemas "orientados a funciones" originalmente tratados por el análisis estructural a fines

de la década de los 70. Este libro analiza los diagramas de entidad-relación y de transición de estados, además de los clásicos diagramas de flujo de datos, y muestra cómo pueden integrarse los tres modelos; esta integración de modelos se volverá más y más importante en los años venideros. Se han incluido en este libro varios otros avances recientes en el análisis estructurado, incluyendo la partición de eventos y la menor importancia del modelado de los sistemas físicos actuales.

Existe una razón más para escribir otro libro sobre el análisis de sistemas: la mayoría de los libros "clásicos" de análisis estructurado están dirigidos a analistas de sistemas adultos y veteranos, con poca o ninguna consideración para la persona más joven que apenas comienza en el campo. Al mismo tiempo, la mayoría de los textos universitarios que tratan el análisis de sistemas y que se escribieron durante los últimos diez años prestaban escasa atención a las nuevas técnicas de análisis estructurado y han continuado dedicando demasiadas páginas a discusiones sobre las tarjetas perforadas y los códigos Hollerith. Además del hecho de que muchos de estos temas son obsoletos, generalmente se ofrece un estudio superficial del hardware de las computadoras, el software y la programación por medio de un curso de "introducción a la computación" que precedería a un curso a profundidad de análisis de sistemas. Este libro procura ser balanceado al reconocer que es necesario algún material introductorio para el estudiante que ha llevado un curso de introducción a las computadoras pero que *nunca* ha hecho análisis de sistemas, al mismo tiempo que reconoce que los conceptos del análisis estructurado son lo suficientemente sencillos como para ser presentados en bastante detalle a nivel de bachillerato y a nivel universitario. Sin embargo, la mayor parte del material introductorio se colocó en los apéndices, de modo que pueda omitirlo el profesional en la industria.

El libro debe ser apropiado para un curso de análisis de sistemas de un semestre, en el nivel de licenciatura; cubre los temas para el curso CIS-86/5 en el currículum modelo "CIS 86" para la licenciatura en informática. Sin embargo, no pretende abarcar tanto el tema del análisis de sistemas como el de su diseño, a pesar de que varias universidades tratan de comprender ambos en un solo semestre. Creo que hay bastante material para discutir en cada una de las dos áreas; recomiendo, para un curso de un semestre de diseño estructurado, los siguientes libros: *Practical Guide to Structured Systems Desing*, segunda edición, de Meilir Page-Jones (YOURDON Press, Englewood Cliffs, N.J., 1988), o bien *Structured Design*, segunda edición, de Ed Yourdon y Larry Constantine (YOURDON Press, Englewood Cliffs, N.J., 1989).

Es posible que los veteranos del análisis de sistemas quieran leer sólo el primer capítulo para orientarse y saltarse el resto de la parte I; los primeros siete capítulos son esenciales para los estudiantes nuevos. Los veteranos estarán familiarizados con los diagramas de flujo de datos, los diccionarios de datos, etc. Sin embargo, en el capítulo 9 hay material que trata de las extensiones de los diagramas de flujo de datos para los sistemas de tiempo real, que pudieran ser novedosos para aquellos que se hayan dedicado exclusivamente a los sistemas de

negocios. También pudieran ser novedosos los diagramas de entidad-relación para los que estén más familiarizados con los "diagramas de estructura de datos"; el capítulo 13, que trata de los diagramas de transiciones de estados, brinda una herramienta de modelado nueva e importante. Y algo que es de suma importancia para el veterano: los capítulos 19 y 20 brindan un planteamiento que contrasta enormemente con el enfoque rígido de arriba-abajo que han seguido los analistas durante años para la construcción del modelo esencial (a veces conocido como el modelo lógico); se trata del método conocido como *partición de eventos*, basado en la obra de McMenamin y Palmer. *El Capítulo 17 recomienda que se elimine el enfoque clásico de modelar el sistema actual del usuario*; los analistas de sistemas cuya técnica se basa en textos de la década de los 70 debieran estudiar esto con cuidado.

Entre los apéndices hay dos casos que ilustran las técnicas y herramientas que se tratan en este libro. El primero es una aplicación típica de negocios, basada en la operación de la editorial YOURDON Press; el segundo es un ejemplo más característico de un "sistema de tiempo real", basado en el sistema de control de un elevador. Ambos se presentan con detalle, aunque esto haga más grueso el libro, es importante para el estudiante ver un ejemplo completo de una especificación. Ambos modelos pueden usarse para discusiones y ejercicios en clase.

El análisis estructurado moderno utiliza años de experiencia con cientos de clientes, miles de estudiantes y docenas de colegas de las empresas YOURDON, Inc., y de otras organizaciones. Me declaro en deuda con todas estas personas, que son demasiadas para poderlas nombrar. Pero hay algunas que merecen una mención especial, pues me ayudaron a intentar que este libro fuera mejor. En la actualidad no se puede escribir un libro de análisis estructurado sin darle reconocimiento a las obras precursoras de Tom DeMarco, Chris Gane y Trish Sarson. Y me siento igualmente en deuda con Steve McMenamin y John Palmer, quienes dieron un importante paso adelante con su obra *Essential Systems Analysis*; de manera similar, Paul Ward y Steve Mellor introdujeron varios conceptos y herramientas importantes para los sistemas de tiempo real con su obra de tres volúmenes, *Structured Development for Real-Time Systems*. También me han ayudado los debates que he tenido con otros colegas, al enseñar análisis estructurado en Estados Unidos e Inglaterra. Agradezco de manera especial a John Bowen, Julian Morgan, Bob Spurgeon, Nick Mandato y Alex Gersznowicz por haberme mostrado maneras muy elocuentes de explicar el análisis estructurado que jamás se me hubieran ocurrido. Además, el profesor Peter Brown y un grupo de alumnos suyos de la Universidad Duquesne depuraron el libro, al utilizarlo como texto para un curso de análisis estructurado; les agradezco sus sufrimientos con todos los errores tipográficos que tenían los primeros borradores.

También quiero agradecerle a Dennis Stipe de la sucursal en Washington D.C. de YOURDON, Inc., por el intenso trabajo de análisis que hizo en el modelo de análisis estructurado del sistema de elevadores del apéndice G. La mayoría de los textos actuales sólo contienen casos de estudio de sistemas orientados a los nego-

cios; este libro contiene tanto un caso orientado a negocios (apéndice F), como un ejemplo de tiempo real, basado en la descripción de la ACM [*Association for Computing Machinery, N. del T.*] de un problema real. Originalmente, Dennis diseñó el modelo de análisis estructurado para un seminario de "guerras de diseño" patrocinado por la sección Washington de la ACM, en 1986, con el propósito de mostrar cómo se manejaría el problema del control de un sistema de cuatro elevadores con las diferentes metodologías de ingeniería de software; desde entonces se ha modificado varias veces.

Peter Brown, Pete Coad, Bob Spurgeon, Steve Weiss, Ron Teemley y Dale Brown mejoraron *enormemente* el borrador de este libro con sus revisiones y comentarios; obviamente, me hago responsable de todos los errores cometidos y de las omisiones que aún queden. Mientras tanto, mi esposa y mis hijos me dieron un gran apoyo abasteciéndome continuamente de Coca Cola dietética y papitas (y ocasionalmente cognac, cuando la ocasión lo ameritaba); para cuando acabé el libro, había aumentado 10 kilos de peso y tuve que ponerme a dieta.

En contraste con lo que los autores paranoicos como yo suelen sufrir a manos de los editores, varias personas de YOURDON Press/Prentice Hall hicieron de la producción de este libro una experiencia encantadora. Pat Jenny y Ed Moura vigilaron el proyecto desde el principio y me dieron ánimos cuando más lo necesitaba. Sophie Papanikolaou supervisó la producción del libro y fue un placer trabajar con ella. Bill Thomas se encargó de la revisión del libro y encontró los miles, c millones, de errores tipográficos y gramaticales. Después, mi esposa, Toni, corrigió de buen grado todos los errores en la computadora, aunque la escuché murmurar en voz baja que un matemático no debería pretender que sabe escribir.

Finalmente, me gustaría agradecerle a mi(s) computadora(s) Macintosh por batallar valientemente con el enorme manuscrito. La mayor parte del escrito se hizo con Microsoft Word (versiones 1.0, 1.05, 3.0, y 3.01) pero también utilicé MacPaint, Fullpaint, SuperPaint, MacDraw, Microsoft Chart, MacProject, Microsoft Multiplan, ChessMaster, ConcertWare, MORE de Living Viceotext, MacBubbles de StarSys Inc., y Design de Meta Software, sin contar varios fragmentos de "arte del recorte" de T/Maker y otros editores. Quienquiera que intente escribir un libro con algo que no sea una Macintosh debería ir a que le examinen la cabeza.

Edward Yourdon
Nueva York, agosto de 1988

PARTE 1: INTRODUCCION

1 INTRODUCCION

El comienzo y el final de toda actividad humana son desordenados: la construcción de una casa, la escritura de una novela, la demolición de un puente y, eminentemente, el término de un viaje.

John Galsworthy
Over the River, 1933

En este capítulo se aprenderá:

1. Por qué es interesante el análisis de sistemas.
2. Por qué es más difícil el análisis de sistemas que la programación.
3. Por qué es importante estar familiarizado con el análisis de sistemas.

Bien. Aquí estamos al comienzo de un largo libro. La perspectiva de leer un libro técnico tan largo probablemente lo aterrorice; pero, si le sirve de consuelo, es aún más aterradora la perspectiva de estar comenzando a *escribirlo*. Afortunadamente, así como los viajes largos se llevan a cabo un día por vez y, por último, un paso a la vez, de igual manera se acaban de leer los libros largos un capítulo por vez y, a fin de cuentas, una frase a la vez.

1.1 ¿POR QUE ES INTERESANTE EL ANALISIS DE SISTEMAS?

Los libros largos a menudo son aburridos; espero que éste no lo sea. Por fortuna, el tema de este libro, *análisis de sistemas*, es interesante. De hecho, el análisis de sistemas es más interesante que cualquier cosa que yo conozca, con la

Aun cuando no se vaya a dedicar a un empleo de oficina (es decir, si piensa ser artista, escritor, músico o atleta), debiera saber de qué trata el análisis de sistemas. Los sistemas computacionales de diversos tipos afectan a toda clase de personas. Aun si jamás piensa construir un sistema computacional ni hacer que le diseñen uno, es inevitable que vaya a hacer uso de sistemas computacionales para sus finanzas, su educación, su interacción con las oficinas de impuestos y del seguro social, y para casi cualquier interacción que pueda llegar a tener con la sociedad moderna. Como lo afirma John Gal en su obra *Systematics* [Gall, 1977].

> Nadie puede, hoy en día, evitar el contacto con los sistemas. Los sistemas están en todas partes: sistemas grandes, sistemas pequeños, sistemas mecánicos y electrónicos, y aquellos sistemas especiales que consisten en asociaciones organizadas de personas. En defensa propia, debemos aprender a vivir con los sistemas, a *controlarlos* antes de que ellos nos controlen a *nosotros*. Como le dijo Humpty Dumpty a Alicia (aunque en otro contexto): "La pregunta es: quién ha de ser el amo, eso es todo."

Para enfatizar esto aún más, tenga en mente que la industria de las computadoras representó aproximadamente el 8% del producto interno bruto (PIB) de los Estados Unidos en 1985; para 1990 se espera que represente el 15% del PIB [3] Casi todos los productos fabricados hoy por las empresas americanas involucran una o más computadoras, y casi cualquier servicio ofrecido al mercado por las empresas norteamericanas se basa en, o es controlado por, un sistema computacional.

1.3 QUE HARA ESTE LIBRO POR USTED

Como ya habrá adivinado, uno de los principales propósitos de este libro es enseñarle más acerca del análisis de sistemas: qué es y cómo se las arregla uno para llevarlo a cabo. Pero aún hay más: mi verdadero propósito es *emocionarlo*, entusiasmarlo tanto con empezar a practicar el análisis de sistemas que querrá acabar este libro a toda prisa y comenzar a trabajar en su primer proyecto. Seymour Papert hace la siguiente remembranza en su obra *Mindstorms* [Papert, 1980],

> Encontré un placer particular en sistemas tales como la caja de velocidades diferencial, que no sigue una simple cadena lineal de causalidad, puesto que el movimiento en el eje de transmisión puede distribuirse de muchas formas diferentes a las dos ruedas, dependiendo de la resistencia que se encuentre. Recuerdo con mucha claridad mi emoción al descubrir que un sistema podía ser válido y completamente comprensible sin ser rígidamente determinístico.

Y como dijo Sir Stanley Eddington [Eddington, 1987],

> Hemos encontrado que donde la ciencia más ha avanzado, la mente no ha hecho sino recuperar de la naturaleza lo que puso en ella.

3 Para más detalles sobre esto, así como para otros análisis sobre el impacto de las computadoras en la sociedad, véase *Nations at Risk* [Yourdon, 1986].

> Hemos encontrado una extraña huella en las playas de lo desconocido. Hemos elaborado profundas teorías, una tras otra, para explicar su origen. Finalmente hemos tenido éxito en reconstruir la criatura que hizo la huella. Y, ¡sorpresa!, somos nosotros.

Otro propósito de este libro es hacerle entender y apreciar que vivimos en un mundo de sistemas, y de sistemas dentro de sistemas, que son parte de otros aún mayores. Por tanto, todo lo que hacemos en nuestras vidas personales y profesionales tiene impacto (a menudo inesperado y no anticipado) sobre los diversos sistemas de los cuales formamos parte. Este enfoque de "pensar en sistemas" no sólo es vital para los analistas profesionales sino para todos los miembros de nuestra sociedad moderna.

Desgraciadamente, este libro no lo podrá convertir en un analista de sistemas con experiencia, como tampoco un libro de teoría musical puede convertirlo en pianista experimentado. Para cuando termine este libro, estará armado con una tremenda cantidad de información que lo ayudará a desarrollar *modelos* precisos de sistemas complejos, y conocerá paso a paso las técnicas para efectuar un esfuerzo de análisis de sistemas. Sin embargo, necesitará aún una gran cantidad de trabajo concreto para poder desarrollar habilidades: cómo entrevistar a una variedad de diferentes usuarios para entender la verdadera esencia de un sistema; cómo presentar los resultados de su trabajo de análisis de sistemas para que todo mundo pueda darse cuenta de los verdaderos costos y beneficios de desarrollar un nuevo sistema; cómo diferenciar problemas de síntomas. Como señaló Barry Bohem en su obra clásica, *Software Engineering Economics* [Bohem, 1981]:

> Cada uno de nosotros como ingeniero de software "individual" tiene la oportunidad de causar un impacto positivo en la sociedad, simplemente volviéndonos más sensibles a las profundas implicaciones que nuestro trabajo tiene en las relaciones humanas, y al incorporar esta sensibilidad a nuestros productos y diseños de software.
>
> Hacer esto bien requiere algo de práctica, así como aprender a balancear las cuestiones de relaciones humanas con la programación y con los asuntos económicos cuantitativos. La gran cosa que hay que recordar al hacerlo es mantener claras nuestras prioridades entre la programación, los presupuestos y las relaciones humanas.

1.4 ORGANIZACION DEL LIBRO

Este libro está organizado en cuatro partes principales, seguidas de una serie de apéndices. La parte I sirve de introducción a todo el libro y éste comienza, en el capítulo 2, por una introducción al concepto de sistemas y la naturaleza del análisis de sistemas; en este capítulo veremos que los sistemas de información usualmente están compuestos por personas, hardware, software (programas de computadora), procedimientos, datos e información. El capítulo 3 describe a las personas que normalmente están involucradas en el desarrollo de un sistema moderno de informa-

ción: los usuarios, los administradores, el personal de operaciones, los miembros del grupo de control de calidad, etc., y el papel especial y las responsabilidades del analista de sistemas. El Capítulo 4 introduce las herramientas de modelado que el analista de sistemas utiliza, incluyendo diagramas de flujo de datos, diagramas de entidad-relación y diagramas de transición de estados. El capítulo 5 presenta los procedimientos (o la metodología) que el analista sigue cuando desarrolla un sistema.

Aunque usted crea conocer muchas de estas cosas ya, hay algunos capítulos en la parte I que es importante que lea, porque definen el tono del resto del libro. El capítulo 2, por ejemplo, presenta y discute los axiomas y principios fundamentales que esperaríamos encontrar en todo el trabajo de análisis de sistemas: el desarrollo de modelos de sistemas, la noción de iteración, y la noción de partición de arriba-abajo. Y el capítulo 6 delinea las principales cuestiones que enfrenta el analista de sistemas hoy: cuestiones de productividad, calidad de sistemas, mantenimiento, y utilización estratégica de información. Finalmente, el capítulo 7 resume los principales cambios que han sucedido en el campo del análisis de sistemas en los últimos diez años.

La parte II trata de las herramientas de modelado de sistemas con detalle. Los capítulos individuales cubren temas de diagramas de flujo de datos (capítulo 9), diccionarios de datos (capítulo 10), especificación de procesos (capítulo 11), diagramas de entidad-relación (capítulo 12), y diagramas de transición de estados (capítulo 13). Los capítulos 15 y 16 presentan otras herramientas de modelado que utilizan los analistas cuando estudian un sistema: diagramas PERT, diagramas de Gantt, diagramas de flujo, diagramas HIPO, diagramas de estructura, etc. Como podremos ver, estas herramientas de modelado permitirán enfocarnos selectivamente a los aspectos individuales de un sistema cuyas características es importante entender: las funciones que el sistema debe desempeñar, los datos que debe manejar y su comportamiento en el tiempo.

Aun cuando usted nunca vea una computadora al terminar de leer este libro, as herramientas de modelado de la parte II pueden serle útiles para modelar (o describir o imaginarse) prácticamente *cualquier* tipo de sistema: biológicos, de negocios, ecosistemas, sistemas de manufactura, políticos, de flujo de materiales, etc. Vivimos en un mundo de sistemas, y la mayor parte de nuestra vida cotidiana se emplea tratando de comprender y trabajar con los múltiples sistemas con los cuales entramos en contacto; las herramientas de modelado son de enorme ayuda en este aspecto.

La parte III se refiere al proceso de análisis de sistemas, es decir, los pasos que un analista lleva a cabo cuando construye el modelo de un sistema. Aquí también, la información que obtendrá será de utilidad independientemente de su profesión; el material definitivamente se dirige hacia la construcción de sistemas de información automatizados. Veremos que el proceso, o metodología, para construir un sistema involucra el desarrollo de diversos tipos de modelos, el último de los cuales será el producto del análisis de sistemas. En muchas empresas este producto se conoce como "especificación funcional", o "documento de requerimientos de negocios", o "diseño de sistemas de negocios". Independientemente de su nombre, es el material para los responsables de la *construcción* misma del sistema, es decir, de diseñar la arquitectura general de las computadoras y su software y, finalmente, de escribir y probar los programas.

Esto nos lleva a la parte IV: la vida después del análisis de sistemas. Exploraremos el paso desde el análisis de sistemas hasta el diseño y discutiremos brevemente los detalles finales de la programación y la prueba. Dado que la mayoría de los sistemas de información automatizados tienen una vida media de varios años (y a menudo de varias décadas), también discutiremos el tema del mantenimiento en el capítulo 24; pero nuestra principal preocupación no será la *programación* para el mantenimiento, sino el mantenimiento del producto del análisis de sistemas. El último capítulo trata del futuro: los cambios evolutivos en el campo del análisis de sistemas que podemos esperar ver durante los años 90 y el próximo siglo.

Los Apéndices que se encuentran al final del libro tratan cuestiones separadas que pueden o no llegar a afectarlo cuando comience a trabajar como analista de sistemas. El Apéndice A, por ejemplo, maneja el tema de las estaciones de trabajo basadas en PCs para el análisis de sistemas, a lo cual pocos analistas tenían acceso a comienzo de la década de los 80, pero que se han vuelto cada vez más comunes en los años 90. El Apéndice B expone las fórmulas de estimación y la métrica utilizadas para calcular el tamaño, duración y costo de un proyecto. El Apéndice C analiza los cálculos de costo-beneficio. El Apéndice D cubre el tema de los recorridos y las inspecciones (*walkthroughs*), que a menudo se utilizan para revisar los productos técnicos del análisis de sistemas. El Apéndice E analiza las técnicas de entrevista y acopio de datos, sobre todo para aquellas entrevistas que se llevan a cabo entre el usuario y el analista de sistemas. Todo esto se ha acomodado en apéndices para que el analista experimentado pueda saltarse fácilmente los que considera prescindibles; los estudiantes principiantes pueden recurrir a los apéndices cuando sea conveniente para cubrir temas que seguramente surgirán durante los proyectos reales.

Los Apéndices F y G presentan dos casos de estudio: uno es un sistema orientado a los negocios, y el otro es un sistema de tiempo real. Si usted es un estudiante que inicia, al final de cada capítulo deberá referirse a estos casos de estudio para ver cómo pueden aplicarse a situaciones reales los principios recién aprendidos. De hecho, debería leer la introducción y antecedentes de cada caso de estudio ahora, para familiarizarse con la naturaleza de cada aplicación.

Cada capítulo tiene preguntas y ejercicios que lo ayudarán a revisar lo que ha aprendido. Algunos ejercicios están etiquetados como "proyectos de investigación", lo cual significa que se enfocan a hechos que no están cubiertos directamente en el capítulo, pero que son pertinentes en el mundo real del análisis de sistemas. Algu-

nas de las preguntas están enfocadas a discusiones en clase; no hay respuestas correctas o incorrectas, aunque sí hay respuestas que son más fáciles de defender que otras.

Terminemos con las introducciones y empecemos. Comenzaremos por hablar de la naturaleza de los sistemas.

REFERENCIAS

1. Tom DeMarco. *Structured Analysis and Systems Specification.* Englewood Cliffs, N.J.: Prentice-Hall, 1979, página 6.
2. John Gall, *Systemantics.* New York: Quadrangle/The New York Times Book Company, 1977, pág. xiii.
3. Barry Bohem, *Software Engineering Economics.* Englewood Cliffs, N.J.: Prentice-Hall, 1981.
4. Seymour Papert, *Mindstorms.* New York: Basic Books, 1980.
5. Edward Yourdon, *Nations at Risk.* Englewood Cliffs, N.J.: YOURDON Press, 1986.
6. Sir Arthur Stanley Eddington, Space, *Time and Gravitation An Outline of the General Theory.* London: Cambridge University Press, 1987.

PREGUNTAS Y EJERCICIOS

1. Explique cómo puede serle útil el análisis ce sistemas en su empleo o profesión, aun si no planea convertirse en programador o analista.
2. Proyecto de investigación: ¿Cuántas personas hay empleadas como analistas de sistemas en el país hoy en día? ¿Cuál es su salario prcmedio? ¿Cuál es su nivel promedio de educación?
3. Proyecto de investigación: ¿Existe escasez de programadores y analistas de sistemas? Trate de encontrar estadísticas de industrias o del Gobierno (por ejemplo de la Secretaría de Comercio o de alguna fundación científica nacional) que predigan los requerimientos nacionales de analistas de sistemas durante los próximos l0 años.
4. Dé l0 ejemplos de sistemas con los que trata o con los cuales interactúa en su vida cotidiana.
5. ¿Estaría estudiando análisis de sistemas si no tuviera la necesidad de hacerlo? Si su respuesta es "No", explique por qué piensa que el material no será útil o pertinente. Encuentre alguna otra persona que estudie este mismo material y tenga un debate constructivo respecto a la utilidad general del análisis de sistemas.
6. ¿Cree usted importante que las personas que no utilizan computadcras (o que no tienen una profesión relacionada) estudien análisis de sistemas? ¿Qué tan conocedores cree que deban ser en este tema?
7. ¿Por qué es probable que el análisis de sistemas sea más interesante que la programación? ¿Está de acuerdo con este punto de vista?
8. ¿Qué cosas debe aprender un analista de sistemas aparte de las habilidades técnicas para construir modelos de sistemas?
9. ¿Por qué pueden ser útiles para estudiar sistemas no computacionales las herramientas de modelado presentadas en este libro?

2 LA NATURALEZA DE LOS SISTEMAS

Finalmente, pondremos al Sol mismo en el centro del Universo. Todo esto lo sugiere la sistemática procesión de sucesos, así como la armonía del Universo entero, si tan sólo encaráramos los hechos, como se dice, "con ambos ojos abiertos".

Nicolás Copérnico
De Revolutionibus Orbium Coelestium, 1543.

En este capítulo se aprenderá:

1. Qué es la definición de un sistema.
2. La diferencia entre sistemas naturales y sistemas hechos por el hombre.
3. Los 19 subsistemas principales encontrados en todos los sistemas vivientes.
4. Las cinco razones principales por las que algunos sistemas no deben automatizarse.
5. Los cinco componentes principales de un sistema de información automatizado típico.
6. La definición y características de varios tipos específicos de sistemas.
7. La definición de los principios generales de sistemas y tres ejemplos de ellos.

No podemos decir mucho acerca del análisis de sistemas hasta que tengamos una idea clara de lo que es un *sistema:* éste es el propósito de este capítulo. Como veremos, existe una definición "oficial" del término en el diccionario, que parecerá algo abstracta. Sin embargo hay muchos usos comunes del término que le serán familiares y existen muchos tipos comunes de sistemas con los que tenemos contacto todos los días.

Se puede estar preguntando "¿Y qué?". Es importante estar familiarizado con diferentes tipos de sistemas debido, por lo menos a dos razones. Primero, aunque su trabajo como analista de sistemas probablemente se enfoque en algún tipo de sistema de información computarizado, generalmente formará parte de un sistema mayor. Así, usted tal vez pueda estar haciendo un sistema de nómina que forma parte de un sistema de recursos humanos, que a su vez forma parte de alguna organización (que en sí, es un sistema), que a su vez es parte de un sistema económico mayor, etc., o bien puede estar haciendo un sistema de control de procesos que es parte de una refinería química o un sistema operativo que sea parte de un "paquete" de software suministrado por el proveedor. Así, para que su sistema tenga éxito, debe entender los demás sistemas con los que va a interactuar.

Muchos de los sistemas computarizados que construimos son reemplazos o nuevas versiones de sistemas no computarizados que ya existen. También, la mayoría de los sistemas computarizados interactúan o tienen interfases con una variedad de sistemas existentes (algunos de los cuales pueden estar computarizados y otros no). Para que nuestro nuevo sistema sea exitoso, debemos entender con razonable detalle cómo se comporta el sistema actual.

Segundo, aunque muchos tipos de sistemas parezcan bastante diferentes, resulta que tienen similitud: existen principios, teorías y filosofías comunes que se aplican muy bien, prácticamente, a *todos* los tipos de sistemas. Por ello, frecuentemente podemos aplicar lo que hemos aprendido acerca de otros sistemas —basándonos en nuestra experiencia diaria, y en la de diversos tipos de científicos e ingenieros— a los sistemas que construimos en el campo de la computación. Por ejemplo, uno de los principios importantes de sistemas, primeramente observado en el campo de la biología, es conocido como la ley de la especialización: entre más adaptado se encuentra un organismo a un ambiente específico, más difícil le será adaptarse a un ambiente diferente. Esto ayuda a explicar la desaparición de los dinosaurios cuando cambió drásticamente el clima de la tierra[1], y también ayuda a los analistas a entender que si optimizan un sistema computarizado para obtener la má-

1 Los paleontólogos aún están discutiendo esto: unos piensan que los dinosaurios desaparecieron en el curso de un periodo relativamente breve, tras el impacto de un gran meteorito con la Tierra, que originó una nube de polvo tan densa que mató a la mayoría de las plantas. Otros argumentan que el cambio fue mucho más gradual, ocurrido en el transcurso de casi un millón de años. De cualquier modo, los dinosaurios estaban altamente adaptados a un determinado tipo de ambiente y no pudieron adaptarse a otro.

xima ventaja de un procesador, de un lenguaje de programación y de un sistema administrador de base de datos, probablemente tendrán grandes dificultades para adaptar dicho sistema de forma que se ejecute en un procesador diferente o con un sistema de administración de base de datos diferente.[2]

De aquí que si comprendemos algo acerca de la *teoría general de sistemas* nos ayudará a entender mejor los sistemas computarizados (automatizados) de información. Esto se vuelve cada vez más importante, pues se desea construir sistemas *estables* y confiables, que funcionen bien en nuestra compleja sociedad, y existen, desde luego, muchos ejemplos de sistemas no computarizados que han sobrevivido durante millones de años: la humilde cucaracha probablemente durará más que cualquier sistema computarizado que se haya podido construir, y más que toda la humanidad también.

Así pues, empecemos con una definición del término básico: sistema. Todo libro de texto que cubra algún aspecto de los sistemas contiene tal definición; he escogido el New *Collegiate Dictionary* de Webster[3], que tiene varias definiciones:

1. Grupo de elementos interdependientes o que interactúan regularmente formando un todo <un ~ sistema numérico >: como

   a.
      (1) Un grupo de cuerpos que interactúan bajo la influencia de fuerzas relacionadas <un ~ gravitacional>
      (2) Un conjunto de sustancias que tiendan al equilibrio <un ~ termodinámico>

   b.
      (1) Un grupo de órganos del cuerpo que juntos llevan a cabo una o más funciones vitales <el ~ digestivo>
      (2) El cuerpo, considerado como una unidad funcional

   c. Un grupo de fuerzas u objetos naturales <un ~ de ríos>

   d. Un grupo de aparatos o una organización que forma una red, especialmente para distribuir algo o para servir a un propósito común <un ~ telefónico>, <un ~ de calefacción>, <un ~ de autopistas>, <un ~ de proceso de datos>

2 También puede ayudar al analista de sistemas a entender el fenómeno del usuario que comúnmente suele realizar actividades tan especializadas que no hay manera de cambiarlas aunque sean computarizadas. Y le recuerda que si llega a desarrollar un sistema computarizado altamente especializado para la aplicación *actual* que quiere el usuario, será muy difícil adaptarlo luego, cuando cambien o evolucionen los requerimientos de éste (y el ambiente en el cual trabaja).

3 *New Collegiate Dictionary* de Webster, Springfield, Mass.: G.& C. Merriam Company, 1977.

2. Juego organizado de doctrinas, ideas o principios, usualmente con la intención de explicar el acomodo o trabajo de un todo sistemático <el ~ newtoniano de la mecánica>

3. a. un procedimiento organizado o establecido <el ~ de mecanografía al tacto>

   b. una manera de clasificar, simbolizar o esquematizar <un ~ taxonómico> <el ~ decimal>.

4. patrón o arreglo armonioso: ORDEN

5. una sociedad organizada o situación social considerada como anuladora: ORDEN ESTABLECIDO

2.1 TIPOS COMUNES DE SISTEMAS

Como podemos ver de la definición anterior, existen muchos tipos diferentes de sistemas; de hecho, casi todo aquello con lo cual entramos en contacto durante nuestra vida cotidiana es un sistema o bien parte de un sistema (o ambas cosas).

¿Significa esto que debemos estudiar todo tipo de sistemas o intentar convertirnos en expertos en sistemas sociales, biológicos y computacionales? ¡Para nada! Sin embargo, es útil organizar los diferentes tipos de sistemas en categorías. Son posibles muchas diferentes formas de categorizar; de hecho, la definición obtenida del diccionario muestra una categorización. Dado que nuestro objetivo son los sistemas computacionales, empezaremos por dividir todos los sistemas en dos categorías: *sistemas naturales* y *sistemas hechos por el hombre.*

2.2 SISTEMAS NATURALES

La gran mayoría de los sistemas no están hechos por el hombre: existen en la naturaleza y sirven a sus propios fines. Es conveniente dividir los sistemas naturales en dos subcategorías básicas: *sistemas físicos* y *sistemas vivientes* Los sistemas físicos incluyen ejemplos tan variados como:

- Sistemas estelares: galaxias, sistemas solares, etcétera.
- Sistemas geológicos: ríos, cordilleras, etcétera.
- Sistemas moleculares: organizaciones complejas de átomos.

Es interesante estudiar los sistemas físicos pues, como humanos entrometidos que somos, a veces queremos modificarlos. También desarrollamos una variedad de sistemas hechos por el hombre, incluyendo sistemas computacionales, que deben interactuar armónicamente con los sistemas físicos; por tanto, suele ser importante modelar dichos sistemas para asegurarnos que los comprendemos lo más completamente posible.

Los *sistemas vivientes*, desde luego, comprenden toda la gama de animales y plantas que nos rodean, al igual que a la raza humana. Y, como lo menciona James Miller en su monumental obra, *Sistemas Vivientes* [Miller, 1978], esta categoría también comprende jerarquías de organismos vivientes individuales, por ejemplo hierbas, manadas, tribus, grupos sociales, compañías y naciones.

El estudio de los sistemas vivientes es una carrera en sí; una breve lectura de la obra de Miller mostrará lo enorme que es dicho tema. El propósito de este libro no es estudiar los sistemas vivientes *per se*; pero algunas de sus propiedades y características familiares pueden utilizarse para ayudar a ilustrar y entender mejor los sistemas hechos por el hombre. A menudo utilizamos una analogía para entender mejor algo con lo cual no estamos familiarizados; entre las más elocuentes de las analogías entre sistemas vivientes y sistemas organizacionales y de negocios, tenemos las obras de Stafford Beer, *Brain of the Firm* [Beer, 1972] y *The Heart of Enterprise* [Beer, 1978].

Una analogía más elaborada puede obtenerse de la categorización hecha por Miller de los 19 subsistemas críticos de todos los sistemas vivientes. Miller argumenta que los sistemas vivos, sean estos de nivel celular, de órgano, de organismo, de grupo, de organización, de sociedad o de sistema supranacional, contienen los siguientes subsistemas:

- El *reproductor*, que es capaz de dar origen a otros sistemas similares a aquel en el cual se encuentra. En una organización de negocios, pudiera ser una división de planeación de instalaciones que hace nuevas plantas y construye oficinas regionales nuevas.
- La *frontera*, que mantiene unidos a los componentes que conforman el sistema, los protege de tensiones ambientales y excluye o permite la entrada de diversos tipos de materia-energía e información. En una organización de negocios, esto pudiera constituir la planta misma (edificio de oficinas, fábrica, etc.) y los guardias u otro personal de seguridad que evitan el ingreso de intrusos indeseables.
- El *inyector*, que transporta la materia-energía a través de la frontera del sistema desde el medio ambiente. En una organización de negocios, éste pudiera ser el departamento de compras o recepción, que introduce la materia prima, los materiales de oficina, etc. o pudiera ser el departamento de pedidos, que recibe pedidos verbales o por escrito de los servicios o productos de la organización.
- El *distribuidor*, que trae material desde el exterior del sistema y lo reparte desde sus subsistemas a cada componente. En una organización de negocios, pudiera estar conformado por las líneas telefónicas, correo electrónico, mensajeros, bandas y una variedad de otros mecanismos.

- El *convertidor*, que cambia ciertos materiales que ingresan al sistema a formas mas útiles para los procesos especiales de dicho sistema particular. Nuevamente, se puede imaginar un buen número de ejemplos de esto en una organización de negocios típica.
- El *productor*, que forma asociaciones estables durables por periodos significativos con la materia-energía que ingresa al sistema o que egresa de su convertidor. Estos materiales sintetizados pueden servir para crecimiento, reparación de daños o reposición de componentes del sistema, o bien para proveer energía para mover o constituir los productos o la información de mercado a su suprasistema.
- El subsistema de *almacenamiento de materia-energía*, que retiene en el sistema, durante diferentes periodos, depósitos de diversos tipos de materia-energía.
- El *expulsor*, que transmite materia-energía hacia el exterior del sistema en forma de desechos o de productos.
- El *motor*, que mueve el sistema o a sus partes en relación con todo o parte del medio ambiente, o bien que mueve a los componentes del ambiente.
- El *soporte*, que mantiene las relaciones espaciales apropiadas entre los componentes del sistema, de manera que puedan interactuar sin ser un lastre o estorbo entre ellos.
- El *transductor* de entrada, que trae señales portadoras de información al sistema, transformándolas en otras formas de materia- energía adecuadas para su transmisión al interior.
- El *transductor interno*, que recibe de otros subsistemas o componentes del sistema señales que portan información acerca de alteraciones significativas en dichos subsistemas o componentes, transformándolos en otras formas de materia-energía transmisibles en su interior.
- El *canal y la red*, que están compuestos por una sola ruta en el espacio físico, o bien por múltiples rutas interconectadas, mediante las cuales las señales portadoras de información se transmiten a todas las partes del sistema.
- El *decodificador*, que altera las claves de información que le es introducida por medio del transductor de entrada o del transductor interno, para dejar una clave privada que pueda ser utilizada internamente por el sistema.
- El *asociador*, que lleva a cabo la primera etapa del proceso de aprendizaje, formando asociaciones duraderas entre elementos de información dentro del sistema.

- La *memoria*, que lleva a cabo la segunda etapa del proceso de aprendizaje, almacenando diversos tipos de información en el sistema durante diferentes periodos.
- El que *decide*, que recibe información de todos los demás subsistemas y les transmite información que sirve para controlar al sistema completo.
- El *codificador*, que altera la clave de información que se le introduce desde otros subsistemas procesadores de información, convirtiéndola, de una clave privada utilizada internamente por el sistema, en una clave pública que puede ser interpretada por otros sistemas en su medio ambiente.
- El *transductor de salida*, que emite señales portadoras de información desde el sistema, transformando los marcadores dentro del sistema en otras formas de materia-energía que pueden ser transmitidas por medio de canales en el medio ambiente del sistema.

Las figuras 2.1 (a) y 2.1 (b) muestran un ejemplo de los 19 principales subsistemas de un equipo de comunicaciones en un crucero transoceánico moderno; las figuras 2.2 (a) y 2.2 (b) muestran los principales subsistemas del crucero mismo; y las figuras 2.3 (a) y 2.3 (b) muestran los principales subsistemas de toda Holanda. Vale la pena estudiarlos, pues ilustran el hecho de que si se observa cualquier sistema que tiene componentes vivientes, es posible localizar los principales subsistemas.

Tenga en mente que muchos sistemas hechos por el hombre (y sistemas automatizados) interactúan con sistemas vivientes; por ejemplo, los marcapasos computarizados interactúan con el corazón humano. En algunos casos, se diseñan sistemas automatizados para reemplazar a sistemas vivos; y en otros, los investigadores consideran a los sistemas vivos (conocidos como computadoras orgánicas) como componentes de sistemas automatizados. Véanse [Hall, 1983], [DeYoung, 1983], [Shrady, 1985] y [Olmos, 1984] para análisis de este punto de vista. Los sistemas vivientes y los sistemas hechos por el hombre a menudo forman parte de un metasistema mayor, y entre mas entendamos acerca de ambos, mejores analistas de sistemas seremos.

2.3 SISTEMAS HECHOS POR EL HOMBRE

Como pudimos apreciar de la definición que se encuentra al comienzo de este capítulo, un buen número de sistemas son construidos, organizados y mantenidos por humanos, e incluyen:

- Sistemas sociales: organizaciones de leyes, doctrinas, costumbres, etcétera.
- Una colección organizada y disciplinada de ideas: el sistema decimal Dewey de organización de libros en bibliotecas, el sistema de reducción de los cuida-kilos, etcétera.

- Sistemas de transporte: redes de carreteras, canales, aerolíneas, buques cargueros, etcétera.
- Sistemas de comunicación: teléfono, télex, señales de humo, señales de mano utilizadas por los corredores de bolsa, etcétera.
- Sistemas de manufactura: fábricas, líneas de ensamblado, etcétera.
- Sistemas financieros: contabilidad, inventarios, libro mayor, bolsa de valores etcétera.

En la actualidad, la mayoría de estos sistemas incluyen las computadoras; de hecho, muchos no podrían sobrevivir sin ellas. Sin embargo, es igualmente importante señalar que dichos sistemas existían antes de que hubiera computadoras; de hecho, algunos sistemas continúan por completo sin computarizar y podrían permanecer así durante muchas décadas más. Otros contienen a la computadora como componente, pero también incluyen uno o más componentes no computarizados (o manuales).

Considérese, por ejemplo, la frase común, "Juan tiene un sistema para llevar a cabo su trabajo" o "Vaya que María tiene una manera sistemática de hacer su trabajo". Tales frases no necesariamente sugieren que María ha computarizado su trabajo o que Juan ha utilizado algunos de los instrumentos formales de modelación discutidos en los capítulos 9 y 10 para documentar (o modelar) cómo propone hacer su trabajo. Pero ciertamente las frases implican que Juan y María han dividido su trabajo en una serie de pasos definidos, la suma acumulativa de los cuales logrará algún propósito general.

El que un sistema de factura humana deba o no ser computarizado es una cuestión que discutiremos a lo largo de este libro; *no es algo que se daba dar por hecho*. Como analista de sistemas, usted naturalmente supondrá que todo sistema con el que se encuentre deberá computarizarse, y el cliente o usuario (el dueño del sistema en cuestión) con quien usted interactúa generalmente supondrá tal predisposición. Como veremos en capítulos posteriores, su labor primaria como analista de sistemas será analizar o estudiar el sistema para determinar su esencia: su comportamiento requerido, *independientemente* de la tecnología utilizada para implantar el sistema[4]. En la mayoría de los casos, podremos determinar si tiene sentido utilizar una computadora para llevar a cabo las funciones del sistema sólo tras haber modelado su comportamiento esencial.

¿Por qué no debieran automatizarse algunos sistemas de procesamiento de información? Puede haber muchas razones; he aquí algunas de las más comunes:

- *Costo*. Puede ser más barato continuar llevando a cabo las funciones y almacenando la información del sistema en forma manual. No siempre es

4 Se discutirán los *modelos esenciales* y la *esencia* de un sistema en el capítulo 17.

cierto que las computadoras sean más rápidas y económicas que el método "anticuado".

- *Conveniencia.* Un sistema automatizado puede ocupar demasiado espacio, hacer demasiado ruido, generar demasiado calor o consumir demasiada electricidad, o bien, en general, ser una molestia. Esto va dejando de ser así con la creciente influencia de los microprocesadores, pero sigue siendo un factor a considerar.
- *Seguridad.* Si el sistema de información mantiene datos confidenciales, el usuario pudiera no creer que el sistema automatizado sea lo suficientemente seguro; tal vez desee mantener la información físicamente protegida y bajo llave.
- Facilidad de *mantenimiento.* El usuario pudiera argumentar que un sistema de información computarizada sería costeable, excepto por el hecho de que no hay ningún miembro del personal que pudiera encargarse del mantenimiento del hardware y o el software de la computadora, de manera que nadie podría reparar el sistema si éste sufriera un desperfecto, ni habría quien pudiera efectuar cambios o mejoras.
- *Políticas.* La comunidad usuaria pudiera recelar que las computadoras amenazan con privarla de sus empleos o con volver aburridos o "mecánicos" sus trabajos o con una docena de otras razones que el analista de sistemas podría considerar irracionales. Pero dado que se trata del sistema del usuario, sus recelos son de primera importancia. Si no desean tener un sistema automatizado, harán todo lo posible por lograr que falle si se les quiere imponer.

2.4 SISTEMAS AUTOMATIZADOS

La mayor parte de este libro se concentrará en los sistemas *automatizados*, es decir, en sistemas hechos por el hombre que interactúan con o son controlados por una o más computadoras. Sin duda, usted ha visto muchos ejemplos diferentes de sistemas automatizados en su vida cotidiana: parece ser que casi cada aspecto de nuestra sociedad moderna está computarizado. Como resultado, podemos distinguir muchos tipos diferentes de sistemas automatizados.

Aunque hay diferentes tipos de sistemas automatizados, todos tienden a tener componentes en común.

- El *hardware de la computadora*: los procesadores, los discos, terminales, impresoras, unidades de cinta magnética, etcétera.
- El *software de la computadora*: los programas de sistemas tales como sistemas operativos, sistemas de bases de datos programas de control de

Subsistemas que procesan tanto materia-energía como información: frontera de a bordo (Bo), la pared del cuarto de radio (artefacto).

Subsistemas que procesan la materia-energía: El ingestor (IN), la camarera que trae comida al cuarto de radio desde la cocina del barco; el distribuidor (DI), el camarero que reparte la comida a los miembros del equipo de comunicaciones; el convertidor (CO), el camarero que parte el pan y corta la carne y el queso para los sandwiches; el productor (PR), el camarero que hace los sandwiches y el café; el almacenista de materia-energía (MS), el camarero que almacena diversos tipos de artefactos, incluyendo alimentos en el refrigerador, sacos y gorras de los miembros del equipo en el clóset, mantas y almohadas en el armario y herramientas y equipo en una gaveta; el expulsor (EX), el camarero que se lleva los utensilios usados, la basura y otros desechos del cuarto de radio; el soporte o sustentador (SU), el piso, las paredes, el techo, y los muebles del cuarto de radio (artefactos).

Subsistemas que procesan información: el transductor de entrada o introductor (it), el operador de radio que recibe los mensajes; el transductor interno (in), el capataz de turno que informa al oficial de señales en jefe sobre la eficiencia y ánimo de los miembros del equipo en su turno; el canal y la red (cn), todos los miembros del grupo que se intercomunican por medio del habla que se transmite a través del aire del cuarto de radio; el decodificador (dc), el operador de radio que transcribe a la lengua propia los mensajes recibidos en clave Morse; la memoria (me), la secretaria que lleva el registro de todos los mensajes recibidos y transmitidos; el decididor (de), el oficial de señales en jefe, que dirige al equipo de comunicaciones; el codificador en Morse (en), el operador de radio que codifica de la lengua propia al código Morse los mensajes; el transductor de salida (ot), el operador de radio que transmite los mensajes por esta vía.

figura 2.1(a): **Subsistemas de un equipo de comunicaciones de un crucero**

telecomunicaciones, además de los programas de aplicación que llevan a cabo las funciones deseadas por el usuario.

- Las *personas*: los que operan el sistema, los que proveen su material de entrada y consumen su material de salida, y los que proveen actividades de procesamiento manual en un sistema.
- Los *datos*: la información que el sistema recuerda durante un periodo.
- Los *procedimientos*: las políticas formales e instrucciones de operación del sistema.

Una manera de ordenar por categorías los sistemas automatizados es por su *aplicación*: sistemas de manufactura, sistemas de contabilidad, sistemas de defensa

Figura 2.1(b): **Subsistemas de un equipo de comunicaciones de un crucero**

militar, etc. Sin embargo, esto no resulta muy útil, pues las técnicas que discutiremos en este libro para analizar, modelar, diseñar e implantar sistemas automatizados generalmente son las mismas, independientemente de su aplicación.[5]

5 Sin embargo, cada aplicación tiene su vocabulario, cultura y procedimientos propios. El usuario por lo general espera que el analista de sistemas sepa algo acerca de los detalles, política y procedimientos de la aplicación, para no tener que explicarle todo desde el principio. Por lo tanto, si va a desempeñar el trabajo de analista de sistemas en un banco, probablemente le sea útil aprender cuanto pueda acerca de la banca. No es camino de un solo sentido: los banqueros aprenden cada día más acerca de la tecnología de los sistemas de información.

Subsistemas que procesan tanto materia-energía como información: el reproductor (Re), los representantes de la corporación propietaria; la frontera de a bordo (Bo), el casco del barco y el personal que lo protege y le da mantenimiento.

Subsistemas que procesan la materia-energía: el ingestor (IN), la escotilla que conduce a la bodega del barco y el personal que ayuda a los pasajeros, a abordar y que estiba el equipaje y el cargamento a bordo del barco; el distribuidor (DI), los pasillos, puentes y escaleras, y los camareros, meseros y mozos que llevan alimentos, bebidas, equipaje y otros diversos tipos de materia-energía por dichos pasillos, además de los pasajeros que por ellos se desplazan de un lado a otro del barco; el convertidor (CO), el personal de la galera que limpia las verduras y prepara otros comestibles para cocinarlos; el productor (PR), los que cocinan la comida y los panaderos que laboran en la galera del barco; el almacenador de materia-energía (MS), la bodega del barco y los tanques de combustible, y el personal responsable de ellos; el expulsor (EX), la chimenea para desechos gaseosos, salidas para la basura y drenaje para los desechos líquidos y sólidos, y el personal de operaciones responsable de asegurar que los desechos sean eliminados adecuadamente; el motor (MO), los motores del barco, su timón, hélices y todo el casco del barco, que mueven pasajeros, tripulación y cargamento a través del mar, junto con los ingenieros responsables de lograr este movimiento; el soporte (SU), el casco, los flancos, las paredes y los puentes del barco, y el personal que los mantiene.

Subsistemas que procesan la información: el transductor de entrada (in), el operador de radio y otros miembros del equipo de comunicaciones que reciben mensajes para el barco; el transductor interno (in), el oficial que informa al oficial de mando sobre el estado de los diversos componentes que forman el barco; el canal y la red (cn), el aire que rodea a los oficiales en el puente, a través del cual transmiten y reciben mensajes; el decodificador (dc), el operador de radio en el equipo de comunicaciones que descifra los mensajes en clave Morse, dejándolos en lengua propia después de ser recibidos; la memoria (me), los cuadernos de bitácora de travesías pasadas, mapas marítimos y el personal que los consulta en el cuarto de mapas; el decididor (de), el capitán y otros oficiales de a bordo; el codificador en clave Morse (en), el operador de radio del equipo de comunicaciones que traduce los mensajes de la lengua propia al código Morse para su transmisión; el transductor de salida (ot), el operador de radio y otros miembros del equipo de comunicaciones que transmiten mensajes desde el barco.

Figura 2.2(a): **Principales subsistemas de un crucero**

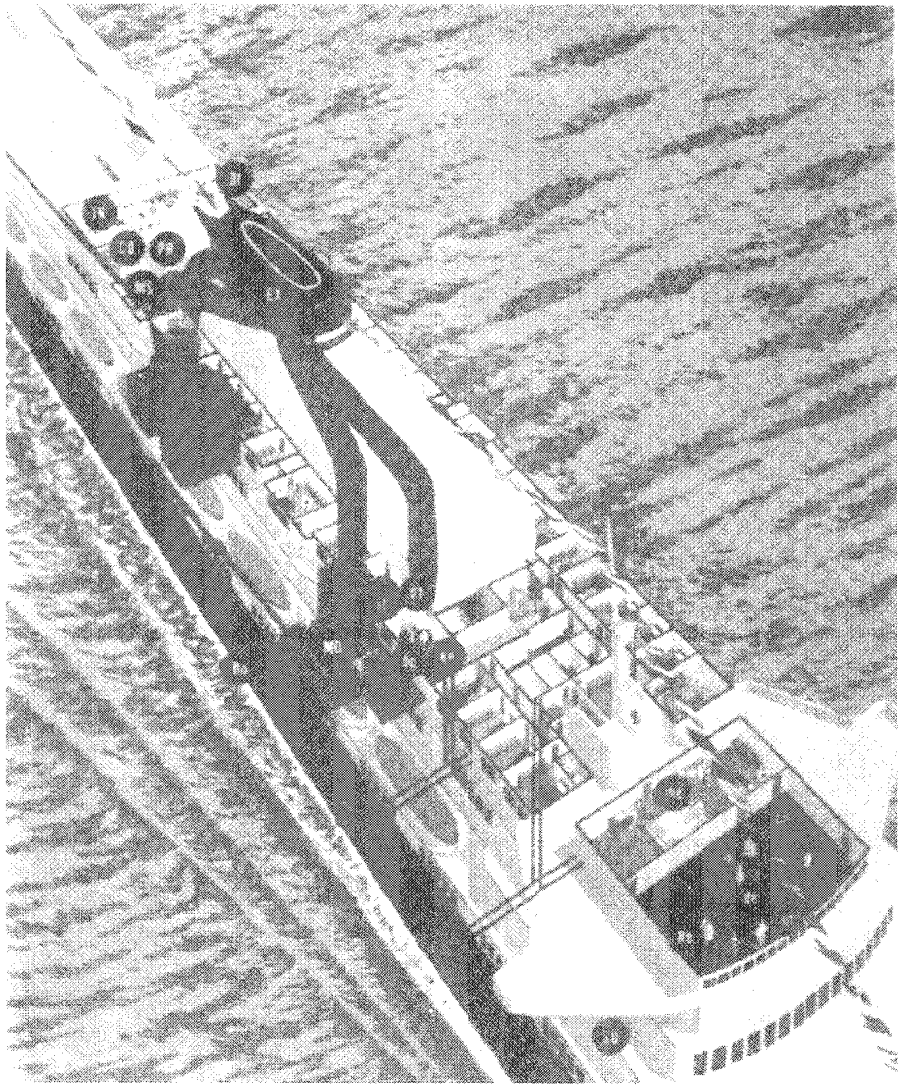


Figura 2.2(b): **Principales subsistemas de un crucero**

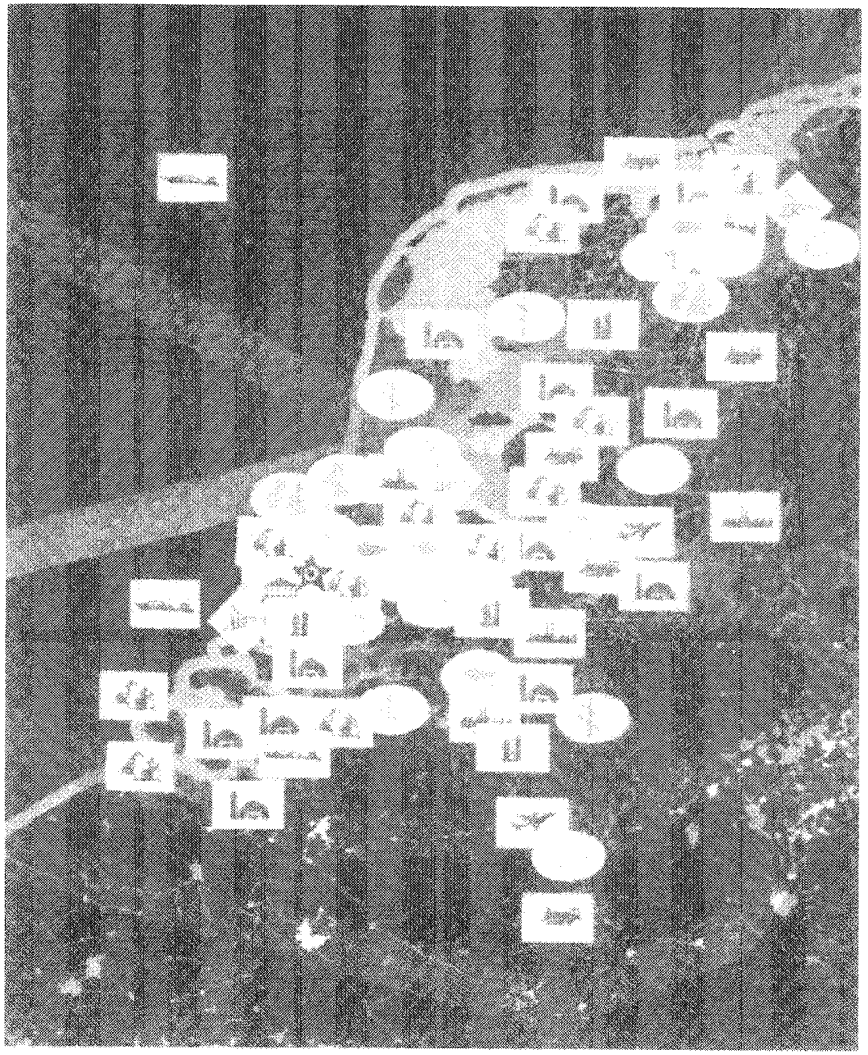


Figura 2.3(a): **Principales subsistemas de Holanda**

Los subsistemas que procesan tanto la materia-energía como la información: La frontera  , las organizaciones que defienden, cuidan o imponen la ley en la frontera nacional.

Los subsistemas que procesan la materia-energía: El ingestor  organizaciones como las aerolíneas, el ferrocarril, las compañías de transporte terrestre de carga y las de transporte marítimo, que importan diversas formas de materia-energía al interior del país; el distribuidor  , las organizaciones nacionales que transportan diversas formas de materia-energía por mar, tierra, ferrocarril o por vía aérea; el convertidor  , las organizaciones que convierten las formas primas de materia-energía a formas que puede utilizar la sociedad; el productor  las organizaciones que fabrican productos para el consumo de la sociedad o para exportar; El almacenador de materia-energía  , organizaciones tales como bodegas, presas y plantas eléctricas que almacenan diversos tipos de materia- energía; el expulsor  , las organizaciones que exportan los productos holandeses a otros países, o que descargan desechos en el mar, y las agencias que deportan a las personas indeseables; el motor  , las unidades de transporte, la industria de la construcción, las fuerzas armadas o la agencia espacial; el soporte  , los edificios públicos y la tierra.

Los subsistemas que procesan la información; el transductor de entrada  , las organizaciones que reciben señales de telégrafo, cable, teléfono o radar, o bien noticias extranjeras que vienen de más allá de las fronteras holandesas; el transductor interno  , la legislatura, los dirigentes de partidos políticos y las organizaciones de encuesta pública que reciben informes y comunicación de todo el país; el canal y las redes  , los sistemas de comunicaciones nacionales; el decodificador,  la oficina de relaciones exteriores que decodifica los mensajes secretos recibidos por las embajadas de Holanda en todo el mundo; el asociador  , las instituciones educativas holandesas; la memoria  , las bibliotecas; el que decide  , la reina y su Gobierno en La Haya; el codificador  , el Secretario de Prensa gubernamental holandés o los autores de discursos; el transductor de salida  las personas que dan los comunicados oficiales de Holanda.

Figura 2.3(b): **Principales subsistemas de Holanda**

Una división en categorías más útil de los sistemas automatizados es la siguiente:

- Sistemas en línea
- Sistemas de tiempo real
- Sistemas de apoyo a decisiones
- Sistemas basados en el conocimiento

Examinaremos con cuidado cada uno de éstos.

2.4.1 Sistemas en línea

En un libro anterior [Yourdon, 1972], definí los sistemas en línea de la siguiente forma:

> Un sistema en línea es aquel que acepta material de entrada directamente del área donde se creó. También es el sistema en el que el material de salida, o el resultado de la computación, se devuelve directamente a donde es requerido.

Esto usualmente significa que el sistema computacional tiene una arquitectura de hardware parecida a la de la figura 2.4.

Una característica común de los sistemas en línea es que entran datos a la computadora o se les recibe de ella en forma *remota*. Es decir, los usuarios del sistema computacional normalmente interactúan con la computadora desde terminales[6] que pueden estar localizadas a cientos de kilómetros de la computadora misma.

Otra característica de un sistema en línea es que los datos almacenados, es decir, sus archivos o su base de datos, usualmente se organizan de tal manera que los componentes individuales de información (tales como un registro individual de reservación aérea o un expediente individual de personal) puedan ser recuperadas modificados o ambas cosas 1) rápidamente y 2) sin tener necesariamente que efectuar accesos a otros componentes de información del sistema. Esto contrasta enormemente con los sistemas fuera de línea o en *lotes* (*batch*), que eran más comunes en las décadas de los años 60 y 70. En un sistema computacional por lotes, la información suele recuperarse de una manera secuencial, lo cual significa que el sistema computacional lee todos los registros de la base de datos, procesando y actualizando aquellos para los cuales haya actividad. La diferencia entre un sistema computacio-

6 Actualmente, la palabra "terminal" se usa de manera tan común en la sociedad que en realidad no requiere definirse. Sin embargo, entérese de que hay muchos sinónimos: "pantalla", "estación de trabajo", "teclado", etc. Además, hay abreviaturas comunes (en el inglés de uso habitual en informática) para describir la unidad de entrada/salida que se emplea para comunicarse con la computadora, como "CRT" para describir el "tubo de rayos catódicos", "VDU" para la "unidad de exhibición visual", etc. Estos términos se utilizarán indistintamente a lo largo del libro.

nal por lotes y uno en línea es análoga a la diferencia entre encontrar una selección musical específica en un cassette o en un disco; lo primero involucra el acceso secuencial a través de todas las pistas, mientras que lo segundo permite el acceso "aleatorio" a cualquiera de las pistas sin tener que escuchar las demás.

Dado que un sistema en línea interactúa directamente con personas (por ejemplo, usuarios humanos en sus terminales), es importante que el analista de sistemas planee cuidadosamente la interfaz humano-computadora.[7] Es decir, el analista debe tener alguna manera de *modelar*, esto, es, de crear modelos de todos los posibles mensajes que el usuario humano puede teclear en su terminal, y de todas las respuestas que el sistema pudiera dar, además de todas las respuestas que pudiera dar el humano ante las respuestas de la computadora, etc. Esto usualmente se lleva a cabo identificando todos los estados en los que la computadora y el usuario pudieran encontrarse, e identificando todos los cambios de estado. Un ejemplo de un estado en el que pudiera encontrarse una computadora de un sistema de cajero bancario automático es "el usuario ha insertado su tarjeta y se ha identificado, pero aún no

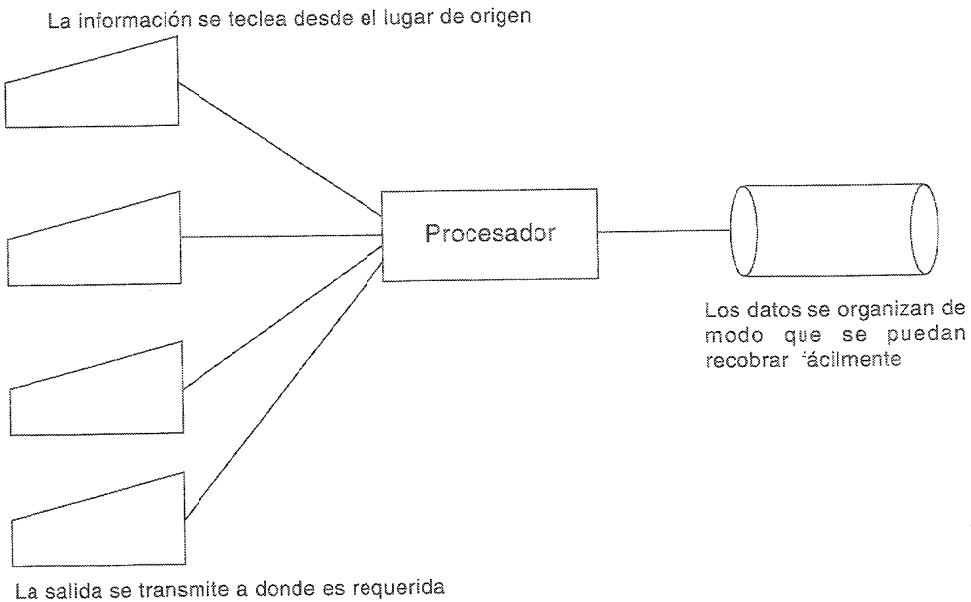


Figura 2.4: **Un sistema en línea**

7 A menudo se hace referencia a esto como "diálogo hombre-máquina", o "interfaz hombre-máquina". Cada vez son más las organizaciones de desarrollo de sistemas que utilizan la expresión "interfaz humano-computadora" o, simplemente, "interfaz humana" para evitar las inferencias sexistas.

me ha dado su clave secreta". Un ejemplo de cambio de estado es "me ha dado su clave secreta y ahora puedo proceder a determinar si desea retirar efectivo o desea que le informe acerca de su estado de cuenta". Otro cambio de estado pudiera ser "ha tratado sin éxito de ingresar su clave tres veces y ahora voy a hacer sonar la alarma". Estos estados y cambios de estado se modelan típicamente con *diagramas de transición de estado*, que discutiremos con detalle en el capítulo 13.

Dado que los sistemas en línea por lo común requieren recuperar datos con rapidez (para poder responder a preguntas y órdenes provenientes de terminales en línea), suele ser muy importante diseñar los archivos y las bases de datos de la manera más eficiente posible. De hecho, a menudo las *operaciones de computación* llevadas a cabo por un sistema en línea suelen ser relativamente triviales, mientras que los *datos* (sobre todo, la estructura y organización de los datos mantenida por el sistema en línea) suelen ser bastante complejos. De aquí que las herramientas de modelado de datos discutidas en el capítulo 12 sean de gran importancia para el analista y para el diseñador de sistemas.

La decisión de convertir o no un sistema ordinario en un sistema en línea es, en el contexto de este libro, una decisión de *puesta en práctica*, es decir, no es algo que debiera ser determinado por el analista de sistemas sino más bien por las personas que ponen en práctica el sistema. Sin embargo, dado que decidir tal cosa tiene un impacto tan obvio en el usuario (la presencia o ausencia de terminales de computadora, etc), es una decisión de puesta en práctica en la cual habitualmente los usuarios querrán participar. De hecho, es parte del *modelo de puesta en práctica del usuario* que discutiremos en el capítulo 21.

2.4.2 Sistemas de tiempo real

Un sistema de tiempo real es considerado por muchos como una variante de un sistema en línea; de hecho, muchos usan ambos términos indistintamente. Sin embargo, es importante distinguirlos: utilizaremos la siguiente definición de [Martín, 1967]:

> Un sistema computacional de tiempo real puede definirse como aquel que controla un ambiente recibiendo datos, procesándolos y devolviéndolos con la suficiente rapidez como para influir en dicho ambiente en ese momento.

La expresión "con la suficiente rapidez" está, desde luego, sujeta a muchas interpretaciones. Ciertamente, existen muchos sistemas en línea (sistemas bancarios, de reservaciones aéreas y sistemas de bolsa) que se espera reaccionen en uno o dos segundos a un mensaje tecleado en la terminal. Sin embargo, en la mayoría de los sistemas de tiempo real, la computadora debe de reaccionar en *milisegundos* y a veces en *microsegundos* a los estímulos que recibe. Esto es característico de los siguientes tipos de sistemas:

- *Sistemas de control de procesos*: los sistemas computacionales que se utilizan para verificar y controlar refinerías, procesos químicos, molinos y operaciones de maquinado.
- *Sistemas de cajeros automáticos*: las "máquinas de efectivo" que muchos de nosotros usamos para hacer depósitos y retiros sencillos en el banco.
- *Sistemas de alta velocidad para adquisición de datos*: los sistemas computacionales que obtienen datos de telemetría a alta velocidad de satélites en órbita o las computadoras que capturan cantidades enormes de datos de experimentos de laboratorio.
- *Sistemas de guía de proyectiles*: los sistemas computacionales que deben rastrear la trayectoria de un proyectil y hacer ajustes continuos a la orientación y empuje de los propulsores.
- *Sistemas de conmutación telefónica*: sistemas computacionales que controlan la transmisión de voz y datos en miles de llamadas telefónicas, detectando los números marcados, condiciones de ocupado y todas las demás condiciones de una red telefónica típica.
- *Sistemas de vigilancia de pacientes*: sistemas computacionales que detectan los "signos vitales" de diversos pacientes (por ejemplo, temperatura y pulso) y que son capaces ya sea de ajustar el medicamento administrado o de hacer sonar la alarma si los signos vitales se mantienen fuera de ciertos límites predeterminados.

Además de la velocidad, existe otra característica que diferencia a los sistemas de tiempo real de los sistemas en línea: estos últimos suelen interactuar con las personas, mientras que los sistemas de tiempo real usualmente interactúan tanto con personas como con un *ambiente* que en generalmente es autónomo y a menudo hostil. De hecho, la principal preocupación del analista de sistemas en tiempo real es que, si la computadora no responde con la suficiente rapidez, el ambiente pudiera quedar fuera de control, los datos de entrada pudieran perderse sin remedio o un proyectil pudiera salirse de su trayectoria tanto que ya no fuera posible recuperarlo, o bien que un proceso de manufactura pudiera explotar[8]. En cambio, un sistema en línea que no responda con la suficientemente rapidez en general no hará más que volver impacientes y gruñones a sus usuarios. Si tienen que esperar más de tres segundos la respuesta de un sistema en línea, las personas pueden "explotar" en sentido figurado, pero no en sentido literal. Esto se ilustra en la figura 2.5.

8 Uno de los ejemplos más interesantes de una situación de tiempo real es el de un equipo de trabajo cuya misión era unir una pequeña computadora a una bomba nuclear. Cuando se detonara la bomba (como parte de un programa de pruebas), la computadora dispondría tan sólo de unos cuantos microsegundos para capturar tantos datos como fuera posible y transmitirlos a un sistema de computadoras remoto, antes de que se desintegraran el hardware y software por la explosión. *Esa* sí que es una real exigencia del procesamiento en tiempo real.

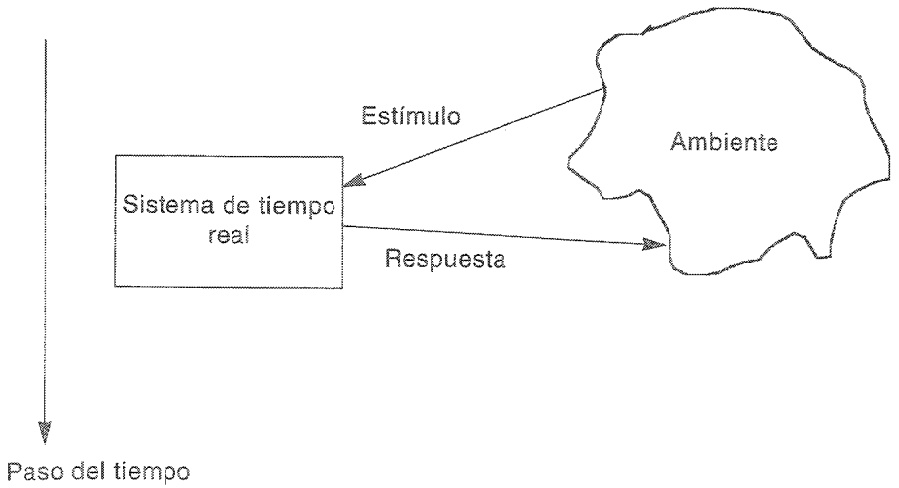


Figura 2.5: **Un sistema de tiempo real**

Dada la preocupación con la respuesta instantánea a las entradas del sistema, un analista que trabaja en sistemas de tiempo real generalmente estará muy preocupado por el *comportamiento dependiente del tiempo* que el sistema tenga. Discutiremos las herramientas para el modelado del comportamiento dependiente del tiempo en el capítulo 13.

Desde un punto de vista de su puesta en práctica, los sistemas de tiempo real (así como algunos sistemas en línea) se caracterizan por lo siguiente:

- Simultáneamente se lleva a cabo el proceso de muchas actividades.
- Se asignan prioridades diferentes a diferentes procesos: algunos requieren servicio inmediato mientras que otros pueden aplazarse por periodos razonables.
- Se interrumpe una tarea antes de concluirla, para comenzar otra de mayor prioridad.
- Existe gran comunicación entre tareas, especialmente dado que muchas tratan diferentes aspectos de un proceso general, como el control de un proceso de manufactura.
- Existe acceso simultáneo a datos comunes, tanto en memoria como en el almacenamiento secundario, por lo cual se requiere de un elaborado proceso de sincronización y "semáforos" para asegurar que los datos comunes no se corrompan.

- Existe un uso y asignación dinámicos de memoria RAM en el sistema computacional, dado que a menudo resulta poco económico (aun cuando la memoria sea barata) asignar suficiente memoria fija para manejar situaciones pico de alto volumen.

2.4.3 Sistemas de apoyo a decisiones y sistemas de planeación estratégica

La mayor parte de los sistemas automatizados de información que se crearon en los Estados Unidos durante los últimos 30 años son sistemas *operacionales* que ayudan a llevar a cabo los detalles del trabajo cotidiano de una organización. Estos sistemas, que también se conocen como sistemas de *procesamiento de transacciones*, incluyen ejemplos tan familiares como los sistemas de nómina, de contabilidad y de manufactura. En muchas organizaciones, en todo Estados Unidos, estos sistemas operacionales se han creado lenta, penosamente y a alto costo. Dado que muchos de ellos se iniciaron hace más de 20 años, están al borde del colapso. De aquí que continuamente se estén creando nuevos sistemas operacionales en las principales organizaciones del mundo entero.

Sin embargo, dado que los sistemas operacionales actuales continúan tambaleándose, muchas organizaciones se están enfocando su atención a un nuevo tipo de sistemas: los *de apoyo a la toma de decisiones*. Como lo implica el término, estos sistemas computacionales no toman decisiones por sí mismos, sino ayudan a los administradores y a otros profesionistas "trabajadores del conocimiento" de una organización a tomar decisiones inteligentes y documentadas acerca de los diversos aspectos de la operación. Típicamente, los sistemas de apoyo a decisiones son pasivos en el sentido de que no operan en forma regular: más bien, se utilizan de manera *ad hoc*, cuando se les necesita.

Existe un gran número de ejemplos sencillos de sistemas de apoyo a decisiones: programas de hoja de cálculo (por ejemplo, Lotus 1-2-3, Multiplan de Microsoft, Framework de Ashton Tate), sistemas de análisis estadístico, programas de pronóstico de mercados, etc. De hecho, una característica común de los sistemas de apoyo a decisiones es que no sólo recuperan y exhiben los datos, sino que también realizan varios tipos de análisis matemáticos y estadísticos de los mismos. Los sistemas de apoyo a decisiones también tienen la capacidad, en la mayoría de los casos, de presentar la información en una variedad de formas gráficas (tablas, gráficos, etc.) al igual que en forma de "reportes" (informes) convencionales. En la figura 2.6 se muestra una hoja de cálculo financiera característica que pudiera utilizar un gerente para evaluar las ganancias de alguna división dentro de su organización; la figura 2.7 es una gráfica típica que presenta las ganancias de dicha división comparadas con el promedio de la industria. Nótese que en ambos casos la información de salida producida por el sistema no "toma" una decisión, sino que provee información relevante para que el gerente pueda decidir.

Algunos sistemas de apoyo a decisiones son útiles para articular y mecanizar las reglas utilizadas para llegar a alguna decisión de negocios. Uno de estos sistemas es un programa llamado Lightyear (de Lightyear, Inc.), que se ejecuta en computadoras personales compatibles con IBM. Permite al usuario (o a múltiples usuarios) describir los detalles de un problema que requiera decisiones; un ejemplo podría ser el problema de decidir en dónde ubicar una nueva oficina. Primeramente, el usuario identifica los criterios que se utilizarán para tomar la decisión. Para el problema de ubicar una nueva oficina esto pudiera incluir, por ejemplo, que "debe ser accesible en transporte público" y que "no debe estar en una zona de alta probabilidad sísmica". Algunos de los criterios son binarios, en el sentido de que si no se satisface uno de ellos, se elimina una alternativa o se ocasiona la selección automática de otra. Algunos de los criterios pueden clasificarse en una escala numérica; por ejemplo, uno de los criterios pudiera ser la tasa de impuestos corporativos, los cuales tomarán diferentes valores numéricos dependiendo de la ciudad y estado donde se ubique la nueva oficina. Y los criterios mismos pueden clasificarse entre sí: pudiera ser que la importancia de los impuestos sea de 5 puntos en una escala de 10, mientras que la conveniencia de tener algún centro comercial cercano pudiera valer 3. Habiendo definido los criterios para llevar a cabo una decisión, las diversas alternativas pueden ser evaluadas y analizadas; la mejor alternativa automáticamente será seleccionada por el programa Lightyear. La figura 2.8 ilustra este proceso.

No hay nada mágico en esto: el programa meramente aplica en una forma mecánica las reglas de evaluación provistas por el usuario. Pero el poder del sistema va más allá del simple cálculo mecánico: fuerza al usuario a articular su propio criterio, lo cual a menudo no se hace. También ofrece una posibilidad neutral de obtener las opiniones de varios usuarios en situaciones en las que es de importancia lograr un consenso. En un asunto emocionalmente delicado, como reubicar una oficina (por ejemplo, reubicar a las familias de quienes llevan a cabo la decisión), puede ser útil no sólo articular los criterios de decisión, sino también la clasificación de criterios hecha por cada persona que participa en la decisión. Si dos miembros del comité de reubicación de oficinas están en desacuerdo, debiera quedarles claro por lo menos en qué se basa su desacuerdo.

Los *sistemas de planeación estratégica* son utilizados por los gerentes en jefe para evaluar y analizar la misión de la organización. En lugar de dar consejos acerca de alguna decisión de negocios aislada, estos sistemas ofrecen consejos más amplios y generales acerca de la naturaleza del mercado, las preferencias de los consumidores, el comportamiento de la competencia, etc. Esto usualmente cae dentro de los dominios del Departamento de Planeación Estratégica o del Departamento de Planeación a Largo Plazo, aunque pudiera tratarse de una actividad más informal, llevada a cabo por uno o dos gerentes.

La planeación estratégica es un concepto que se hizo popular durante la Segunda guerra mundial (aunque algunas organizaciones obviamente la practicaron desde mucho antes) y es tema de muchos libros; véase [Steiner, 1979], [Drucker, 1974] y [Ackoff, 1970]. Los sistemas de planeación estratégica no son programas de computadora en sí; son complejas combinaciones de actividades y procedimientos,

muchas de los cuales las llevan a cabo humanos utilizando información obtenida de fuentes externas (estudios de mercado, etc.) y datos internos de los sistemas operacionales de la organización y los sistemas de apoyo a decisiones. Steiner señala que hay muchos tipos de sistemas de planeación estratégica, según el tamaño y naturaleza de la organización.

Las figuras 2.9 y 2.10 muestran dos modelos típicos. El sistema de planeación estratégica basada en el análisis de brecha de posición trata de identificar la discrepancia entre la posición actual de la organización (en términos de ganancias, rentabilidad, etc.) y la posición deseada por la gerencia, los accionistas y otros.

Los sistemas de planeación estratégica conforman por sí solos un tema y no se tratarán con detalle en este libro. Haremos énfasis primordialmente en los sistemas de apoyo a decisiones y los sistemas operacionales.

Nótese la relación existente entre los tres distintos tipos de sistemas que se discuten en esta sección. Como lo muestra la figura 2.11, los sistemas operacionales representan la base sobre la cual se cimentan los sistemas de apoyo a decisiones y de planeación estratégica. Los sistemas operacionales *crean* los datos requeridos por los sistemas de nivel superior y continúan actualizando los datos de una manera continua.

Proyección de pérdidas y ganancias de la compañía

	C1	C2	C3	C4	TOTAL
Ventas nacionales	400	425	250	375	1450
Ventas internacionales	100	150	200	125	575
Cuotas por licencias	25	60	50	25	160
Ingresos diversos	10	10	15	10	45
TOTAL DE INGRESOS	535	645	515	535	2230
Costo de ventas	123	148	118	123	513
Salarios	100	120	120	125	465
Otros gastos de empleo	15	18	18	19	70
Renta	15	15	15	18	63
Teléfono	20	20	20	20	80
Correos	5	6	5	7	23
Viajes/diversiones	10	10	10	10	40
Contabilidad/asuntos legales	10	10	15	10	45
Depreciación	12	13	13	14	52
Gastos diversos	5	5	5	5	20
TOTAL DE GASTOS	315	365	339	351	1371
GANANCIAS/PERDIDAS	220	280	176	184	859

Figura 2.6: **Reporte tabulado de una hoja de cálculo típica**

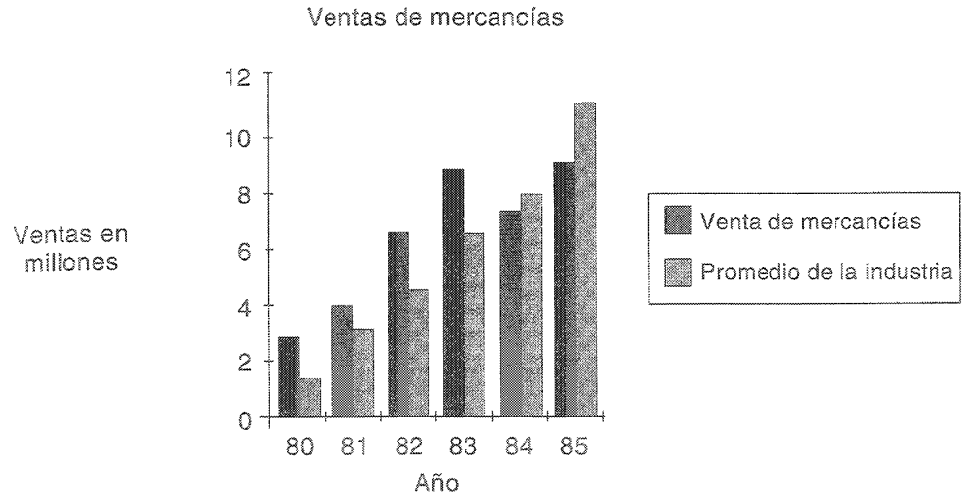


Figura 2.7: **Una gráfica típica hecha con un sistema de apoyo a la toma de decisiones**

La forma piramidal de la figura 2.11 representa otra característica típica de los sistemas de información que se pueden encontrar en la mayoría de las organizaciones hoy en día: el *tamaño* de los sistemas operacionales (medidos en años-persona, o en millones de instrucciones de COBOL, etc.) excede por mucho al de los sistemas de apoyo a la toma de decisiones y al de los sistemas de planeación estratégica. Pero podemos esperar que esto cambie gradualmente a lo largo de la siguiente década. Como se mencionó anteriormente, muchas organizaciones han pasado los últimos 30 años construyendo sus sistemas operacionales: *en gran medida, el trabajo ya está hecho* [9]. La mayor parte de la labor que se lleva a cabo actualmente en algunas de esas organizaciones importantes consiste en el desarrollo de sistemas de apoyo a la toma de decisiones y de sistemas de planeación estratégica.

9 Existen algunas excepciones: las organizaciones más pequeñas aún no han computarizado la mayor parte de sus operaciones diarias; los viejos sistemas desarrollados por las compañías Fortune 500 en la década de los años 60 están al borde del colapso; los nuevos sistemas que se necesitan para las fusiones de empresas, las adquisiciones y los estudios de mercado y nuevos productos; además la comunidad de la defensa aparentemente tiene una lista interminable de nuevos sistemas que se necesitan crear. Sin embargo, la tendencia general es la de olvidar los sistemas operacionales y dedicarse a los sistemas de apoyo a las decisiones.

2.4.4 Sistemas basados en el conocimiento

Un término relativamente novedoso en la industria de las computadoras es el de "sistemas expertos" o "sistemas basados en el conocimiento". Dichos sistemas se asocian con el campo de la inteligencia artificial, la cual Elaine Rich definió de la siguiente manera [Rich,1984]:

> La meta de los científicos de la computación que trabajan en el campo de la inteligencia artificial es producir programas capaces de imitar el desempeño humano en una gran variedad de tareas "inteligentes". Para algunos sistemas expertos la meta está próxima a ser alcanzada; para otros, aunque aún no sabemos construir programas que funcionen bien por sí solos, podemos comenzar a crear programas capaces de auxiliar a las personas en la ejecución de alguna tarea.

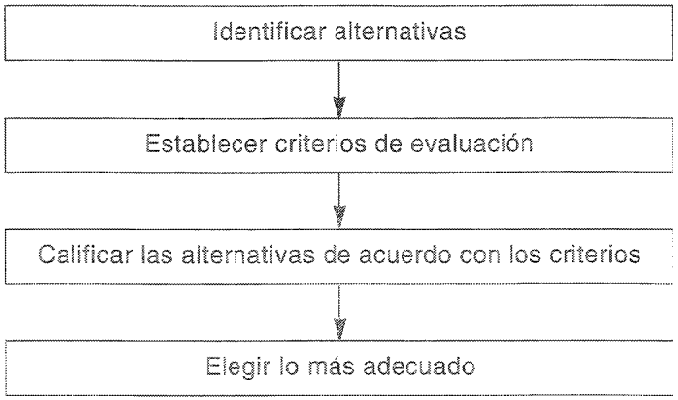


Figura 2.8: **El sistema Lightyear de apoyo a la toma de decisiones**

Dos eminentes autores en el campo de la inteligencia artificial, Feigenbaum y McCorduck, describen los sistemas basados en el conocimiento y los sistemas expertos [Feigenbaum y McCorduck, 1983] de la siguiente manera:

> Los sistemas basados en el conocimiento, por decir lo obvio, contienen grandes cantidades de diversos conocimientos que emplean en el desempeño de una tarea dada. Los sistemas expertos son una especie de sistema basado en el conocimiento, aunque ambos términos a menudo se utilizan indistintamente.
>
> ¿Qué es precisamente un sistema experto? Es un programa de computadora que contiene el conocimiento y la capacidad necesarios para desempeñarse en un nivel de experto. El desempeño experto significa, por ejemplo, el nivel de desempeño de médicos que llevan a cabo diagnósticos y procesos terapéuticos, o de físicos u otras personas de gran experiencia que llevan a cabo tareas de ingeniería, de administración o científicas. El sistema experto es un apoyo de alto nivel intelectual para el experto humano, lo cual explica su otro nombre, asistente inteligente.
>
> Los sistemas expertos por lo general se construyen de tal manera que sean capaces de explicar las líneas de razonamiento que llevaron a las decisiones que tomaron. Algunos de ellos pueden incluso explicar por qué descartaron ciertos caminos de razonamiento y por qué escogieron otros. Esta transparencia es una característica primordial de los sistemas expertos. Los diseñadores trabajan arduamente para lograrla, pues comprenden que el uso que se le dará al sistema experto dependerá de la credibilidad de que disfrute por parte de los usuarios, y la credibilidad surgirá debido a un comportamiento transparente y explicable.

Aún se piensa en los sistemas expertos como una especie de sistemas especializados, que utilizan hardware especial y lenguajes especiales, como LISP y PROLOG. Sin embargo, han comenzado a aparecer sistemas expertos sencillos, para computadoras personales estándar, y "cascarones" de sistemas expertos, que son estructuras de software para el desarrollo de aplicaciones específicas de sistemas expertos, también sencillos, en ambientes basados en COBOL estándar.

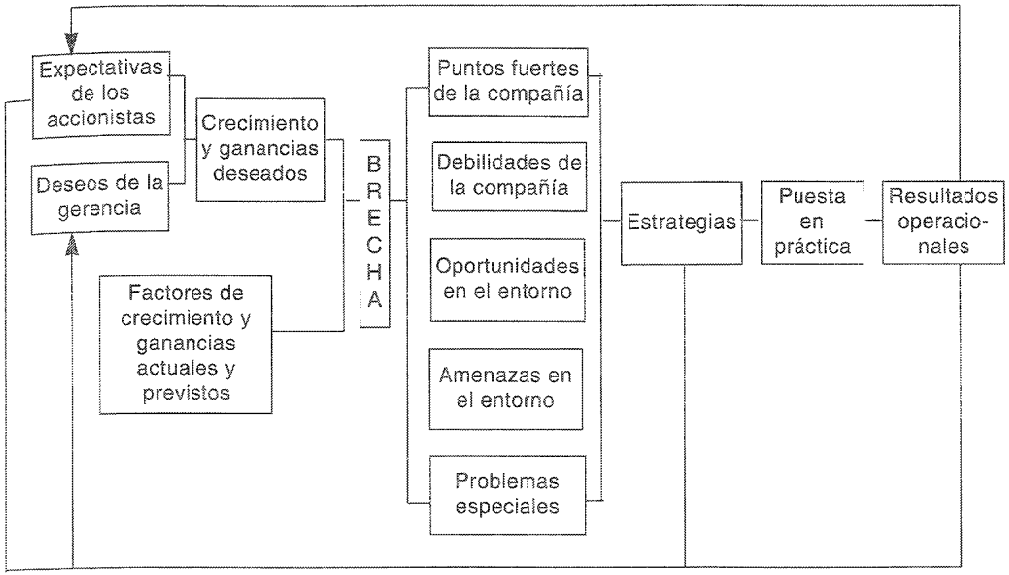


Figura 2.9: **Un modelo de planeación estratégica basado en el análisis de brecha de posición**

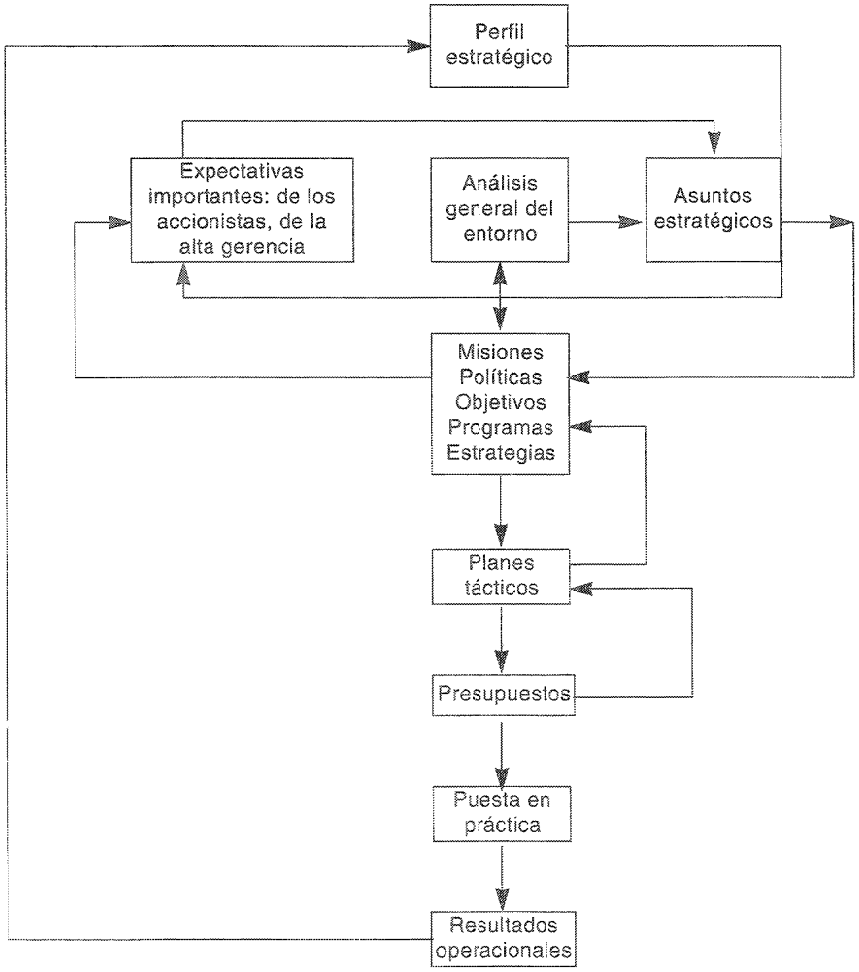


Figura 2.10: **Un modelo de planeación estratégica basado en la fuerza del mercado**

Aunque los sistemas expertos van más allá de los alcances de este libro, gradualmente se convertirán en un componente cada vez más importante de los sistemas "típicos" en los que trabaja un analista de sistemas. A fines de la década de los 80, los investigadores comenzaron a estudiar la relación entre las técnicas de desarrollo de software clásico y la inteligencia artificial; un estudio típico es [Jacob y Froscher, 1986]. Keller [Keller, 1987] prevé un momento en el futuro cercano en el

Figura 2.11: **La jerarquía de los sistemas de procesamiento de información**

cual los sistemas de IA y los sistemas expertos formarán parte de la actividad "normal" del análisis de sistemas; otros, como [Barstow, 1987] y [Lubars y Harandi, 1987], prevén que la inteligencia artificial auxiliará a los analistas de sistemas en la documentación de los requerimientos del usuario para mediados de la década de los 90. Posteriormente se tratará este punto.

2.5 PRINCIPIOS DE SISTEMAS GENERALES

Todos los ejemplos expuestos tienen una cosa en común: todos son *sistemas*. Mientras que pueden diferir en varias cosas, también poseen muchas características comunes. El estudio de dichas "características comunes" se conoce como *teoría general de sistemas* y es un tema fascinante de explorar. Para obtener un panorama inicial del tema, véase [Weinberg, 1976]; para un panorama más formal, consúltese [Bertalanffy, 1969], y para un panorama más humorístico de la frecuentemente perversa naturaleza de los sistemas, véase la encantadora obra de Gall, *Systemantics* [Gall, 1977].

Aunque el tema de la teoría general de los sistemas va más allá de lo que abarca este libro, existen algunos principios "generales" que son de interés particular para quienes crean sistemas automatizados de información, e incluyen los siguientes:

1. *Entre más especializado sea el sistema, menos capaz es de adaptarse a circunstancias diferentes.* Esto a menudo se utiliza para describir los sistemas biológicos (por ejemplo, los animales tienen dificultad para adap-

tarse a nuevos ambientes), pero se aplica también a los sistemas computacionales. Entre más general sea un sistema, menos óptimo será para una situación determinada; pero entre más óptimo sea, para tal situación menos adaptable será a nuevas circunstancias. Esto presenta un problema para muchos sistemas de tiempo real, que tienen que ser optimizados para poder proveer respuestas suficientemente rápidas a los estímulos externos. Pero el proceso de optimización suele aprovechar las idiosincrasias del hardware especial de la computadora y del software de sistemas utilizados en el proyecto, lo cual significa que pudiera ser muy difícil transportar el sistema a un hardware distinto. El principio también es de importancia para muchos sistemas de negocios, los cuales "reflejan" la política del usuario, que pudiera también ser altamente especializada. Entre más especializados sean los requerimientos para un sistema de nómina, por ejemplo, menos probable es que pueda utilizarse un paquete comercial.

2 *Cuanto mayor sea el sistema mayor es el número de sus recursos que deben dedicarse a su mantenimiento diario.* La biología es, una vez más, el ejemplo más familiar de este principio: los dinosaurios pasaban la mayor parte del su tiempo llenándose de alimento para poder mantener sus enormes cuerpos. Esto también se aplica a ejércitos, a compañías y a una gran variedad de otros sistemas, incluyendo los sistemas automatizados que estudiará en este libro. Un pequeño sistema de "juguete", del tipo que se puede crear en una sola tarde, por ejemplo, involucrará usualmente muy poca "burocracia", mientras que un sistema grande requerirá de un esfuerzo *enorme* en áreas tan "improductivas" como la revisión de errores, la edición, el respaldo, el mantenimiento, la seguridad, y la documentación.[10]

3. *Los sistemas siempre forman parte de sistemas mayores y siempre pueden dividirse en sistemas menores.* Este punto es importante por dos razones: primeramente, sugiere una forma obvia de organizar un sistema

10 A menudo, los usuarios no aprecian este fenómeno, y ésta pudiera ser una de las razones por las cuales actualmente están fascinados con los lenguajes de cuarta generación y las herramientas para hacer prototipos. Puede crearse con rapidez un sistema en un lenguaje de cuarta generación que haga las partes centrales del procesamiento (y que de esta manera recompense instantáneamente al usuario), pero costará mucho trabajo añadirle la inteligencia necesaria para revisar errores, respaldar, dar mantenimiento, asegurar, afinar, documentar, etc. Debe tomarse esto en cuenta, ya que de no ser así probablemente el usuario lo acosará para que construya un sistema "rápido y sucio" que a fin de cuentas fallará. Para dar una idea de los alcances de algo tan mundano como la documentación, considere la siguiente estadística, de la que rinde informe Capers Jones en *Programming Productivity* (Nueva York: McGraw Hill, 1986): Un sistema grande de telecomunicaciones tenía 120 palabras en inglés en cada renglón de código fuente, haciendo un total de 30 millones de palabras y de 60,000 páginas; un sistema gubernamental grande tenía 200 palabras en inglés por renglón de código fuente, haciendo un total de 125 millones de palabras y 250,000 páginas de documentación.

computacional que estemos tratando de desarrollar, por el procedimiento de dividirlo en sistemas menores (veremos mucho de esto en capítulos posteriores de este libro). Pero aún más importante es el hecho de que sugiere que la definición del sistema que estamos tratando de desarrollar es arbitraria; pudimos haber escogido un sistema ligeramente menor o mayor. Escoger lo que deberá *abarcar* un sistema y definirlo cuidadosamente para que todo mundo conozca su contenido es una actividad importante; se discutirá con mayor detalle en el capítulo 18. Esto es más difícil de lo que pudiera parecer: tanto los usuarios como los analistas a menudo piensan que la frontera del sistema es fija e inmutable y que todo lo que se encuentre fuera no merece la pena de ser estudiado. Estoy endeudado con Lavette Teague y Christopher Pidgeon [Teague y Pidgeon, 1985] por haber localizado el siguiente ejemplo de sistemas dentro de sistemas, tomado de [Eberhard, 1970]:

> Una ansiedad inherente en los métodos de diseño es la naturaleza jerárquica de la complejidad. Esta ansiedad se mueve en dos direcciones, el escalamiento y la regresión infinita. Utilizaré un cuento, *La advertencia de la manija*, para ilustrar el principio del escalamiento.
>
> Esta fue mi experiencia en Washington cuando tuve dinero hasta para regalar. Si contrato a un diseñador y le digo; "La manija de la puerta de mi oficina no es de buen diseño, carece de imaginación. ¿Me podría usted diseñar una nueva manija?" El me contestaría "Sí" y, tras acordar un precio, se marcha. Una semana más tarde regresa y me dice: "Sr. Eberhard, he estado pensando acerca de esa manija. Primero, debiéramos preguntarnos si abrir y cerrar una puerta por medio de una manija es la mejor forma de hacerlo". Yo le contesto, "Bien, creo en la imaginación. Hágase usted cargo". Más tarde, él regresa y me dice: "¿Sabe? He estado pensando en su problema y la única razón por la cual ocupa una manija es porque supone que necesita una puerta para su oficina. ¿Está seguro de que una puerta es la mejor manera de controlar el acceso y su privacidad?". "No, no estoy seguro de eso", replico. "Bueno, pues quiero dedicarme a ese problema". Regresa una semana más tarde y me dice: "la única razón por la cual debemos preocuparnos por el problema de la apertura es porque usted insiste en tener cuatro paredes en torno a su oficina. ¿Está seguro de que ésta sea la mejor manera de organizar este espacio para el tipo de trabajo que desempeña como burócrata?" Yo le respondo: "No, no estoy seguro en absoluto". Bueno, esto "escalaría" hasta que nuestro diseñador regresara con una cara muy seria (esto de hecho sucedió en dos contratos, aunque no exactamente por este mismo proceso) diciendo: "Sr. Eberhard, debemos decidir si la democracia capitalista es la mejor manera de organizar nuestro país, antes de que yo pueda atacar su problema".

> Por otro lado tenemos el problema de la regresión infinita. Si cuando esta persona se enfrentó al diseño de la manija hubiera dicho: "Espere, antes de preocuparme por la manija deseo estudiar la forma de una mano humana y lo que el ser humano es capaz de hacer con ella", yo le hubiese dicho: "Bueno". Luego hubiera regresado y me hubiera dicho: "Cuanto más pensé al respecto más me convencí de que se trata de un problema de ajuste. Lo que quiero estudiar primero es cómo se forma el metal y qué tecnología existe para fabricar objetos con metal, para así poder conocer los verdaderos parámetros para ajustarla a la mano". "Bueno", le hubiera contestado. Pero entonces me hubiera dicho: "¿Sabe?, he estado considerando la formación de metales y todo parece depender de las propiedades metalúrgicas. Quiero pasar tres o cuatro meses revisando el aspecto metalúrgico, para poder comprender mejor el problema". "Bueno", le hubiese contestado. Después de tres meses él hubiera vuelto diciendo: "Sr. Eberhard, cuanto más estudio el problema metalúrgico más me convenzo de que es la estructura atómica la que se encuentra en el fondo de todo esto". Y así, nuestro diseñador acabaría inmiscuido en la física atómica por la manija. Esta es una de nuestras ansiedades, la naturaleza jerárquica de la complejidad.

4. *Los sistemas crecen.* Desde luego, esto pudiera no ser verdad para *todos* los sistemas pues violaría un principio general muy familiar, la ley de la conservación de la energía. Pero muchos sistemas con los que estamos familiarizados sí crecen y resulta importante reconocerlo, pues a menudo omitimos (como analistas y como diseñadores de sistemas) tomar esto en cuenta al comenzar a crear el sistema. Un sistema de información típico, por ejemplo, crecerá hasta el punto de incluir más software, más datos, más funciones y más usuarios que los que inicialmente habíamos planeado. Por ejemplo, en una encuesta clásica de alrededor de 500 organizaciones de procesamiento de datos en los Estados Unidos, Lientz y Swanson [Lientz y Swanson, 1980] encontraron que la cantidad de código contenida en un sistema automatizado existente aumenta aproximadamente en un 10 por ciento al año y el tamaño de la base de datos se incrementa en alrededor de un 5 por ciento al año. No se puede suponer que un sistema ya hecho pueda permanecer estático; el costo de hacerlo crecer a medida que transcurre el tiempo debe incluirse en los cálculos de "costo- beneficio", que se discutirán en el capítulo 5 y en el apéndice C.

2.6 RESUMEN

Los analistas de sistemas en la profesión del procesamiento de datos a menudo son víctimas de la ley de la especialización anteriormente expuesta: se convierten en expertos en su propio campo, sin darse cuenta de la existencia de otros tipos de "constructores de sistemas" y de que se pudieran aplicar algunos principios generales. El propósito primordial de este capítulo ha sido ampliar su horizonte y ofrecer una mayor perspectiva antes de ahondar más en el estudio de los sistemas de información automatizados.

Obviamente, uno no puede convertirse en experto en sistemas vivientes, sistemas físicos y todo tipo de sistemas hechos por el hombre además de los sistemas de información automatizados. Pero dado que los sistemas que es probable que uno cree casi siempre interactúan con esos otros, es importante estar consciente de su existencia. Al comprender que otros sistemas obedecen a muchos de los mismos principios generales que observan los sistemas computacionales que esté haciendo, será más probable que tenga éxito al definir los límites entre su sistema y el mundo exterior.

REFERENCIAS

1. Edward Yourdon, *Design of On-Line Computer Systems*. Englewood Cliffs, N.J.: Prentice-Hall, 1972, Pág. 4
2. James Martin, *Design of Real-Time Computer Systems*. Englewood Cliffs, N.J.: Prentice-Hall, 1967.
3. James Grier Miller, *Living Systems*. New York: McGraw- Hill, 1978.
4. George Steiner, *Strategic Planning*. New York: Free Press, 1979.
5. Peter Drucker, *Management: Tasks, Responsibilities, Practices*. New York: Harper & Row, 1974.
6. Russell L. Ackoff, *A Concept of Corporate Planning*. New York: Wiley, 1970.
7. Stafford Beer, *Brain of the Firm*. New York: Wiley, 1972.
8. Stafford Beer, *The Heart of Enterprise*. New York: Wiley, 1978.
9. Stephen Hall, "Biochips", *High Technology*, diciembre, 1983.
10. H. Garrett DeYoung, "Biosensors", *High Technology*, noviembre, 1983.
11. Nicholas Shrady, "Molecular Computing", *Forbes*, julio 29, 1985.
12. David, Olmos, "DOD Finances Case Western Biochip Research Center", *Computerworld*, septiembre 3, 1984.
13. Elaine Rich, "The Gradual Expansion of Artificial Intelligence", IEEE *Computer*, mayo, 1984.
14. Edward Feigenbaum y Pamela McCorduck, *The Fifth Generation*. Reading, Mass.: Addison-Wesley, 1983.

15. R.J.K. Jacob y J.N. Froscher, "Software Engineering for Rule-Based Software Systems", *Proceedings of the 1986 Fall Joint Computer Conference*. Washington, D.C.: IEEE Computer Society Press, 1986.

16. Robert E. Keller, *Expert Systems Technology: Development and Application*. Englewood Cliffs, N.J.: Prentice-Hall, 1987.

17. Robert Alloway y Judith Quillard, "User Managers' Systems Needs", CISR Working Paper 86. Cambridge, Mass.: MIT Sloan School Center for Information Systems Research, abril, 1982.

18. Ludwig von Bertalanffy, *Teoría General de los Sistemas*. México: Fondo de Cultura Económica, 1976.

19. Gerald Weinberg, *An Introduction to General Systems Thinking*. New York: Wiley, 1976.

20. John Gall, *Systemantics*. New York: Quadrangle/The New York Times Book Company, 1977.

21. D. Barstow, "Artificial Intelligence and Software Engineering", *Proceedings of the 9th International Software Engineering Conference*, abril, 1987.

22. M.D. Lubars y M.T. Harandi, "Knowledge-Based Software Design Using Design Schemas", *Proceedings of the 9th International Software Engineering Conference*, abril, 1987.

23. Bennet P. Lientz y E. Burton Swanson, *Software Maintenance Management*, Reading, Mass.: Addison-Wesley, 1980.

24. Lavette Teague y Christopher Pidgeon, *Structured Analysis Methods for Computer Information Systems*. Chicago: Science Research Associates, 1985.

25. John P. Eberhard, "We Ought to Know the Difference", *Engineering Methods in Environmental Design and Planning*, Gary T. Moore, ed. Cambridge, Mass.: MIT Press, 1970, pp. 364-365.

PREGUNTAS Y EJERCICIOS

1. Dé dos ejemplos de cada una de las definiciones de sistema obtenidas del diccionario *Webster*, expuestas al comienzo del capítulo 2.

2. Dé cinco ejemplos de sistemas que hayan durado más de un millón de años y que aún existan hoy en día.

3. Dé cinco ejemplos de sistemas hechos por el hombre que hayan durado más de 1000 años. En cada caso, dé una breve explicación de por qué han durado y de si se pudiera esperar que continúen durante los siguientes 1000 años.

4. Dé cinco ejemplos de sistemas *no* hechos por el hombre que hayan fallado durante su vida. ¿Por qué fallaron?

5. Dé cinco ejemplos de sistemas hechos por el hombre que hayan fallado durante su vida. ¿Por qué fallaron?

6. Proyecto de investigación: lea la obra *Living Sistems*, de Miller, y haga una crítica.

7. Proyecto de investigación: lea la obra de Beer, *Brain of the Firm*, y haga una crítica.

8. Proyecto de investigación: lea la obra de Beer, *The Heart of Enterprise*, y haga una crítica.

9. De la sección 2.3, dé un ejemplo de un sistema hecho por el hombre que, en su opinión, *no* debiera automatizarse. ¿Por qué piensa que no debiera automatizarse? ¿Qué pudiera salir mal?

10. Dé un ejemplo de un sistema no automatizado que, en su opinión, *debiera* automatizarse. ¿Por qué piensa que debiera automatizarse? ¿Cuales serían los beneficios? ¿Cuál sería el costo? ¿Qué tanto puede confiar en los beneficios y en los costos?

11. Dé ejemplos de los 19 subsistemas de Miller para los siguientes tipos de sistemas automatizados: a) nómina, b) control de inventarios, c) el sistema telefónico.

12. Escoja una pequeña organización con la cual esté relativamente familiarizado, o bien un departamento o división de una organización grande. Para la organización escogida, lleve a cabo un inventario de los sistemas que utiliza. ¿Cuántos de éstos son sistemas operacionales? ¿Cuántos son sistemas de apoyo a decisiones? ¿Cuántos son sistemas de planeación estratégica? ¿Existen otras categorías útiles de sistemas? Para ayudarlo a enfocar su atención en esto, consulte [Alloway y Quillard,1982].

13. Dé cinco ejemplos de su propia experiencia de a) sistemas de tiempo real, b) sistemas en línea, c) sistemas de apoyo a la toma de decisiones, d) sistemas de planeación estratégica y e) sistemas expertos.

14. La figura 2.4 muestra una configuración típica de hardware para un sistema en línea. Dibuje el diagrama para una configuración de hardware *distinta* que sea razonable. ¿Tiene sentido tener parte de los datos de sistema localizados en las terminales? ¿En qué momento del desarrollo del sistema debiera discutirse esto con el usuario?

15. Dé un ejemplo de un sistema comercial que se describa como de "inteligencia artificial" o como un sistema "basado en el conocimiento" y que, en su opinión,

no está siendo descrito honesta o exactamente. ¿Por qué piensa que la descripción sea engañosa?

16. ¿Podría aplicarse el modelo estímulo-respuesta mostrado en la figura 2.5 a otros sistemas que no sean de los sistemas de tiempo real? ¿No responden acaso *todos* los sistemas a estímulos? ¿Qué tienen de especial los sistemas de tiempo real?

17. ¿Realmente puede tomar decisiones un sistema de apoyo a la toma de decisiones? Si no, ¿por qué no? ¿Qué pudiera hacerse para modificar un típico sistema de apoyo a la toma de decisiones para que *pudiera* tomarlas? ¿Sería deseable esto? ¿Cuales son los inconvenientes?

3 LOS PARTICIPANTES EN EL JUEGO DE LOS SISTEMAS

Todo el mundo es un escenario,
Y los hombres y mujeres son simples actores:
Tienen sus entradas y salidas;
Y un hombre, en el transcurso de su vida,
Realiza muchos papeles.

Shakespeare.
As You Like It, II, vii

En este capítulo se aprenderá:

1. Cuáles son las categorías de personas con las que interactuará a lo largo de un proyecto.
2. Cuáles son las tres principales categorías de usuarios, clasificados según su trabajo.
3. Cuáles son las reacciones de los usuarios durante un proyecto de desarrollo de sistemas.
4. Cuál es la diferencia entre los usuarios novatos y los expertos.
5. Cuál es el papel de la administración en un proyecto de desarrollo de sistemas.
6. Cuál es el papel de un analista en un proyecto de desarrollo de sistemas.
7. Qué otros papeles se pueden dar en un proyecto de desarrollo de sistemas.

Como analista de sistemas, usted trabajará en proyectos de desarrollo con una variedad de personas. Los personajes cambiarán de un proyecto a otro; las personalidades variarán dramáticamente, y el *número* de personas con las que tendrá que interactuar puede ir de una sola hasta docenas. Sin embargo, los papeles son bastante constantes, y verá los mismos una y otra vez.

Ser un analista con éxito requiere algo más que una simple comprensión de la tecnología de las computadoras. Entre otras cosas, requiere de habilidades interpersonales: pasará buena parte de su tiempo trabajando con otras personas, muchas de las cuales hablan un "idioma" muy diferente al suyo y encontrarán extraño e intimidante su "idioma" técnico computacional. Por eso, es importante que conozca las expectativas que los demás tendrán de usted y las que usted deberá tener de ellos.

En este capítulo se enfoca la atención sobre las características de las siguientes categorías principales de "jugadores" que probablemente encontrará en un proyecto característico de desarrollo de sistemas:

- Usuarios
- Administración
- Auditores, personal de control de calidad, y verificadores de normas
- Analistas de sistemas
- Diseñadores de sistemas
- Programadores
- Personal de operaciones

Cada una de estas categorías se describe a continuación.

3.1 USUARIOS

El participante más importante en el juego de los sistemas es alguien que el analista conoce como *usuario*. El usuario es aquél (o aquéllos) para quien se construye el sistema. Es la persona a la que tendrá que entrevistar, a menudo con gran detalle, a fin de conocer las características que deberá tener el nuevo sistema para poder tener éxito.

Debe hacerse notar que la mayoría de los usuarios no se refieren *a sí mismos* como "usuarios" (a menudo se utiliza esta palabra en otros contextos para describir a un drogadicto). En algunas organizaciones se evita ese problema utilizando el término *cliente* o *dueño* para identificar al usuario. El usuario es el "dueño" en el sentido de que es él quien recibe el sistema cuando se termina de crearlo. Y el usuario

es el "cliente" por lo menos en dos sentidos importantes: 1) como en muchas otras profesiones, "el cliente siempre tiene la razón", sin importar lo exigente, desagradable o irracional que pueda parecer y 2) el cliente es el que a fin de cuentas paga el sistema y usualmente tiene el derecho de rehusarse a pagar si no está conforme con el producto.

En la mayoría de los casos, es fácil identificar al usuario (o usuarios): de manera característica, es aquel que formalmente solicita un sistema. En una organización pequeña, esto suele ser un procedimiento muy informal; pudiera consistir simplemente en que el usuario llame por teléfono al analista oficial de sistemas y le diga: "Oye, Adriana, necesito un nuevo sistema para dar seguimiento a nuestra nueva campaña de comercialización". En una organización grande, el inicio de un proyecto de desarrollo de sistemas suele ser mucho más formal. Por lo común, la "solicitud de consideración y estudio de sistemas", como se le suele conocer, pasa por diversos niveles de aprobación antes de que se involucre al analista de sistemas. El capítulo 5 trata esto más a fondo.

Sin embargo, hay un gran número de situaciones en las que no se conoce la identidad del verdadero usuario o bien en las que hay poca oportunidad de que éste interactúe con el analista. Un ejemplo muy común de esto es el de un sistema en proceso de ser desarrollado por un negocio de consultoría o por una compañía productora de software: la interacción que exista entre el cliente y la compañía pudiera llevarse a cabo a través de administradores de contratos u otras agencias administrativas, a veces con cláusulas explícitas de que el analista *no* puede tener comunicación directa con el usuario. Aun si el sistema se desarrolla por completo dentro de una sola organización, el "verdadero" usuario pudiera nombrar a un representante para trabajar con el analista, por estar demasiado ocupado personalmente con otros asuntos.[1]

Obviamente, en situaciones de este tipo, hay una gran posibilidad de malos entendidos: lo que el usuario quiere que el sistema haga pudiera no serle comunicado de manera correcta al analista, y lo que éste crea que está construyendo para el usuario pudiera no serle comunicado tampoco de manera correcta, hasta que ya estuviera todo terminado, cuando ya sería demasiado tarde. De esto podemos sacar dos conclusiones importantes:

- Siempre que sea posible, el analista debiera tratar de establecer contacto directo con el usuario. Aun si se encuentran involucradas otras personas como intermediarios (por ejemplo, para tratar detalles de los contratos o asuntos administrativos), es importante tener reuniones con la persona

1 Una situación común de esta naturaleza es la del encargado de contratar proyectos en una organización gubernamental. En la mayoría de los casos, esta persona no es el usuario y puede no conocer mucho acerca de las verdaderas necesidades de éste, pero resulta ser el nominado para mantener cualquier comunicación oficial con la persona (o compañía) que deberá desarrollar el sistema.

que en último término recibirá el sistema. De hecho, suele ser aún mejor si el usuario forma parte activa del equipo encargado del proyecto. En muchas organizaciones, el usuario suele ser el gerente de proyectos; incluso, algunos argumentan que el usuario mismo debiera *llevar a cabo* el proyecto.

- Si *no* es posible comunicarse directamente con el usuario, la documentación generada por el analista se vuelve aún más importante. La parte II de este libro se dedica a las herramientas de modelado que pueden utilizarse para describir el comportamiento de un sistema de manera *formal* y *rigurosa*. Es esencial usar este tipo de herramientas para evitar malos entendidos costosos.

3.1.1 La heterogeneidad de los usuarios.

Uno de los errores más frecuentes que cometen en el campo de las computadoras sobre todo los programadores y a veces también los analistas, es suponer que todos los usuarios son iguales. La palabra "usuario", como sustantivo singular, da a entender que el analista sólo tendrá que interactuar con una persona. Aun cuando sea obvio que deberá intervenir más de un usuario, se tiene la tendencia a pensar en ellos como un grupo de humanos amorfo y homogéneo.

Decir simplemente que un usuario difiere de otro es insuficiente: claro, tienen diferentes personalidades, diferente preparación, diferentes intereses, etc. Pero también hay diferencias *importantes* que usted debe tener en mente al trabajar como analista. He aquí dos maneras de clasificar a los usuarios:

- Por categoría de trabajo o nivel de supervisión
- Por nivel de experiencia en el procesamiento de datos

3.1.2 Clasificación de los usuarios por categoría de trabajo

En un proyecto típico de análisis de sistemas se pasará una considerable cantidad de tiempo entrevistando a los usuarios para determinar los requerimientos del sistema. Pero, ¿cuáles usuarios?, ¿a qué nivel? Desde luego, esto dependerá del proyecto y de las políticas de su organización. Sin embargo, habitualmente tendrá que interactuar con individuos de tres categorías de trabajo: usuarios *operacionales*, usuarios *supervisores* y usuarios *ejecutivos*.[2]

Los usuarios *operacionales* son oficinistas, administradores y operadores que son los que más probablemente tendrán contacto diario con el nuevo sistema (a menos que esté usted construyendo un sistema de apoyo a las decisiones, en cuyo caso tendrá poco contacto con este grupo). Por eso, en una organización grande, tendrá que entrevistar a secretarias, agentes de seguros, bibliotecarios, oficinistas encargados de fletes, personal encargado de solicitudes y "ayudantes" de todos los tamaños, formas y colores. En el caso de un sistema de tiempo real, pudiera tener que hablar con usuarios operacionales tales como ingenieros, físicos, obreros, pilotos, operadoras telefónicas, etc. Debe tener tres cosas en mente cuando se trabaja con usuarios de nivel operacional:

1. Los usuarios de este nivel se preocupan mucho por las funciones que tendrá el sistema, pero es más probable aún que se preocupen por los detalles de la *interfaz humana*. Por ejemplo: ¿Qué tipo de teclado estaré usando para comunicarme con el sistema?; ¿es como el teclado de la máquina de escribir que he estado usando durante años; ¿como aparecerán las cosas en la pantalla?; ¿deslumbrará mucho la pantalla?; ¿se podrán leer fácilmente los caracteres?;[3] ¿cómo me indicará el sistema si he cometido un error?; ¿tendré que volver a teclear todo?; ¿qué tal si quiero "borrar" algo que teclee hace un momento?; cuando el sistema me haga un informe, ¿en dónde estará localizada la información en la página?; ¿puedo hacer que se imprima la fecha y la hora en la parte superior de cada hoja?, etc. Estos son detalles que el supervisor del usuario de nivel operacional pudiera o no tomar en cuenta, pero que, como se podrá imaginar, son vitales para el éxito de un sistema y se tendrán que abordar.[4] Esto significa que, como analista, necesitará tener comunicación directa con el usuario operacional o, en el peor de los casos, estar *muy* seguro de que la persona que representa a éste tenga presentes tales detalles. Estos se discuten más a fondo en el modelo de puesta en práctica por el usuario, en el Capítulo 21.

2. Los usuarios operacionales tienden a poseer un panorama "local" del sistema; por lo general son conocedores del trabajo específico que hacen y de las personas con las que tienen comunicación inmediata (clientes, supervisores, colegas, etc.). Sin embargo a menudo no están familiarizados con el panorama general; es decir, puede ser que tengan dificultad para describir cómo es que su actividad propia encaja dentro de la organiza-

2 Hay variantes de esta terminología: [Teague y Pidgeon, 1985], por ejemplo, se refieren también al "usuario beneficiado", el que recibirá los beneficios del nuevo sistema. Esta persona pudiera no tener contacto directo con el sistema, pero se beneficiará de alguna manera con el servicio mejorado o la funcionalidad del nuevo sistema.

3 Hay argumentos en relacion con esto que hacen hincapié en el hecho de que un sistema nuevo es parte de un sistema aún mayor: el usuario puede preguntar: "¿Me lastimará la espalda o me dará tendinitis el estar sentado frente a una terminal todo el día?", "¿Necesito preocuparme por la radiación proveniente de una pantalla de video?", "¿Qué tal si no sé teclear?" y, lo más importante, "¿Qué tal si este nuevo sistema me reemplaza en el trabajo y me deja sin empleo?"

4 En casos extremos, esto también significa que es el usuario operacional quien puede hacer o deshacer un sistema nuevo. Los usuarios operacionales pueden parecer pasivos y puede ser que no tengan la autoridad o el poder para aprobar un proyecto de desarrollo de sistemas, pero si lo sabotean, o simplemente no lo usan, el sistema nuevo habrá fallado.

ción global o para describir el carácter global de su organización. Esto rara vez se debe a torpeza, sino a que no tienen interés en el asunto. O también pudiera reflejar que el usuario supervisor no les haya dado a conocer nada de eso porque así lo prefiere. Una consecuencia de esta situación es que el analista debe poder desarrollar modelos de sistemas que permitan *ambos* panoramas (es decir, descripciones de partes pequeñas y detalladas del sistema, independientemente de otras partes) *y* descripciones globales (es decir, panoramas de alto nivel del sistema entero que evitan caer en detalles).

3. Los usuarios operacionales suelen pensar en los sistemas en términos físicos, es decir, en términos de la tecnología de puesta en práctica que comúnmente se utiliza para "implantar" o hacer uso del sistema, o en términos de la tecnología que imaginan que *pudiera* utilizarse. Las discusiones abstractas acerca de "funciones" y "tipos de datos" pueden resultar difíciles; de aquí que el analista de sistemas pudiera requerir hablar con el usuario exclusivamente en términos familiares. Luego, como una actividad aparte, puede traducir esta descripción física a un "modelo esencial", es decir, a un modelo de lo que el sistema *debe* hacer, independientemente de la tecnología usada para realizarlo. Esto se discute más a fondo en el capítulo 17.

Los usuarios *supervisores* son, como el término lo da a entender, empleados como supervisores: usualmente administran a un grupo de usuarios operacionales y son responsables de sus logros (obviamente, se puede imaginar más de un nivel de usuarios supervisores en una organización grande). Pueden tener el título de supervisor, pero pueden ser también jefes de turno, gerentes, ejecutivos, jefes de ingeniería u otra multitud de cosas. Lo importante acerca de los usuarios supervisores es que:

- Muchos de ellos son usuarios operacionales que han sido promovidos. Por eso, usualmente están familiarizados con el trabajo de sus subordinados operacionales y se puede suponer que estarán de acuerdo con sus necesidades, preocupaciones y perspectivas. Sin embargo, esto no siempre es así. Dado que el mercado, la economía y la tecnología han cambiado tanto, el trabajo operacional de hoy en día puede diferir *mucho* de lo que era hace 20 años.

- Una de las razones por las cuales pudiera suponerse que no hay comunicación entre el usuario supervisor y el operacional es porque el primero a menudo debe regirse por un presupuesto. De aquí que a menudo se interesa en un nuevo sistema de información por la posibilidad de incrementar el volumen de trabajo realizado disminuyendo a la vez el costo de procesar las transacciones, y reduciendo también los errores en el trabajo. También pudiera ocurrírsele que un sistema nuevo le dará oportunidad de supervisar el trabajo (e incluso la actividad minuto a minuto) de cada usuario operacional. Dependiendo de cómo se realice esto, los usuarios operacionales pudieran tener o no la misma perspectiva que el usuario supervisor.

- Debido a este énfasis en la eficiencia operacional, por lo general el usuario supervisor es el que ve al nuevo sistema como una forma de reducir el número de usuarios operacionales (por despido o arrepentimiento) o de evitar que aumente su número al crecer el volumen de trabajo. Esto no es ni bueno ni malo, pero a menudo es el punto focal de batallas políticas, en las cuales el analista suele encontrarse en medio.[5]

- Por las mismas razones, el usuario supervisor a menudo actúa como intermediario entre el analista y los usuarios operacionales, arguyendo que estos últimos están demasiado ocupados como para perder su tiempo hablando con el analista. El supervisor replicará: "Después de todo, necesitamos el sistema computacional *precisamente* porque estamos tan ocupados". Como se podrá imaginar, ésta es una posición muy peligrosa para usted. Después de todo, el usuario operacional es el que se preocupará más por la interfaz humana del sistema y es poco probable que el supervisor pueda hacerse eco debidamente de estas necesidades.

- El usuario supervisor a menudo piensa en los mismos términos físicos que el operacional, y su perspectiva a menudo resulta tan local como la de este último. Desde luego, uno esperará que una persona de nivel administrativo tuviera un panorama más global; como corolario, pudiera resultar que el usuario supervisor ya no recuerde algunas de las detalladas políticas de negocios que los usuarios operacionales llevan a cabo.

- Finalmente, será el usuario supervisor con quien usted tendrá el contacto cotidiano primario. Es el que definirá los requerimientos y las políticas de la empresa que su sistema deberá realizar. Pudiera ser sólo un miembro pasivo del equipo (en el sentido de que participará sólo cuando se le entreviste), o bien un miembro de tiempo completo o incluso, como se mencionó anteriormente, el gerente del proyecto.

Los usuarios de *nivel ejecutivo* en general no se involucran directamente con el proyecto de desarrollo del sistema, a menos que el proyecto sea tan amplio y tan importante que tenga un impacto de primer orden en la empresa. Sin embargo, para

5 Adviértase que ésta es una característica de un sistema operacional (como se definió en el capítulo 2), pero generalmente no lo es de los sistemas de apoyo a decisiones. Nótese también que los gerentes o administradores de nivel superior por lo general se interesan más en los sistemas que les ofrecen una ventaja competitiva que en aquellos que reducen personal operacional en una o dos personas.

un proyecto normal, el usuario ejecutivo suele estar dos o tres niveles arriba de la acción asociada con el proyecto. En la medida que usted se involucre con ellos, probablemente descubrirá lo siguiente acerca de los usuarios ejecutivos:

- Pueden proporcionar la iniciativa para el proyecto, pero es más probable que sirvan sólo como autoridad para financiar las solicitudes que se originan en niveles más bajos de la organización.
- Por lo común, no fueron previamente usuarios operacionales o, si lo fueron, habrá sido hace tanto tiempo que cualquier experiencia que tengan al respecto será obsoleta. Por ello, no se encuentran en posición que les permita definir los requerimientos del sistema para aquellos que lo estarán manejando cotidianamente. Como excepción de esto tenemos el sistema de apoyo a decisiones que se discutió en el capítulo 2; tal sistema lo utilizaran primordialmente usuarios supervisores y ejecutivos.
- Los usuarios ejecutivos se preocupan más por los detalles estratégicos y las ganancias/pérdidas a largo plazo. De aquí que, por lo regular, estén menos interesados en asuntos operacionales tales como abatir los costos de transacción o ahorrarse tres oficinistas, que es lo que Paul Strassman llama "los beneficios de la informática" en su obra [Strassman, 1985]. Esto es, los nuevos mercados, los nuevos productos o la nueva ventaja competitiva que pudiera obtenerse con el sistema.
- Los usuarios de nivel ejecutivo generalmente se interesan más en el panorama global del sistema. En consecuencia, suelen no interesarse por los detalles. Como ya se mencionó, esto significa que debemos utilizar las herramientas de modelado que permiten dar un panorama global del sistema para los usuarios ejecutivos (y para cualquier otra persona que lo requiera), así como porciones detalladas del sistema para los usuarios operacionales que son los "expertos locales".
- Similarmente, los usuarios ejecutivos por lo general pueden trabajar con modelos abstractos de un sistema; de hecho, ya están acostumbrados a trabajar con modelos abstractos tales como modelos *financieros*, modelos de *mercado*, modelos *organizacionales* y modelos de *ingeniería* (de nuevos productos, oficinas, etc.). En realidad, no estarán interesados en los "modelos físicos" del sistema y se preguntarán por qué se toma usted la molestia de mostrárselos.

En resumen, usted interactuará con tres tipos o niveles diferentes de usuarios, como lo muestra la figura 3.1. Recuerde que tienen distintas perspectivas, intereses y prioridades y, a menudo distinta preparación. Estos tres tipos de usuarios se pueden caracterizar como lo muestra la tabla 3.1.

En la explicación anterior insinué que al usuario no siempre le agrada la perspectiva de un nuevo sistema; de hecho, a menudo se opondrán activamente a él. Este es casi siempre el caso con los usuarios operacionales (dado que son los que lo tendrán que usar), pero también se puede encontrar resistencia por parte del usuario supervisor (dado que éste pudiera sentir que el sistema tendrá un impacto negativo sobre la eficiencia o ganancias del área de la cual es responsable), o incluso por parte del usuario ejecutivo. Como lo señala Marjorie Leeson en su obra [Leeson, 1981],

> El analista que entiende de motivación básica, del por qué las personas se resisten al cambio y cómo se resisten a él, puede tal vez superar en parte la resistencia. La mayoría de los textos de administración hacen referencia a la *doraderarquía de las necesidades*, de A.H. Maslow. Las cinco categorías, desde la de más baja hasta de la más alta prioridad, son

Necesidad	Ejemplo
1. Fisiológica	Alimento, vestido y casa
2. Seguridad y estabilidad económica	Protección contra el peligro y la pérdida del empleo
3. Social	Poder identificarse con individuos y grupos
4. Egoísta	Reconocimiento, situación social e importancia
5. Realización	Realizarse al máximo en la creatividad y el desarrollo personal

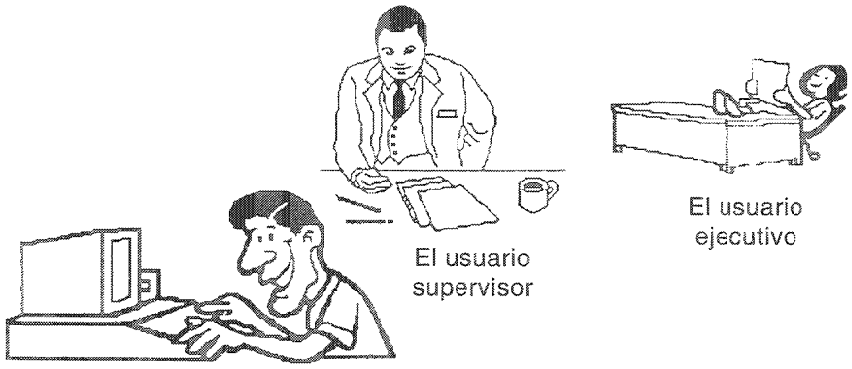


Figura 3.1: **Los tres tipos de usuarios**

Así, si encuentra que algunos usuarios se resisten a la idea de tener un nuevo sistema, deber pensar en la posibilidad de que una o más de estas necesidades no se esté satisfaciendo. Desde luego, es raro que el usuario se preocupe acerca del nivel fisiológico de la necesidad, pero no sorprende el hecho de que pueda preocuparse por la pérdida de su empleo. También es común que los usuarios (sobre todo los operacionales) se preocupen porque el sistema vaya tal vez a llevarlos a *no* poderse identificar con los grupos sociales que les son familiares; temen que estarán encadenados a una terminal todo el día y que pasarán todo su tiempo interactuando con una computadora en lugar de hacerlo con otros humanos. El usuario operacional que se haya vuelto experto en la realización de determinada labor de procesamiento manual de información pudiera temer que el nuevo sistema le perjudique en sus necesidades "egoístas"; y el usuario que sienta que el sistema le restará creatividad a su trabajo también se resistirá.

Tabla 3.1: Características de los diferentes usuarios

Usuario operacional	Usuario supervisor	Usuario ejecutivo
Usualmente tiene un panorama local	Puede o no tener un panorama local	Tiene un panorama global
Hace funcionar el sistema	Generalmente, está familiarizado con a operación	Provee la iniciativa para el proyecto
Tiene una visión física del sistema	Lo rigen consideraciones presupuestales	No tiene experiencia operacional directa
	Actúa a menudo como intermediario entre los usuarios y los niveles superiores de administración	Tiene preocupaciones estratégicas

3.1.3 Clasificación de los usuarios en categorías por nivel de experiencia

Debería ser obvio que los diferentes usuarios tendrán diferentes niveles de experiencia; desafortunadamente, es común que los analistas supongan que *todos* los usuarios son idiotas en lo que concierne al uso de computadoras. Tal vez esta suposición fuera admisible hace 10 años, pero es probable que le ocasione meterse en problemas en muchas organizaciones hoy en día[6]: actualmente se puede diferenciar

6 Aun cuando cada usuario con el que se encuentre no conozca y no tenga interés por la tecnología de las computadoras, debiera evitar el error común de considerarlos a todos como una forma de vida subhumana. Los analistas y programadores jóvenes, sobre todo los experimentadores que empezaron a utilizar las computadoras desde la escuela primaria, suponen que todos deben estar fascinados con las computadoras y tener habilidad para usarlas, y que aquellos que no cumplan con esto son 1) retrasados mentales, o bien 2) miembros de una generación antigua y, por tanto, indignos de consideración o respeto. Mientras tanto, el mundo está lleno de usuarios que no gustan de las computadoras por una variedad de razones legítimas, *y hay usuarios que están demasiado ocupados tratando de ser expertos en su propia profesión o negocio como para preocuparse por ser expertos en computadoras.* Tienen la misma opinión de los programadores de computadoras y de

entre amateurs, novatos presuntuosos y un pequeño (pero creciente) grupo de verdaderos expertos.

El amateur es aquél que jamás ha visto una computadora y que exclama a todo pulmón y con frecuencia que él "no entiende todo este asunto de las computadoras". A menudo, este tipo de usuario suele ser un empleado o negociante de mediana edad que ha sobrevivido felizmente a lo largo de 16 años de educación y de otros 10 o 20 años en un empleo *antes* de que se introdujeran las computadoras. Sin embargo, también es común encontrar usuarios jóvenes (de veintitantos años) que encuentran a las computadoras aburridas, intimidantes o inaplicables en sus vidas. Esto presenta un reto para el analista de sistemas al que le encanta hablar del "acceso en línea" y los "diálogos hombre-máquina dirigidos por menús" y terminología por el estilo. *Pero si el analista hace bien su trabajo, no hay razón por la cual el usuario deba interesarse por las computadoras o tener grandes conocimientos acerca de ellas.*

En realidad, el verdadero problema con el usuario amateur es un tanto más sutil: puede ser que encuentre difícil de entender el "lenguaje" que el analista usa para describir las características, funciones y opciones que ofrece el sistema que se va a implantar, aun cuando se evite la terminología obviamente relacionada con las computadoras. Como veremos en las partes II y III, el trabajo del analista de sistemas comprende la creación de varios *modelos* del sistema que se implantará. Estos modelos son representaciones formales y rigurosas de un sistema computacional y al mismo tiempo son representaciones *abstractas* del sistema. La mayoría de los modelos comprenden gráficas (imágenes) apoyadas con textos detallados y la representación global (que es necesaria para asegurar una descripción formal y rigurosa) da a algunos usuarios la impresión de ser abrumadoramente matemática y por lo tanto no legible. Pudiera tratarse de usuarios que recuerdan la dificultad de leer las notaciones gráficas complejas utilizadas en química orgánica o la notación igualmente compleja que se utiliza en el cálculo diferencial y en el álgebra. Cualquiera que sea la razón el resultado es el mismo: dejando de lado el entendimiento de la tecnología computacional, si el usuario ni siquiera puede entender el modelo de un sistema, hay poca probabilidad de que le satisfaga el sistema cuando por fin esté terminado.[7]

los analistas de sistemas que la que tienen de los electricistas, carpinteros, plomeros y mecánicos automotrices: aprecian las habilidades y destrezas requeridas para llevar a cabo el trabajo, pero exhiben una total falta de interés en los detalles. Comprender este punto en muchos casos determinará si usted tendrá éxito o no en sus primeros proyectos como analista.

7 Como analogía: si le fueran a construir su casa, empezaría por discutir las características deseadas con el arquitecto. Tras mucho discutir, éste se iría a su oficina y luego volvería cor varios bosquejos o maquetas a escala de la casa. Si usted se rehusara a mirar los bosquejos o alegara que son "demasiado matemáticos", el arquitecto tendría pocas probabilidades de éxito. Lo que con toda probabilidad haría usted es llevarlo a una casa existente y decirle "constrúyame una como esa". Desgraciadamente, a menudo no estamos en una posición adecuada para hacer eso en el campo de las computadoras, aunque, muchas veces, la elaboración de prototipos es una manera viable de lograr lo mismo.

Un segundo tipo de usuario es aquél que pudiéramos llamar "el novato presuntuoso"; es una persona que ha tenido que ver con uno o dos proyectos de desarrollo de sistemas o (peor aún) es un usuario que posee una computadora personal y que ha escrito uno o dos (¡uf!) programas en BASIC. Por lo común, alega saber *exactamente* lo que quiere que el sistema haga y suele señalar todos los errores que el analista cometió en el último proyecto. Esto está bien, excepto por una cosa: *a menudo se enzarza demasiado en discusiones sobre la tecnología específica que se usar para realizar el sistema*. Por eso, el usuario pudiera decirle al analista: "Necesito un nuevo sistema de procesamiento de pedidos y quiero que se construya con una red local que conecte a nuestras PCs IBM, y creo que debiéramos usar dBASE-III o PC-FOCUS". A la larga, éstas *pudieran* resultar ser las decisiones técnicas correctas, pero es prematuro considerar siquiera el hardware, el lenguaje de programación o los paquetes de base de datos antes de documentar los verdaderos requerimientos del sistema. De hecho, en un caso extremo, la "sugerencia" del usuario acerca del hardware y software apropiados pudiera convertirse en "una solución en busca de problema", es decir, el descubrimiento de que se tienen recursos de hardware y software poco utilizados a los que se les pudiera dar otro uso.

Desde luego, hay algunos usuarios que *realmente* entienden el análisis de sistemas, y también la tecnología de las computadoras (además de su propia profesión). Es un placer trabajar con tales personas; de hecho, el único problema pudiera ser que el usuario y el analista obtengan tal placer de la discusión sobre herramientas y técnicas del análisis de sistemas, que se olviden por completo de que su verdadero objetivo es implantar un sistema.[8]

3.2 ADMINISTRACION

El término "administración" es bastante amplio; de hecho, es probable que el analista de sistemas entre en contacto con diversos tipos de administradores:

- *Administradores usuarios.* Son administradores que están a cargo de varias personas en el área operacional donde se va a implantar el nuevo sistema. Esto se discutió anteriormente. Por lo general son administradores de nivel medio que desean sistemas que produzcan una variedad de informes internos y de análisis a corto plazo. Los informes internos suelen ser informes financieros, operacionales, de fallas, y otros por el estilo.

8 También levanta el ánimo ver que cada vez hay más de estos "expertos" que están llegando a ocupar puestos altos en la administración de organizaciones de negocios, y posiciones clave en otras partes de nuestra sociedad. Citibank y American Airlines, además de algunas otras compañías y organizaciones de alta tecnología, están dirigidas por personas que ascendieron a través de los rangos del procesamiento de datos. Hacia mediados de la década de los 80, había aproximadamente media docena de miembros del Congreso de los Estados Unidos con antecedentes de programación y análisis.

- *Administradores de informática.* Son las personas encargadas del proyecto en sí de sistemas, y los administradores de nivel superior encargados de la administración global y distribución de los recursos de todo el personal técnico de la organización de creación o desarrollo de sistemas.

- *Administración general.* Son los administradores de nivel superior que no están directamente involucrados con la organización de informática ni son de la organización usuaria. Pudiera ser el presidente de la organización o el jefe de administración financiera (el contralor, el director de finanzas, etc.). Generalmente se interesan más bien por los sistemas de planeación estratégica y de apoyo a decisiones que se discutieron en el capítulo 2. A pesar de que la administración superior sí requiere informes financieros internos, no suele necesitar la cantidad de detalles que ocupan los administradores usuarios (sobre todo en el área de informes de fallas). Además, se concentran más en la información externa: reglas gubernamentales, informes de la competencia por el mercado, informes sobre nuevos productos y mercados, etc.

La principal interacción entre el analista de sistemas y todos estos administradores tiene que ver con los *recursos* que se asignarán al proyecto. Es tarea del analista identificar y documentar los requerimientos del usuario y las *limitaciones dentro de las cuales se tendrá que implantar el sistema*. Por lo común, estas limitaciones son los recursos: personas, tiempo y dinero. De aquí que finalmente el analista hará un documento que diga: "El nuevo sistema deberá llevar a cabo las funciones X, Y y Z, y deberá desarrollarse en seis meses, con no más de tres programadores y con un costo máximo de 100,000 dólares."

Obviamente, la administración querrá que se le asegure que el proyecto de desarrollo del sistema se está manteniendo dentro de estos márgenes; es decir, que no se está atrasando ni está rebasando el presupuesto. Pero esto es un asunto de la administración de proyectos, no del análisis de sistemas.[9] Los administradores de las diferentes áreas funcionales suelen formar un comité directivo que ayuda a clasificar por prioridades los proyectos de desarrollo potencial, de manera que se lleven a cabo primero los más costeables.

Hay varios puntos que conviene tener en mente acerca de los administradores:

- Cuanto más alto nivel ocupen menos probable es que sepan (o que les importe saber) de la tecnología de las computacoras. Aunque esto sea una generalización, suele ser válida, dada la generación actual de administradores superiores. Esto no debiera afectarle a usted como analista (es más difícil la labor de los diseñadores de sistemas), pero debe recor-

9 Sin embargo, en ocasiones el analista puede estar muy involucrado con la administración. Discutiremos esto con más detalle en el capítulo 16, al igual que en el apéndice B.

dar que ha de concentrarse en tratar de las características *esenciales* del sistema cuando esté hablando con ellos.

- Las metas y prioridades de la administración pudieran entrar en conflicto con las de los usuarios, sobre todo las de los usuarios operacionales y los usuarios supervisores. La administración pudiera incluso querer imponerles un sistema y obligarlos a usarlo (por ejemplo, si la organización usuaria no ha podido responder a los cambios en el mercado o si no ha resultado lucrativa).

- Una variante de lo anterior es la siguiente: pudiera ser que la administración no esté dando los recursos, los fondos o el tiempo que los usuarios crean necesarios para implantar un sistema efectivo. Es cómodo para el analista y el usuario, en este caso, responder a esto que "la administración no entiende", pero a menudo se trata de una decisión consciente y calculada. En el Apéndice B se trata más a fondo el tema de la política de programación y financiamiento de recursos.

- El término "administración" da a entender un grupo homogéneo de personas que piensan todas de la misma manera; desde luego, la realidad suele ser muy diferente. Los administradores tienen diferentes puntos de vista y opiniones, y a menudo tienen diferentes metas y objetivos. Discuten y compiten unos con otros. Por esto, pudiera suceder que algunos miembros de la administración estén a favor del nuevo sistema y otros estén rotundamente en contra. Y la indiferencia que sufren algunos proyectos es aún peor; finalmente mueren tras años de rodeos y rodeos.

- También es cómodo suponer que una vez que la administración toma una decisión colectiva acerca de un determinado proyecto se atiene a dicha decisión. Sin embargo, no necesariamente sucede así: pudiera ser que fuerzas externas obliguen a la administración a acelerar determinado proyecto, a quitarle recursos o, de plano, abandonarlo. Esto suele causar una depresión emocional enorme a los que intervienen en el proyecto, incluyéndolo a usted como analista.

La relación entre su proyecto de desarrollo de sistemas y la administración pudiera depender en gran medida de la estructura administrativa global de su organización, sobre todo de la relación de las actividades del desarrollo de sistemas con el resto de la organización. La figura 3.2(a) muestra la estructura organizacional clásica. Nótese que toda la organización de procesamiento de datos rinde cuentas al jefe de finanzas y contabilidad. La razón de esto es que muchas organizaciones grandes originalmente introdujeron las computadoras para automatizar su contabilidad (nóminas, libro mayor y cuentas).

Desde la década de los 70, algunas organizaciones empezaron a darse cuenta de que esta estructura organizacional era bastante desproporcionada; garantizaba

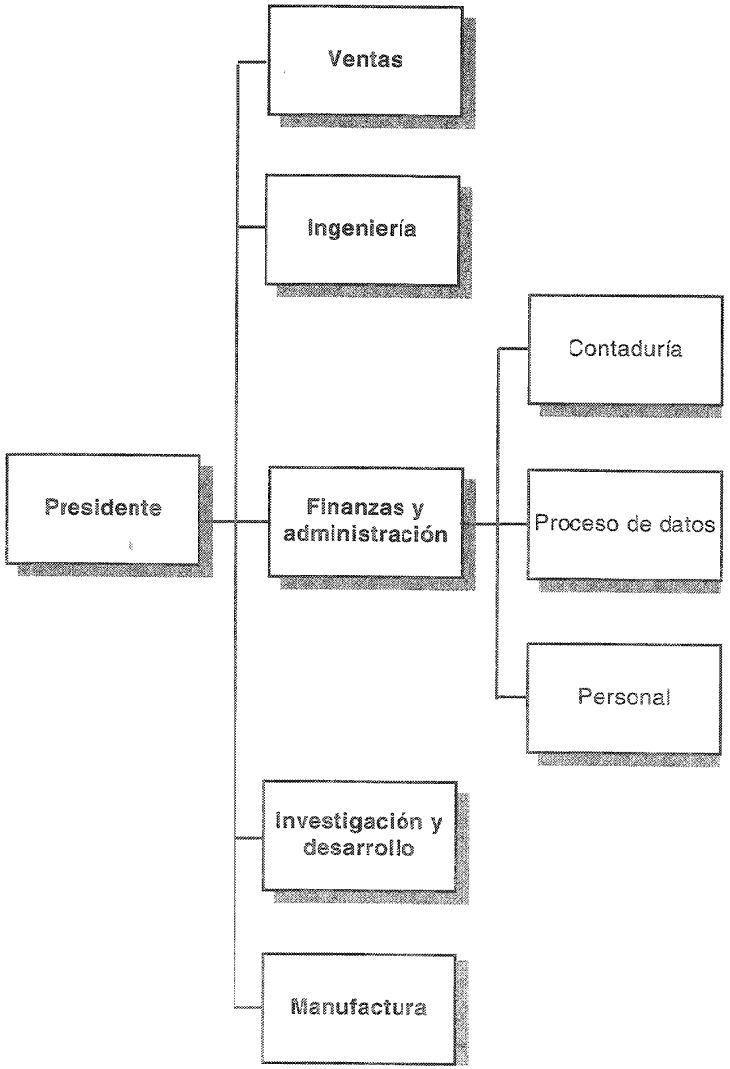


Figura 3.2(a): **Un esquema más común de organización**

que el proceso de datos estuviera enfocado más bien a aplicaciones contables y que tuviera entonces poco que ver con otras partes de la organización. Además, al empezar a implantar el proceso automatizado de datos (por ejemplo, en la manufactura, la comercialización y la ingeniería), algunas organizaciones cambiaron al esquema

mostrado en la figura 3.2(b). Al obligar al grupo de proceso de datos a informar directamente al presidente de la organización, es obvio para todos que el proceso de datos se vuelve tan crítico para la supervivencia de la organización como la manufactura, la ingeniería, las ventas, etc.

Sin embargo, para la década de los 80, en algunas organizaciones ya habían empezado a darse cuenta de que el departamento de proceso de datos se había convertido en un "imperio", con sus propias políticas y prioridades; mientras tanto, las organizaciones usuarias se encontraron con que tenían toda una lista de nuevos proyectos retrasados en espera de ser desarrollados por el departamento de informática.[10] Esto coincidió con la introducción y proliferación de computadoras personales potentes y baratas. Por ello, en algunos departamentos de usuarios empezaron a pensar que podían desarrollar sus propios sistemas, sin depender de una función centralizada. Como resultado de eso, algunas organizaciones tienen ahora una estructura como la que se muestra en la figura 3.2(c). Aunque aún existe un departamento central de proceso de datos o informática para aplicaciones "clásicas" tales como la nómina y el libro mayor, buena parte del proceso departamental lo llevan a cabo grupos de desarrollo de sistemas *dentro de los departamentos*.

Si trabaja para una organización por el estilo de la descrita por la figura 3.2(a), puede encontrarse con que el analista de sistemas y los usuarios de los otros departamentos no son tan buenos como deberían; de hecho, es probable que descubra que la mayoría de los proyectos de desarrollo de sistemas son de "proceso de transacciones", como los que pudiera encontrarse en un departamento de contabilidad. Si su organización se asemeja más a la de la Figura 3.2(b), hay una buena probabilidad de que su grupo de desarrollo de sistemas tenga una razonable "vistosidad" política en los altos rangos de la empresa; sin embargo, también pudiera detectar una creciente frustración por el rezago de los nuevos sistemas en espera de desarrollo. Y si trabaja en una empresa caracterizada por la figura 3.2(c), es probable que tenga mucho *más* contacto directo con los usuarios de su sistema; de hecho, pudiera ser que les rinda cuentas directamente a ellos. Y es más probable que llegue a trabajar con computadoras personales y en pequeñas redes de sistemas computacionales, comprados directamente por el departamento del usuario.

3.3 AUDITORES, CONTROL DE CALIDAD Y DEPARTAMENTO DE NORMAS O ESTANDARES

Según sea el tamaño de su proyecto y la naturaleza de la organización para la que trabaja, pudiera haber auditores, personal de control de calidad o miembros del departamento de normas o estándares participando en su proyecto. Se ha agrupado a estas personas en una sola categoría porque su objetivo y perspectiva se parecen en general, si no es que son iguales.

10 Discutiremos el retraso en la creación de las aplicaciones con más detalle en el capítulo 6.

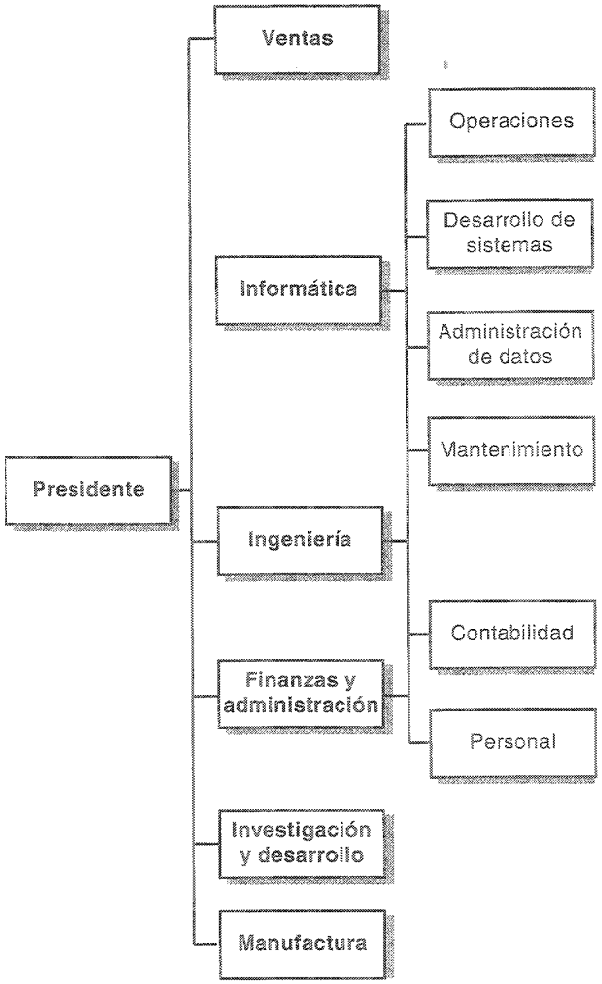


Figura 3.2(b): **Un esquema más común de organización**

El objetivo general de este equipo revuelto es asegurar que su sistema se desarrolle de acuerdo con diversos estándares o normas *externos* (es decir, externos a su proyecto): estándares de contabilidad desarrollados por la agencia contable de su organización, estándares desarrollados por otros departamentos de su organización o por el usuario que recibir su sistema; y posiblemente estándares impuestos por diversas dependencias gubernamentales reguladoras.

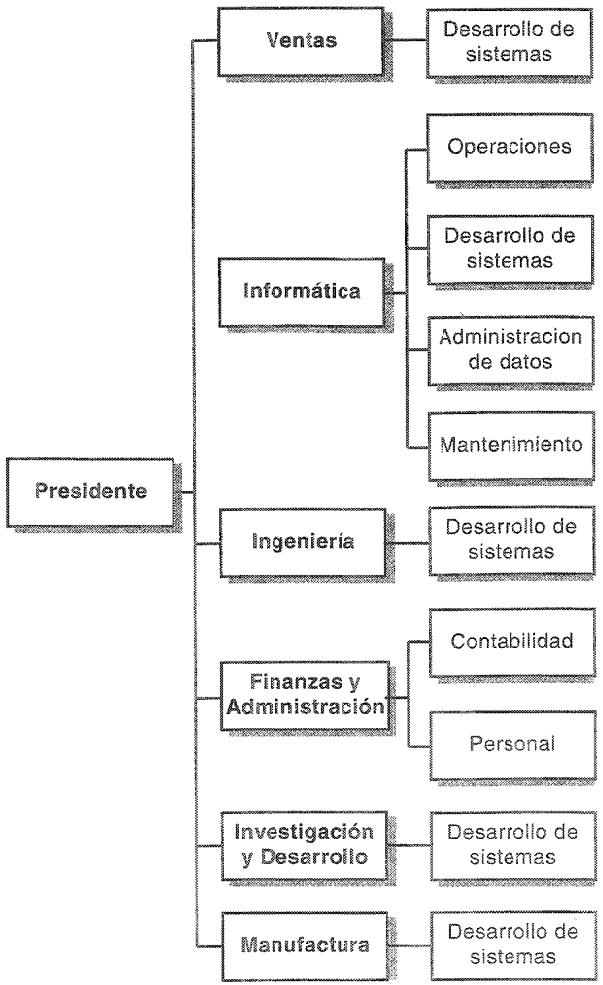


Figura 3.2(c): **El desarrollo de sistemas dentro de las organizaciones usuarias**

Hay tres problemas que debe prever, cuando esté trabajando con auditores, personal de control de calidad o miembros del departamento de normas o estándares:

1. A menudo no se involucran sino hasta el final en el proyecto. Después de que se ha terminado con el trabajo del análisis del sistema, el diseño y la programación, cuando se ha comenzado con la prueba formal. A estas alturas, por supuesto, es muy difícil hacer cambios importantes en el sistema.

2. A menudo están familiarizados con alguna notación o formato antiguos para documentación de requerimientos de sistemas (diagramas de flujo). Por eso, es importante asegurarse de que los modelos del sistema que usted desarrolle sean comprensibles (véase el capítulo 4).[11]

3. Por desgracia, los miembros de este grupo a menudo se interesan más por la forma que por el contenido: si sus documentos no tienen la presentación *exacta* que se exige pudieran ser rechazados.

3.4 EL ANALISTA DE SISTEMAS

Este es *usted*. El analista de sistemas es el personaje clave en cualquier proyecto de desarrollo de sistemas, y en otras partes de este Capítulo ya se ha mostrado la manera en la que el analista interactúa con otros participantes del juego.

En un sentido más amplio, el analista desempeña varios papeles:

- *Arqueólogo y escribano.* Como analista, una de sus principales labores es descubrir detalles y documentar la política de un negocio que pudieran existir sólo como "tradiciones tribales" transmitidas de generación a generación por los usuarios.

- *Innovador.* El analista debe distinguir entre síntomas, problemas del usuario y *causas*. Con sus conocimientos de la tecnología de las computadoras, el analista debe ayudar al usuario a explorar aplicaciones novedosas y más útiles de las computadoras así como formas nuevas de hacer negocios. Aunque muchos de los sistemas antiguos sólo se limitaban a perpetuar el negocio original del usuario, pero a velocidades electrónicas, hoy en día los analistas se enfrentan al desafío de ayudar al usuario a encontrar productos y mercados radicalmente innovadores, con la ayuda de la computadora. Pudiera ser aconsejable que lea la obra *Lateral Thinking*, de Edward De Bono [De Bono, 1970], para conocer formas nuevas e interesantes de considerar los problemas.

- *Mediador.* Como se mencionó anteriormente en este capítulo, el analista a menudo se encuentra en medio, entre usuarios, administradores, programadores, auditores y otros diversos participantes, los cuales frecuentemente están en desacuerdo entre sí. Es tentador para el analista

11 Sin embargo, en por lo menos algunos casos, esto está cambiando. Por ejemplo, muchas de las ocho grandes empresas contables de los Estados Unidos ya están bien familiarizadas con las herramientas de documentación del análisis estructurado descritas en este capítulo; por eso no debieran tener problema alguno en participar como auditores de alguno de sus proyectos.

imponer su propia opinión respecto a cómo debe ser el sistema o cuáles funciones debe cumplir. Pero su labor primordial es obtener un consenso y esto requiere del delicado arte de la diplomacia y la negociación.

- *Jefe de proyecto.* Este no es un papel universal, pero aparece con la suficiente frecuencia como para ser digno de mencionarse aquí. Dado que el analista suele tener más experiencia que los programadores que laboran en el proyecto y dado que se le asigna al mismo antes de que ellos empiecen a trabajar, hay una tendencia natural a asignar al analista las responsabilidades de la administración íntegra.

Esto significa que, como analista de sistemas, se necesita más que simple habilidad para dibujar diagramas de flujo y otros diagramas técnicos. Se requiere facilidad en el *manejo de personas* para poder entrevistar a los usuarios, mediar en desacuerdos y sobrevivir a las inevitables batallas políticas que se dan en todos los proyectos excepto los más triviales. Se necesita tener *conocimientos de aplicación* para entender y apreciar los asuntos del usuario. Se requiere *habilidad en computación* para entender los usos potenciales del hardware y el software en los asuntos del usuario. Y (obviamente) se necesita una mente lógica y organizada: debe ser capaz de ver un sistema desde diferentes perspectivas, debe poder dividirlo en niveles de subsistemas y debe ser capaz de pensar en el sistema en términos abstractos además de físicos.[12]

¡Nadie dijo que iba a ser un trabajo fácil!

3.5 DISEÑADORES DE SISTEMAS

Como hemos dado a entender en discusiones anteriores, el diseñador de sistemas es quien recibe los resultados de su trabajo de análisis: la labor de él es transformar la petición, libre de consideraciones de tecnología, emanada de los requerimientos del usuario, en un diseño arquitectónico de alto nivel que servirá de base para el trabajo de los programadores. En el capítulo 22 se discutirá la naturaleza de esta labor.

En muchos casos, el analista y el diseñador son la misma persona o el mismo grupo unificado de personas. Aun cuando sean personas distintas, es importante que se mantengan en contacto directo a lo largo de todo el proyecto. La razón por la cual se necesita esta *retroalimentación* continua entre diseñador y analista es la siguiente: el analista tiene que ofrecer información detallada suficiente como para que el diseñador pueda elaborar un diseño tecnológicamente superior y el diseñador debe proveer suficiente información para que el analista pueda darse cuenta de si los

12 De hecho, es debido a este requisito de ser experto en *muchas* áreas, que la mayoría de los que se dedican a la computación sienten que la inteligencia artificial y los sistemas expertos no se podrán aplicar al análisis de sistemas por muchos años más. Se discute esto más a fondo en el capítulo 25.

requerimientos que del usuario está documentando son tecnológicamente posibles. Basándose en la información recibida, el analista posiblemente tendrá que negociar con el usuario para modificar otros requerimientos.

3.6 LOS PROGRAMADORES

Se puede argumentar que en el mejor de los mundos no habría contacto entre un analista y un programador. Sobre todo en los proyectos grandes de desarrollo de sistemas, es probable que los diseñadores funcionen como un "amortiguador" entre los analistas y los programadores; es decir, los analistas entregan sus resultados (una descripción no técnica de los requerimientos del sistema) a los diseñadores, quienes a su vez entregan los suyos (una descripción arquitectónica del hardware y software que se usará para poner en práctica el sistema) a los programadores.

Existe otra razón por la cual el analista y el programador pudieran tener un contacto muy reducido, o nulo, entre sí: a menudo, se lleva a cabo el trabajo siguiendo una secuencia muy estricta en algunos proyectos de desarrollo de sistemas.[13] Por eso, la labor del analista se hace primero y *se termina por completo* antes de que comience la labor de programación. Esto significa que el analista muy probablemente estará incluso asignado ya a otro proyecto antes de que el programador intervenga en el actual.

Sin embargo, es probable que sí haya *algún* contacto entre programadores y analistas, por lo siguiente:

- En los proyectos pequeños, los papeles de analista, diseñador y programador se combinan, de tal manera que una sola persona hace tanto el papel de analista como el de diseñador y por tanto interactúa con el programador. O pudiera suceder que una sola persona realice la labor de diseñador y la de programador.
- El analista a veces sirve de administrador del proyecto, así que aunque haya concluido su labor de especificación de los requerimientos del sistema, aún estará involucrado en el proyecto y tendrá algún contacto con el programador.
- A menudo es el programador el que descubre errores y ambigüedades en la "propuesta de requerimientos" entregada por el analista, pues es durante la programación, como dice mi colega Scott Guthery, cuando "la llanta se adapta al asfalto", donde una reseña superficial de los requerimientos del sistema se traduce en un juego de instrucciones muy específicas y detalladas de COBOL. Si algo falta, o está mal o confuso, el

13 Discutiremos en el capítulo 5 algunas alternativas a este enfoque secuencial, sobre todo las conocidas como desarrollo evolutivo o rastreo rápido. De hecho, en algunos proyectos el análisis continúa a la vez que se está llevando a cabo la programación.

programador tiene dos opciones: pedirle una aclaración al analista o bien preguntarle al usuario.[14]

- Como se mencionó en el capítulo 2, muchas organizaciones se están viendo en la necesidad de reemplazar los sistemas operacionales originales que se crearon hace 20 años. En la gran mayoría de estos proyectos de reemplazo, casi no hay documentación que describa 1) cómo funciona el sistema o, más importante aún, 2) qué es lo que se supone que el sistema debe hacer. Y dado que los sistemas tienen 20 años de antigüedad, hay toda una nueva generación de usuarios involucrados. Los usuarios que inicialmente especificaron los requerimientos del sistema ya se jubilaron o renunciaron, y la nueva generación tiene pocas nociones sobre esos requerimientos. A estas alturas, todas las miradas se vuelven hacia el *programador de mantenimiento*, que ha estado manteniendo el sistema durante los últimos años; es probable que éste también sea un trabajador de segunda o tercera generación, que nunca haya tenido contacto con los diseñadores y programadores que construyeron originalmente el sistema. Como lo señala Nicholas Zvegintzov, (autor del boletín *Software Maintenance News*),

> Hasta ahora, el profesional clave de la computación era alguien que pudiera conocer lo suficiente acerca de las necesidades de las organizaciones como para expresarlas en términos computacionales. En el futuro, al computarizarse irrevocablemente nuestra sociedad, el profesional clave será alguien que pueda aprender lo suficiente acerca de los sistemas computacionales como para expresarlos en términos humanos. Sin ese alguien, habremos perdido el control de nuestra sociedad. Ese alguien es el ingeniero a la inversa. Los encargados del mantenimiento de software son los ingenieros a la inversa de la sociedad.

- Algunas organizaciones están empezando a cambiar sus equipos de desarrollo de proyectos de una estructura vertical a una horizontal. La distribución típica de responsabilidades (que se supone a lo largo de todo este libro) implica que todas las tareas del analista se le asignen a una sola persona (o a un solo grupo de personas); de manera similar, todas las actividades de diseño se le asignan al diseñador y todas las de programación al programador. En la medida en que se cumpla esto, ciertamente parecería que los analistas y los programadores tienen poco contacto entre sí. No obstante algunas organizaciones están empezando a percatarse de que en esto existe un conflicto inherente: los analistas suelen ser relativamente de alto nivel y de gran experiencia dentro de la empresa; sin embargo se espera que ellos lleven a cabo no sólo las labores de alto nivel, tales como el establecimiento conceptual de los requerimientos del sistema, sino también labores de bajo nivel, como los engorrosos detalles de los requerimientos del usuario. Existe un conflicto similar con los programadores, quienes típicamente suelen ser empleados menores y de menos experiencia. Una solución sería darle al personal técnico superior (cuyo título resulta ser el de analista) *todas* las tareas de alto nivel: el análisis de alto nivel de sistemas, el diseño de alto nivel, y *la codificación de los módulos de alto nivel del sistema*. Similarmente, a las personas de nivel técnico inferior se les dará tareas detalladas de bajo nivel en las áreas de análisis, de diseño y de programación. En la medida en que esto se siga, los analistas y los programadores mantendrán un contacto cercano durante todo el proyecto; de hecho, cada uno hará parte del trabajo que anteriormente le correspondía al otro. Esto se volverá a discutir en el capítulo 23.

14 De hecho, el contacto directo entre el programador y el usuario es más común de lo que pudiera pensarse. En muchos casos, el analista nunca llega a describir los detalles de bajo nivel del sistema, y los usuarios de alto nivel con los que se comunica el sistema pudieran no estar al tanto ni estar interesados en dichos detalles. Por eso, a menudo el programador *debe* hablar directamente con el usuario de bajo nivel para descubrir exactamente qué es lo que se supone que debe hacer el sistema. Esto es importante, pues muchas organizaciones se quejan del hecho de que el 50% de sus proyectos de desarrollo de sistemas se dedican a las pruebas; pudiera suceder que el trabajo que se hace con el pretexto de probar sea de hecho la labor de análisis que se pudiera (y probablemente se debiera) haber hecho anteriormente.

3.7 EL PERSONAL DE OPERACIONES

Así como se pudiera argumentar que el analista nunca se encontraría con un programador, pudiera argumentarse también que no necesitara tener contacto con el personal de *operaciones* responsable del centro de cómputo, la red de telecomunicaciones, la seguridad del hardware y del software, además de la ejecución de los programas, el montaje de los discos y el manejo de la salida de las impresoras. Todo esto sucede después de haber sido tanto analizado y diseñado como programado y probado el sistema.

Sin embargo, hay más de lo que parece a simple vista: el analista debe entender las *restricciones* impuestas al nuevo sistema por el personal de operaciones, pues esto influye en la especificación detallada que produzca. Es decir, el analista pudiera elaborar un documento que diga: "el nuevo sistema de pedidos deberá ser capaz de llevar a cabo las funciones X, Y y *Z y, para poder satisfacer los requisitos impuestos por el departamento de operaciones, no debe de ocupar más de 16 megabytes de memoria de la computadora principal. El sistema debe implantarse utilizando terminales IBM 3270 estándar comunicadas con la red XYZ de telecomunicaciones de la compañía".*

En algunos casos, los detalles operacionales del sistema pudieran reducirse a una cuestión de negociación entre el usuario y el grupo central de operaciones de la computadora. Esto es muy común hoy en día, dado que a menudo los usuarios están en posibilidades de adquirir sus propias computadoras personales o minicompu-

tadoras de tamaño apropiado para sus departamentos. A pesar de que la mayoría de estas computadoras pueden ser utilizadas por oficinistas o personal administrativo de la organización usuaria (es decir, no se requiere personal que tenga la especialización del de operaciones), y a pesar de que muchas de ellas pueden trabajar en un ambiente normal de oficina (es decir, que no requieren del sistema especial de conexiones y del aire acondicionado que necesitan las grandes máquinas), aún suele resultar que estas computadoras pequeñas tendrán que comunicarse con la computadora principal (por ejemplo, para bajar parte de una base de datos o para subir los resultados de un cálculo departamental), y a menudo resultará que las PC o computadoras personales pequeñas o las minicomputadoras tendrán que comunicarse entre sí a través de una red local o de algún otro sistema de telecomunicaciones. Todo esto implica usualmente la interacción con el personal de operaciones; sin su aprobación sólo se podría construir un sistema realmente independiente.

Estos asuntos operacionales se documentan como parte de la tarea del análisis conocida como *modelo de puesta en práctica o implantación del usuario*. Esto se cubre con detalle en el capítulo 21.

3.8 RESUMEN

Como se vio en este Capítulo, el analista de sistemas es un orquestador, un comunicador y un facilitador. En las Partes II y III se hará evidente que el analista lleva a cabo una gran cantidad de trabajo él solo, pero que realiza aun más trabajo en armonía con los demás participantes del juego de los sistemas. Como analista, cuanto más conozca acerca de las personas con las que trabaje tanto mejor le irá.

Todos los participantes son personas y tienen diferentes metas, prioridades y perspectivas. Aunque pudieran estar públicamente comprometidos con el éxito del proyecto podrían tener razones ocultas para oponerse a uno o más aspectos de éste.

Las preguntas y ejercicios al final de este Capítulo tienen como propósito hacerle pensar más acerca de estos temas. Tal vez desee consultar el excelente libro de Block que trata de la política de los proyectos [Block, 1982] o incluso la obra clásica de Sun Tzu sobre el arte de la guerra [Tzu, 1983].

REFERENCIAS

1. Paul Strassman, *Information Payoff*. Nueva York: Free Press, 1985.
2. Robert Block, *The Politics of Projects*. Nueva York: YOURDON Press, 1982.
3. Alan Brill, *Building Controls into Structured Systems*. Nueva York: YOURDON Press, 1982.
4. Sun Tzu, *El arte de la guerra*. Nueva York: Delacorte Press, 1983.

5. Edward De Bono, *Lateral Thinking*. Nueva York: Penguin Books, 1970.
6. Marjorie Leeson, *Systems Analisis and Design*. Chicago: Science Research Associates, 1981.
7. Lavette C. Teague, Jr. y Christopher Pidgeon, *Structured Analysis Methods for Computer Information Systems*. Chicago: Science Research Associates, 1985.

PREGUNTAS Y EJERCICIOS

1. Mencione por lo menos un participante adicional con el que pudiera interactuar en un proyecto de desarrollo de sistemas.
2. Describa un proyecto en el cual el analista no tenga contacto directo con el verdadero usuario. ¿Cuáles son las ventajas y desventajas de esta situación? ¿Qué otros arreglos alternos pudieran haberse hecho?
3. ¿Se le ocurre algún otro término que pueda usarse para el usuario, además de propietario o cliente?
4. ¿Se le ocurre alguna situación donde el analista no debiera hablar con el usuario?
5. ¿Qué ventajas y desventajas se tendrían al ser el usuario miembro de tiempo completo del equipo del proyecto de desarrollo del sistema? ¿Se le ocurre algún proyecto específico en el que sería particularmente positivo incluir a un usuario en el equipo?
6. ¿Cuáles son las ventajas y desventajas de que el usuario sea administrador del equipo encargado del proyecto de desarrollo del sistema? ¿Se le ocurre algún proyecto específico donde fuera muy positivo tener de administrador del proyecto a un usuario?
7. ¿Cuáles son las ventajas y desventajas de que el usuario desarrolle el sistema de información él solo? ¿Se le ocurre algún proyecto donde fuera bueno que el usuario hiciera las veces de analista, diseñador, programador y administrador?
8. ¿Cuánto tendría que saber el usuario de computadoras y software para poder participar en un equipo de proyecto durante la fase de análisis del sistema? ¿Cuánto tendría que saber de las herramientas y técnicas del análisis de sistemas?
9. ¿Cuánto tendría que saber un usuario acerca de las computadoras y el software para poder administrar un equipo de proyecto de desarrollo de sistemas? ¿Cuánto necesitara saber acerca del análisis de sistemas para ser buen administrador?

10. ¿Cuánto debe saber un usuario de computadoras y software para poder llevar a cabo él solo un proyecto de desarrollo de sistemas? ¿Cuánto debiera saber acerca del análisis de sistemas?

11. ¿Qué precauciones especiales tomaría como analista de sistemas si no tuviera contacto directo con el usuario? ¿Cree que serían suficientes las herramientas descritas en este libro?

12. En la sección 3.1.2 se mencionan varias de las preocupaciones que pudiera tener el usuario operacional acerca de un sistema nuevo. Mencione las tres más probables. ¿Cree que estas preocupaciones son razonables o que sólo reflejan la típica falta de familiaridad del usuario con las computadoras?

13. ¿Qué responsabilidad ética y moral tiene el analista con el usuario operacional si el primero está convencido de que no causará despidos, pero el usuario se preocupa por la posibilidad de que sí los cause? (Véase también la pregunta 19)

14. Describa el escenario en el que los usuarios operacionales pudieran ocasionar que un nuevo sistema no funcione. ¿Cree que su escena sea realista? ¿No podría el usuario supervisor simplemente ordenar que se use el sistema?

15. ¿Cuándo cree que deban discutirse con los usuarios los asuntos relacionados con la interfaz humana? ¿Al comienzo del proyecto? ¿A finales de éste? ¿Cuál es la interacción indicada? (Puede consultar el capítulo 21 si lo desea).

16. ¿Cree que sea poco realista que los usuarios operacionales tengan sólo un panorama local del sistema en el que participan? Cree que sea seguro para el analista dar por un hecho esto? ¿Cree que esto sea positivo? ¿Debiera tratar el analista de proporcionar un panorama global a los usuarios operacionales?

17. Dé un ejemplo del panorama físico de un sistema o de su implantación, que pudiera tener el usuario operacional. ¿Le encuentra algún problema a esto?

18. ¿Qué debe hacer el analista si el usuario supervisor no le permite hablar directamente con los usuarios operacionales? Cómo puede el analista manejar esta situación?

19. ¿Qué responsabilidad ética o moral tiene el analista con el usuario supervisor si los usuarios operacionales le expresan su preocupación acerca de posibles despidos ocasionados por el nuevo sistema? (Véase la pregunta 13.)

20. Dé un ejemplo de un sistema en el que el usuario supervisor pueda no estar familiarizado con la política detallada de negocios a la que se estén ateniendo los usuarios.

21. ¿Por qué los usuarios típicos del nivel ejecutivo normalmente no se interesan por el posible ahorro que representaría la reducción del personal, que se hará posible con la puesta en práctica o la implantación del nuevo sistema?

22. ¿Qué tanto se deben involucrar los usuarios del nivel ejecutivo en el desarrollo de un nuevo sistema de información?

23. ¿Qué opciones tiene el analista si el usuario no entiende los modelos abstractos en el papel?

24. ¿Cómo deberá hacerse cargo el analista del "novato presuntuoso" descrito en este capítulo? ¿Qué hacer si el usuario insiste en un determinado hardware o software para el nuevo sistema?

25. ¿Qué tanta responsabilidad debe asumir el analista por la obtención del consenso de los usuarios? ¿Qué tal si el analista no logra hacerlo?

26. ¿Qué riesgos cree que afronta el analista provenientes de la administración, según se expuso en la sección 3.2? ¿Qué puede hacer el analista para minimizar estos riesgos?

27. ¿Qué debe hacer el analista si las metas y prioridades de la administración entran en conflicto con las de los usuarios?

28. ¿Cuándo cree que deba hacerse participante en proyecto a la gente de operaciones?

29. ¿Debe la misma persona (o el mismo grupo de personas) llevar a cabo tanto el análisis como el diseño (y la programación) del sistema? ¿Cuáles son las ventajas y desventajas?

4 HERRAMIENTAS DEL ANALISIS ESTRUCTURADO

La naturaleza cuenta con una especie de sistema coordenado geométrico-aritmético, pues tiene toda clase de modelos. Nuestra experiencia de la naturaleza es por medio de modelos y todos son muy hermosos. Me di cuenta de que el sistema de la naturaleza debe ser una verdadera belleza, pues encontramos en química que las asociaciones siempre se dan en lindos números enteros. No hay fracciones.

R. Buckminster Fuller
De "In The Outlaw Area: Profile by Calvin Tomkins" (En zona proscrita: semblanza por Calvin Tomkins), *The New Yorker*, 8 de enero de 1966.

El hombre es un animal que emplea herramientas... Sin ellas nada es, con ellas lo es todo.

Thomas Carlyle,
Sartor Resartus, Tomo I, capítulo 4.

En este capítulo se aprenderá:

1. Para qué utiliza las herramientas de modelado un analista.
2. La naturaleza y componentes de un diagrama de flujo de datos.
3. Los componentes de un diccionario de datos.
4. Los componentes de una especificación de proceso.
5. Cómo modelar datos almacenados y relaciones entre datos.
6. Cómo modelar el comportamiento dependiente del tiempo de un sistema.
7. Cómo modelar la estructura de un programa de computadora.

Gran parte de la labor que desempeñará como analista involucra el *modelado* del sistema que desea el usuario. Como veremos en este capítulo y con más detalle en la parte II, hay muchos tipos diferentes de modelos que podemos elaborar, así como hay muchos modelos diferentes que puede hacer de una casa nueva un arquitecto. Los modelos de análisis de sistemas que se discuten en este libro son, en su mayoría, modelos en *papel* del futuro sistema, es decir, representaciones abstractas de lo que al final será una combinación de hardware y software de computadora.

El término "modelo" pudiera parecerle algo formal y atemorizante, pero representa un concepto que usted ha manejado durante la mayor parte de su vida. Considere los siguientes tipos de modelos:

- *Mapas:* modelos bidimensionales de nuestro mundo en que vivimos.
- *Globos terráqueos:* modelos tridimensionales de nuestro mundo.
- *Diagramas de flujo:* representaciones esquemáticas de las decisiones y la secuencia de actividades para llevar a cabo un determinado procedimiento.
- *Dibujos arquitectónicos:* representaciones esquemáticas de un edificio, o de un puente, etcétera.
- *Partituras musicales:* representaciones gráficas y textuales de notas musicales y tiempos de una pieza musical.

Aunque no sepa leer el modelo arquitectónico que se muestra en la figura 4.1, el concepto de dicho modelo no debería asustarle; no es demasiado difícil imaginarse que pudiera aprender a leer y entender tales modelos, aun si jamás piensa crear uno usted mismo. De manera similar, probablemente no sepa aún leer o interpretar muchos de los modelos usados por los analistas de sistemas, pero sabrá leerlos y crearlos cuando termine de leer este libro. Los usuarios con los que trabaja podrán ciertamente leer los modelos (con una pequeña ayuda inicial) y pudieran incluso ser capaces de crearlos.

¿Por qué construimos modelos? ¿Por qué no se construye simplemente el sistema mismo? La respuesta es que podemos construir modelos de manera tal que enfatizamos ciertas propiedades críticas del sistema, mientras que simultáneamente desacentuamos otros de sus aspectos. Esto nos permite comunicarnos con el usuario de una manera enfocada, sin distraernos con asuntos y características ajenas al sistema. Y si nos damos cuenta de que nuestra comprensión de los requerimientos del usuario no fue la correcta (o de que el usuario cambió de parecer acerca de sus requerimientos), *podemos hacer cambios en el modelo o desecharlo y hacer uno nuevo, de ser necesario.* La alternativa es tener algunas reuniones preliminares con el usuario y luego construir todo el sistema; desde luego, existe el riesgo de que el producto final no sea aceptable, y pudiera ser excepcionalmente costoso hacer un cambio a esas alturas.

Por esta razón, el analista hace uso de herramientas de modelado para:

- Concentrarse en las propiedades importantes del sistema y al mismo tiempo restar atención a otras menos importantes.
- Discutir cambios y correcciones de los requerimientos del usuario, a bajo costo y con el riesgo mínimo.
- Verificar que el analista comprenda correctamente el ambiente del usuario y que lo haya respaldado con información documental para que los diseñadores de sistemas y los programadores puedan construir el sistema.

No todas las herramientas de modelado cumplen con estos propósitos: por ejemplo, una descripción narrativa de 500 páginas de los requerimientos del usuario

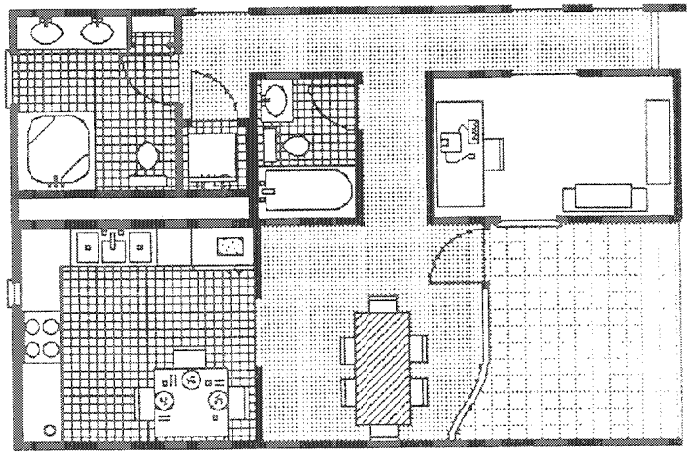


Figura 4.1: **Un modelo arquitectónico típico**

(que es, grosso modo, un modelo) podría 1) Contribuir a obscurecer *todas* las propiedades del sistema, 2) Costar más en su elaboración que el sistema mismo y 3) no verificar si el analista comprendió o no las necesidades reales del usuario. En el capítulo 8 se explorarán con más detalle las características que debe tener una herramienta de modelado para serle útil al analista.

Ahora, presentaremos y discutiremos brevemente tres herramientas de modelado de sistemas importantes: el diagrama de flujo de datos, el diagrama de entidad-relación y el diagrama de transición de estados. El diagrama de flujo de datos ilustra las *funciones* que el sistema debe realizar; los diagramas de entidad-relación hacen énfasis en las *relaciones* entre los datos y el diagrama de transición de estados se

enfoca al comportamiento dependiente del tiempo del sistema. Los capítulos 9 al 16 tratan estas y otras herramientas con más detalle. Como veremos, las tres herramientas principales consisten en gráficas (imágenes) y herramientas textuales adicionales. Las gráficas proporcionan una manera fácil de leer para que el analista pueda mostrarle a los usuarios las principales *componentes* del modelo, al igual que las *conexiones* (o interfases) entre componentes. Las herramientas de modelado textuales adicionales presentan definiciones precisas del *significado* de los componentes y conexiones.

4.1 MODELADO DE LAS FUNCIONES DEL SISTEMA: EL DIAGRAMA DE FLUJO DE DATOS

Un viejo adagio de la profesión de desarrollo de sistemas dice que un sistema de proceso de datos involucra tanto los datos como el proceso, y no se puede construir un sistema exitoso sin considerar ambos componentes. El aspecto de proceso de un sistema ciertamente es algo importante de modelar y de verificar con el usuario. El modelado que llevamos a cabo puede describirse en una variedad de maneras:

- ¿Con qué funciones debe desempeñar el sistema? ¿Cuáles son las interacciones entre dichas funciones?
- ¿Qué transformaciones debe llevar a cabo el sistema? ¿Qué entradas se transforman en qué salidas?
- ¿Qué tipo de labor debe realizar el sistema? ¿De dónde obtiene la información para llevar a cabo dicha labor? ¿Dónde entrega los resultados de su labor?

La herramienta de modelado que utilizamos para describir la transformación de entradas a salidas es un *diagrama de flujo de datos*. En la figura 4.2 se muestra un diagrama de flujo de datos típico.

Los diagramas de flujo de datos consisten en procesos, agregados de datos, flujos y terminadores:

- Los *procesos* se representan por medio de círculos, o "burbujas", en el diagrama. Representan las diversas funciones individuales que el sistema lleva a cabo. Las funciones transforman entradas en salidas.
- Los *flujos* se muestran por medio de flechas curvas. Son las conexiones entre los procesos (funciones del sistema) y representan la información que dichos procesos requieren como entrada o la información que generan como salida.
- Los *agregados de datos* se representan por medio de dos líneas paralelas o mediante una elipse. Muestran colecciones (o agregados) de datos que

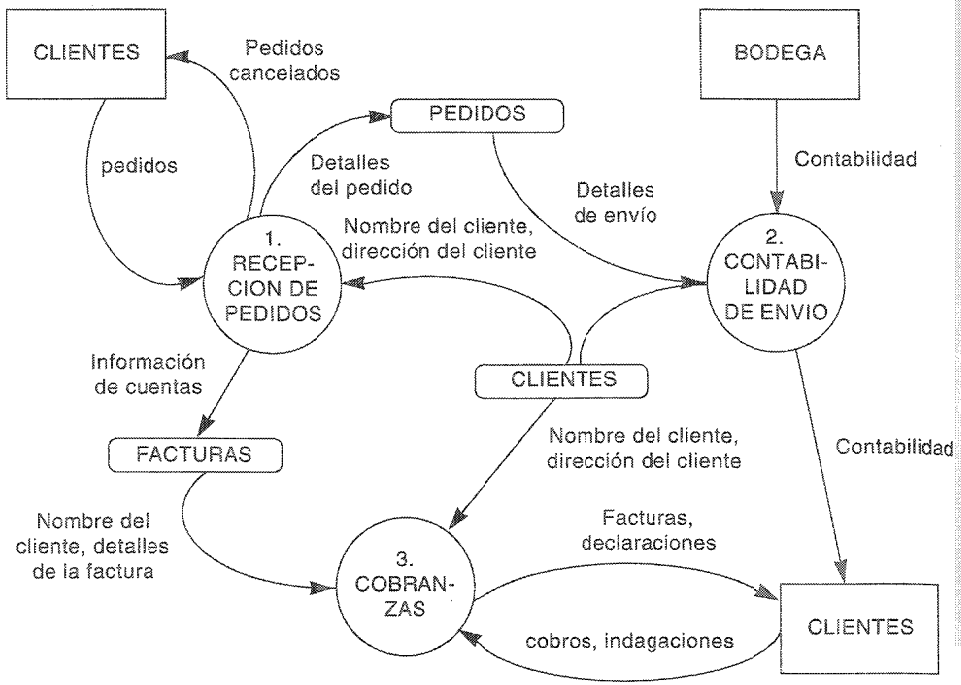


Figura 4.2: **Un diagrama de flujo de datos típico**

el sistema debe recordar por un periodo de tiempo. Cuando los diseñadores de sistemas y los programadores terminan de construir el sistema, los agregados existirán como archivos o bases de datos.

- Los *terminadores* muestran las entidades externas con las que el sistema se comunica. Típicamente se trata de individuos o grupos de personas (por ejemplo, otro departamento o división dentro de la organización), sistemas de cómputo externos y organizaciones externas.

Los diagramas de flujo de datos se discuten con más detalle en el capítulo 9. (Además de los procesos, flujos y agregados, un diagrama de flujo de datos puede tener también *flujos de control, procesos de control, y agregados de control.* Estos resultan útiles para modelar los sistemas de tiempo real y se discuten con más detalle en el capítulo 9.)

Aunque el diagrama de flujo de datos proporciona una visión global bastante conveniente de los componentes funcionales del sistema, no da detalles de éstos. Para mostrar detalles acerca de *qué* información se transforma y de *cómo* se transforma, se ocupan dos herramientas textuales de modelado adicionales: El *diccionario de datos* y la *especificación de procesos.* La figura 4.3 muestra un diccionario de datos típico para el diagrama de flujo de datos que vimos en la figura 4.2. De manera similar, la figura 4.4 muestra una especificación de proceso típica para *un solo proceso* del diagrama de flujo de datos de la figura 4.2.

Nombre =	**Tratamiento de cortesía o título + nombre + apellidos**
Tratamiento de cortesía o título =	[Sr. | Srta. | Sra. | Dr. | Prof.]
Nombre =	**{carácter válido}**
Apellido =	**{carácter válido}**
Carácter válido =	[A-Z | a-zl ' | - | |]

Figura 4.3: **Un diccionario de datos típico**

1. **Si** el monto en dólares de la factura multiplicado por el número de semanas de retraso en el pago rebasa los 10,000 dólares **ENTONCES:**
   a. Proporcionar una fotocopia de la factura al encargado de ventas que llamará al cliente.
   b. Anotar en el reverso de la factura que se le dio una copia al vendedor, junto con la fecha en la que se hizo esto.
   c. Volver a archivar la factura para estudiarla de nuevo dentro de dos semanas.
2. **EN CASO CONTRARIO, SI** se han enviado más de cuatro recordatorios **ENTONCES:**
   a. Dar una copia de la factura al vendedor apropiado para que llame al cliente.
   b. Registrar en el reverso de la factura que una copia ha sido enviada al vendedor, y la fecha en la que se hizo esto.
   c. Volver a archivar la factura para reexaminarla dentro de una semana.
3. **EN CASO CONTRARIO** (la situación aún no ha alcanzado proporciones serias):
   a. Añadir 1 al contador de avisos de moratoria registrado en el inverso de la factura (si no se ha registrado tal contador, escribir: "cuenta vencida de avisos de moratoria = 1")

b. Si la factura archivada es ilegible **ENTONCES** mecanografiar una nueva.

c. Enviar una copia de la factura al cliente, con el sello: "n-ésimo aviso: pago de factura vencido. Favor de remitir inmediatamente", donde n es el valor de avisos de moratoria.

d. Registrar en el reverso de la factura la fecha en la que se envió el n-ésimo aviso de moratoria.

e. Volver a archivar la factura para examinarla dentro de dos semanas.

Figura 4.4: **Especificación de proceso típica**

Queda mucho que decir acerca de los diagramas de flujo de datos, los diccionarios de datos y las especificaciones de procesos; en los capítulos 9 a 11 se presentan más detalles de esto. Veremos, por ejemplo, que la mayoría de los sistemas complejos se modelan con más de un diagrama de flujo de datos; de hecho, pudiera haber docenas o centenares de diagramas, acomodados de acuerdo con niveles de jerarquía. Y veremos también que hay convenciones para la manera de etiquetar y numerar los elementos del diagrama, y también hay guías y reglas que permiten distinguir entre diagramas buenos y malos.

4.2 EL MODELADO DE DATOS ALMACENADOS: EL DIAGRAMA DE ENTIDAD-RELACION

Aunque el diagrama de flujo de datos es una herramienta muy útil para modelar sistemas, sólo resalta un aspecto principal de un sistema: sus funciones. La notación de los agregados de datos en los diagramas de flujo de datos muestra la existencia de uno o más grupos de datos almacenados, pero deliberadamente dice muy poco acerca de sus detalles.

Todos los sistemas almacenan y usan información acerca del ambiente en el cual interactúan; a veces, la información es mínima, pero en la mayoría de los sistemas actuales es bastante compleja. No sólo deseamos conocer en detalle qué información hay en cada agregado de datos, sino que también queremos conocer la relación que existe *entre* agregados. Este aspecto del sistema no es resaltado por el diagrama de flujo de datos, pero sí lo hace otra herramienta: el *diagrama de entidad-relación*. La figura 4.5 muestra un diagrama típico de entidad-relación.

El diagrama de entidad-relación consta de dos componentes principales:

1. *Tipos de objetos.* Se representan por medio de un rectángulo en el diagrama. Esto representa una colección o conjunto de objetos (cosas) del mundo real cuyos miembros juegan algún papel en el desarrollo del sistema; pueden además ser identificados de manera única y ser descritos por uno o más atributos.

2. *Relaciones.* Se representan por medio de rombos en el diagrama y son la serie de conexiones o asociaciones entre los tipos de objetos que están conectados con la relación por medio de flechas.

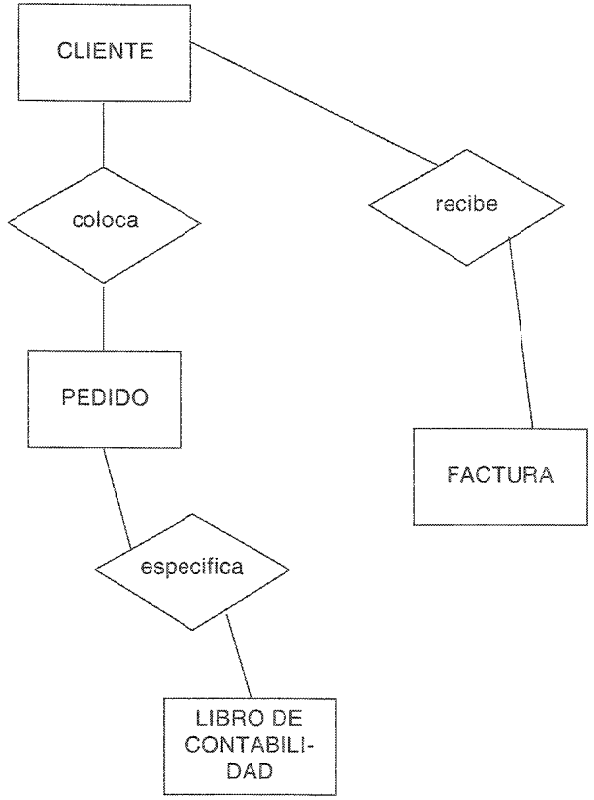


Figura 4.5: **Un diagrama de entidad-relación**

Al igual que con el diagrama de flujo de datos, hay mucho qué decir acerca del diagrama de entidad-relación y se tratará con más detalle en el capítulo 12. Veremos que existen ciertos tipos de objetos especializados, así como guías para asegurar que el diagrama de entidad-relación sea completo y congruente.

Al igual que con el diagrama de flujo de datos, es necesario acompañar el diagrama de entidad-relación con información textual detallada. El diccionario de datos que vimos por primera vez en la figura 4.3 también puede usarse para mantener in-

formación apropiada acerca de objetos y relaciones. Esto se tratará más a fondo en los capítulos del 10 al 12.

4.3 EL MODELADO DEL COMPORTAMIENTO DEPENDIENTE DEL TIEMPO: EL DIAGRAMA DE TRANSICION DE ESTADOS

El comportamiento dependiente del tiempo, es decir, la secuencia con la cual se hará el acceso a los datos y se ejecutarán las funciones es un tercer aspecto de muchos sistemas complejos. Para algunos sistemas computacionales de empresas este aspecto no es importante, puesto que la secuencia es esencialmente trivial. Así, en muchos sistemas computacionales (aquellos que ni son de tempo real, ni están en línea), la función N no puede llevar a cabo su labor hasta que recibe la entrada que requiere; y esta entrada se produce como salida de una función N-1, y así sucesivamente.

Sin embargo, muchos sistemas en línea y de tiempo real, tanto en el campo de los negocios como en el de la ciencia y la ingeniería, tienen complejas relaciones en el tiempo que deben modelarse tan cuidadosamente como las funciones y las relaciones entre datos. Por ejemplo, muchos sistemas de tiempo real deben responder dentro de un margen breve, posiblemente de tan sólo unos microsegundos, a ciertas entradas provenientes del ambiente exterior. Y deben estar preparados para recibir diversas combinaciones y secuencias de entradas a las cuales se debe responder adecuadamente.

La herramienta de modelado que utilizamos para describir este aspecto del comportamiento de un sistema es el *diagrama de transición de estados*, que a veces se abrevia (por sus siglas en inglés) STD. Un diagrama típico se muestra en la figura 4.6: modela el comportamiento de una lavadora controlada por computadora. En este diagrama, los rectángulos representan los *estados* en los que se puede encontrar el sistema (por ejemplo, "escenarios" o "situaciones" reconocibles). Cada estado representa entonces un periodo durante el cual el sistema sigue algún comportamiento observable; las flechas que conectan un rectángulo con otro representan el cambio de estado o transiciones de un estado a otro. Hay una o más *condiciones* (sucesos o circunstancias que propiciaron el cambio de estado) asociadas con cada cambio de estado, y una o más (o tal vez ninguna) *acciones*, es decir respuestas, salidas o actividades que se llevan a cabo como parte del cambio de estado. En el capítulo 13 se examinará con más detalle el diagrama de transición de estados.

4.4 EL MODELADO DE LA ESTRUCTURA DE LOS PROGRAMAS: EL DIAGRAMA DE ESTRUCTURAS

Aunque no las usará mucho como analista de sistemas, debe estar al tanto de que se utilizan muchas herramientas de modelado adicionales durante la creación de un sistema complejo. Por ejemplo, los diseñadores de sistemas suelen utilizar los diagramas de flujo de datos, diccionarios de datos, especificaciones de procesos y

diagramas de entidad-relación y de transición de estados creados por el analista para crear una arquitectura de software, es decir, una jerarquía de módulos (los que a veces se conocen como subrutinas o procedimientos) para realizar los requerimientos del sistema. Una herramienta gráfica de modelado comúnmente utilizada para representar tal jerarquía de software es el *diagrama de estructuras*; en la figura 4.7 se muestra uno típico. En este diagrama, cada rectángulo representa un *módulo* (por ejemplo, una subrutina de FORTRAN, un procedimiento de Pascal, o un párrafo o subprograma de COBOL). Las flechas que conectan los rectángulos representan las invocaciones de módulos (por ejemplo, llamados de subrutinas o llamados de procedimientos). El diagrama también muestra los parámetros de entrada que se le dan a cada módulo invocado, y los parámetros de salida devueltos por cada módulo cuando termina su labor y le devuelve el control al que lo llama.

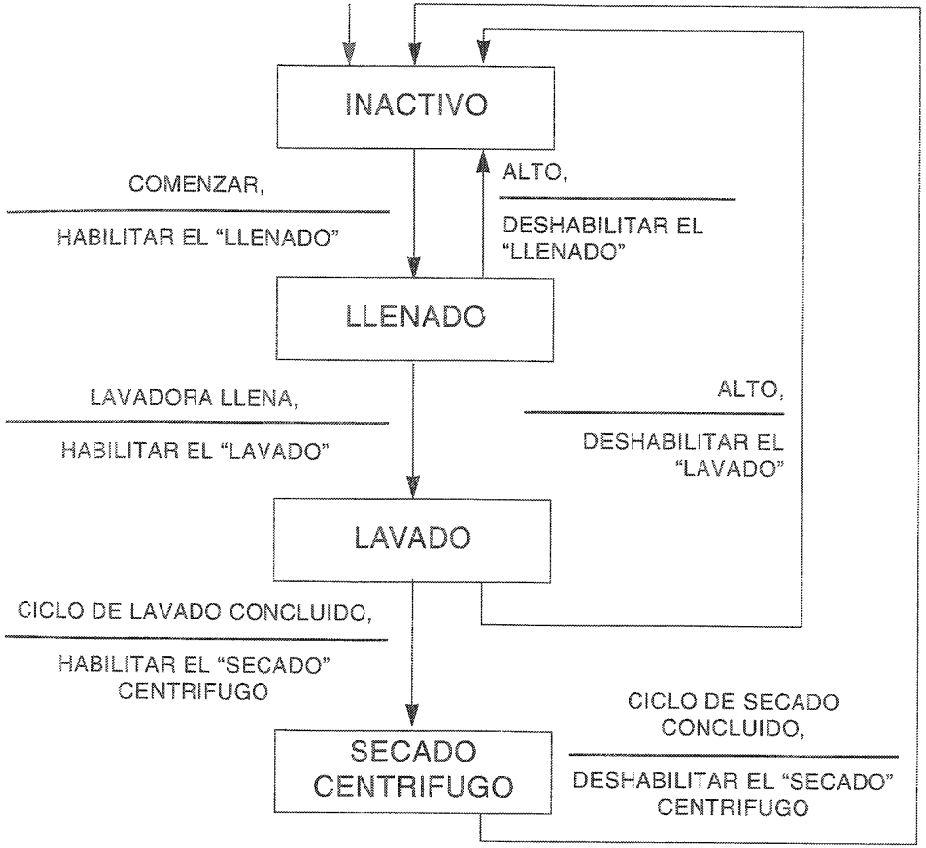


Figura 4.6: **Un diagrama de transición de estados**

A pesar de que el diagrama de estructuras es una herramienta excelente para los diseñadores de sistemas, no es el tipo de modelo que normalmente se mostraría al usuario, pues modela un aspecto de la *implantación* del sistema, no de sus requerimientos.[1] Discutiremos nuevamente los diagramas de estructuras en el capítulo 22.

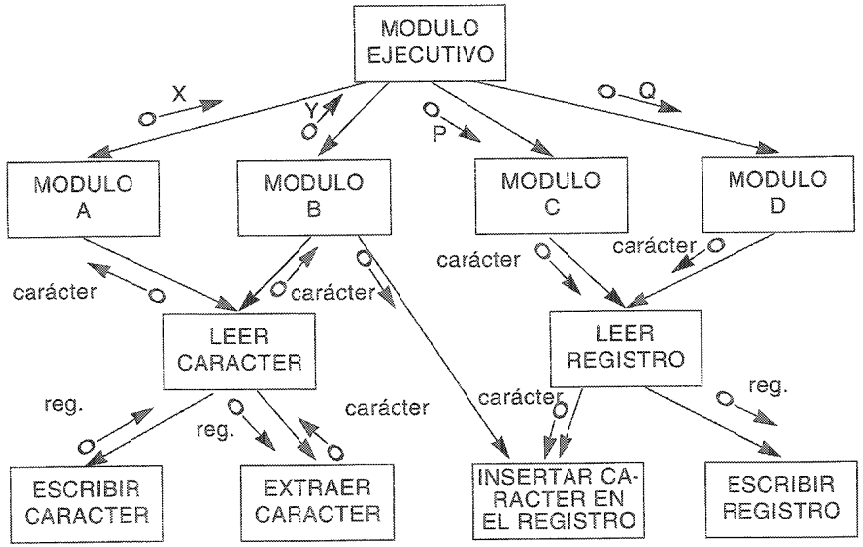


Figura 4.7: **Un diagrama de estructuras**

4.5 RELACIONES ENTRE MODELOS

Como podrá verse, cada modelo gráfico descrito en este capítulo se enfoca a un aspecto distinto de un sistema: el diagrama de flujo de datos ilustra las funciones, el diagrama de entidad-relación resalta las relaciones entre datos y el diagrama de transición de estados resalta el comportamiento dependiente del tiempo del sistema. Dado que los sistemas típicos son muy complejos, es útil estudiar por separado cada uno de estos aspectos. Por otro lado, estos tres panoramas del sistema deben ser consistentes y compatibles entre sí.

1 Como se señaló en el Capítulo 3, algunos usuarios saben más que otros de computadoras, y algunos usuarios desempeñan un papel más activo en los proyectos de desarrollo de sistemas que otros. Si está trabajando con un usuario que es miembro de tiempo completo del equipo del proyecto, o tal vez incluso si es el jefe del proyecto, y si es conocedor del diseño de sistemas, no hay razón por la que no deba mostrarle un diagrama de estructuras. Sin embargo, si el usuario sólo se interesa por describir qué tiene que hacer el sistema, probablemente no le interese ver un diagrama que describa la organización de las subrutinas de FORTRAN que cubrirán dichos requisitos.

En el capítulo 14 se examinarán algunas reglas de consistencia que aseguran que esta característica esté presente en los tres principales modelos de sistemas, junto con sus modelos textuales detallados. Por ejemplo, veremos que cada agregado del diagrama de flujo de datos debe corresponder un objeto o relación del diagrama de entidad-relación.

4.6 RESUMEN

Los modelos que se muestran en este capítulo son relativamente simples y fáciles de leer o interpretar. Se tuvo mucho cuidado para lograr esto, pues la intención es que puedan leerlos los usuarios sin gran preparación. Sin embargo, como veremos en los capítulos 9 a 16, se requiere mucho trabajo y cuidado para crear diagramas y asegurarse de que sean completos y consistentes, y que representen de manera precisa los requerimientos del usuario.

PREGUNTAS Y EJERCICIOS

1. La introducción al capítulo 4 menciona mapas, globos terráqueos, diagramas de flujo de datos, dibujos arquitectónicos y partituras musicales como ejemplos de modelos. Mencione otros tres ejemplos de modelos usados comúnmente.

2. ¿Qué definición da el diccionario de la palabra "modelo"? ¿Se puede aplicar a los sistemas de información?

3. ¿Por qué se utilizan modelos en el desarrollo de los sistemas de información? Mencione tres razones.

4. ¿Cómo respondería si el usuario le dijera que opina que los modelos son un desperdicio de tiempo y que debería empezar a codificar?

5. ¿Cree que la descripción *verbal* que un usuario dé acerca de los requerimientos de su sistema deba considerarse como un modelo? ¿Por qué sí o por qué no?

6. ¿Por qué sería útil tener más de un modelo para un sistema?

7. Todos los modelos que se discuten en el capítulo 4 son modelos en papel. ¿Se le ocurre algún otro tipo de modelo?

8. La mayoría de los modelos que se discuten en el capítulo 4 son herramientas gráficas (por ejemplo, imágenes) ¿Cuáles cree que puedan ser las ventajas de utilizar imágenes como herramientas de modelado?

9. ¿Considera que las herramientas de modelado gráfico sean suficientes para representar los requerimientos de un sistema de información? ¿Por qué sí o por qué no?

10. ¿Quién debería ser el responsable de la creación de los modelos necesarios para describir los requerimientos de un sistema de información?

11. ¿Debería esperarse que los usuarios crearan sus propios modelos? De ser así, ¿en qué circunstancias?

12. ¿Cuáles son los tres principales objetivos de un diagrama de flujo de datos?

13. ¿Cuáles son los cuatro principales componentes de un diagrama de flujo de datos?

14. Nótese que los procesos se representan por medio de círculos en un diagrama de flujo de datos y los terminadores se representan con rectángulos. ¿Cree que esto sea significativo? ¿Qué pasaría si los procesos se representaran?

15. Nótese que la figura 4.2 muestra tres diferentes procesos, pero no indica cuántas computadoras puedan estar trabajando en el sistema. ¿Cree que esto sea significativo? ¿Qué cambios se requerirían si el equipo encargado del proyecto decidiera implantar el sistema con una sola computadora? ¿Y con tres?

16. Nótese que la figura 4.2 muestra varios distintos diagramas de flujo de datos entre procesos, pero no indica el medio físico que se usará para transportar los datos. ¿Cree que esto sea significativo? ¿Qué puede ocurrir si los realizadores del sistema deciden transportar datos entre procesos utilizando líneas de telecomunicación? ¿Qué pasa si deciden transportarlos de un proceso a otro utilizando cinta magnética?

17. ¿Cuál es el propósito del diccionario de datos?

18. ¿Quién debiera encargarse de crear el diccionario de datos? ¿Quién debería ser responsable de mantenerlo al día?

19. La figura 4.3 muestra la definición que da el diccionario de datos de un nombre. ¿Qué cree que puedan significar los paréntesis, (), en dicha definición?

20. ¿Cuál es el propósito de la especificación de procesos?

21. ¿Cuántas especificaciones de proceso debería esperar ver en una especificación completa de los requerimientos del usuario?

22. ¿Quién debería encargarse de la especificación de procesos? ¿Quién debería actualizarla?

23. Nótese que la especificación de procesos mostrada en el ejemplo de este capítulo se parece en algo a la codificación de programas. ¿Qué piensa de la idea de usar pseudocódigo para escribir las especificaciones de procesos? ¿Qué piensa de la idea de utilizar un verdadero lenguaje de programación (por ejemplo Ada), como lo han sugerido muchos, para las especificaciones de programas? ¿Por qué estaría bien o mal usar un verdadero lenguaje de programación?

24. ¿Cuál es el propósito de un diagrama de entidad- relación?

25. ¿Cuáles son los principales componentes de un diagrama de entidad-relación?

26. ¿Cuántos tipos de objetos se muestran en la figura 4.5?

27. ¿Cuántas relaciones se muestran en la figura 4.5?

28. ¿Proporciona el diagrama de entidad-relación al lector alguna información sobre las funciones que lleva a cabo el sistema?

29. Proporciona el diagrama de flujo de datos al lector alguna información acerca de los tipos de objetos o sobre las relaciones entre tipos de objetos en el sistema?

30. ¿Dónde deberían describirse los detalles de tipos de objetos y relaciones que se muestran en un diagrama de entidad-relación?

31. ¿Cuál es el propósito de un diagrama de transición de estados?

32. ¿Cuáles son los componentes de un diagrama de transición de estados?

33. ¿Son útiles los diagramas de transición de estados para modelar sistemas computacionales por lotes (batch)? ¿Por qué sí o por qué no?

34. ¿Cuál es el propósito de un diagrama de estructuras?

35. ¿Cuáles son los componentes gráficos de un diagrama de estructuras?

36. ¿Es de esperarse que el usuario sea capaz de leer y entender un diagrama de estructuras? ¿Debería esperarse que el usuario sea capaz de crear uno?

37. Describa la relación existente entre un diagrama de entidad-relación y un diagrama de flujo de datos.

38. ¿Existe alguna relación entre un diagrama de flujo de datos y un diagrama de estructuras? De ser así, ¿cuál es?

5 EL CICLO DE VIDA DEL PROYECTO

Todo error humano es impaciencia, una renunciación prematura al método, una engañosa sujeción a un engaño.

Franz Kafka, Cartas

En este capítulo se aprenderá:

1. El concepto del ciclo de vida de un proyecto.
2. Las características del ciclo de vida de un proyecto clásico.
3. Las diferencias entre proyectos clásicos y semiestructurados.
4. Los componentes del ciclo de vida estructurado.
5. Las diferencias entre ciclos de vida radicales y conservadores.

Para ser un buen analista de sistemas se requiere más que simples herramientas de modelado; necesitamos *métodos*. En la profesión de desarrollo de sistemas, los términos "método", "metodología", "ciclo de vida del proyecto" y "ciclo de vida del desarrollo de sistemas" se usan de manera casi indistinta. En la parte III veremos métodos detallados de cómo efectuar el análisis de sistemas, en el contexto más amplio de un método –conocido como ciclo de vida del proyecto estructurado–, para llevar a cabo el desarrollo global de un sistema nuevo.

Antes de presentar el ciclo de vida del proyecto estructurado, es importante examinar el ciclo de vida del proyecto clásico que se trata en muchos textos y se utiliza en muchas organizaciones para el desarrollo de sistemas hoy en día, sobre todo

para identificar sus limitaciones y puntos débiles. Después de este examen haremos una breve exposición acerca del ciclo de vida del proyecto *semiestructurado:* un ciclo de vida de proyecto que incluye algunos, pero no todos los elementos del desarrollo moderno de sistemas. En seguida se presentará el ciclo de vida del proyecto *estructurado,* mostrando una visión global de las principales actividades y de cómo se interrelacionan. Por último, se verán brevemente el ciclo de vida *formador de prototipos* que popularizaron Bernard Boar, James Martin, y varios vendedores de lenguajes de programación de cuarta generación.

También exploraremos el concepto de *desarrollo iterativo* o *descendente.* En particular, se presentarán los conceptos de desarrollo iterativo *radical* y desarrollo iterativo *conservador.* Dependiendo de la naturaleza de un proyecto de desarrollo de sistemas, puede haber razones válidas para adoptar un método en lugar de otro, e incluso algunos proyectos pudieran requerir una combinacion de ambos.

5.1 EL CONCEPTO DE CICLO DE VIDA DE UN PROYECTO

Como pudiera esperarse, las organizaciones pequeñas de proceso de datos tienden a ser relativamente informales: los proyectos de desarrollo de sistemas nacen de conversaciones entre el usuario y el administrador del proyecto (que puede ser a la vez el analista, el programador, el operario y el conserje), y el proyecto procede desde el análisis hasta el diseño e implantación sin mayor alboroto.

Sin embargo, en las organizaciones más grandes, las cosas se llevan a cabo de manera mucho más formal. La comunicación entre los usuarios, la administración y el equipo del proyecto suele ser por escrito, y todo mundo entiende que el proyecto pasará por diversas fases antes de completarse. Aun así, es sorprendente ver la gran diferencia entre las maneras en que dos administradores afrontan sus respectivos proyectos. De hecho, a menudo se deja a discreción del administrador determinar las fases y actividades de su proyecto, y cómo se llevarán a cabo.[1]

Recientemente, sin embargo, ha empezado a cambiar el enfoque que se le da al desarrollo de sistemas. Cada vez son más las organizaciones grandes y pequeñas que están adoptando un ciclo de vida uniforme y único para sus proyectos. Esto a veces se conoce como el plan del proyecto, la metodología del desarrollo del sistema o, simplemente, "la forma en la que hacemos las cosas aquí". El manual del ciclo de vida del proyecto suele ser un libro tan voluminoso como el compendio de normas, que yace (usualmente sin ser leído) sobre el escritorio de todo analista y

1 Esto suena como si la anarquía prevaleciera en la mayoría de las organizaciones de proceso electrónico de datos. Sin embargo, hay dos situaciones comunes que llevan a este enfoque individualista aun en la organización más ejemplar: 1) La organización altamente descentralizada, donde cada departamento tiene su grupo de proceso electrónico de datos con sus propias normas locales y 2) el periodo de varios años tras de que el último "ciclo de vida oficial del proyecto" se juzgara inútil y se descartara.

programador. Ese manual ofrece un procedimiento común a seguir para desarrollar un sistema computacional que puede orientar a cualquier miembro de la organización de desarrollo de sistemas.

El enfoque puede ser casero o, como alternativa, pudiera ser que la organización para el desarrollo de sistemas decida comprar un paquete de administración de proyectos y ajustarlo a las necesidades de la compañía.[2] Parece obvio que, aparte de darle empleo a las personas que crean los manuales de ciclo de vida de los proyectos (y a aquellos que escriben libros de texto al respecto), es conveniente la metodología del proyecto. ¿De qué sirve entonces tener un ciclo de vida de un proyecto? Existen tres objetivos principales:

1. Definir las actividades a llevarse a cabo en un proyecto de desarrollo de sistemas.
2. Lograr congruencia entre la multitud de proyectos de desarrollo de sistemas en una misma organización.
3. Proporcionar puntos de control y revisión administrativos de las decisiones sobre continuar o no con un proyecto.

El primer objetivo es de particular importancia en una organización grande donde constantemente está ingresando personal nuevo a los puestos de administración de proyectos. El administrador novato pudiera no tomar en cuenta o subestimar la importancia de fases clave del proyecto si se basa sólo en su intuición. De hecho, pudiera suceder que los programadores y analistas de bajo rango no entiendan dónde y cómo encajan sus esfuerzos individuales en el proyecto global, a menos que se les dé una descripción adecuada de *todas* las fases del proyecto.

El segundo objetivo también es importante en una organización grande. Para los niveles más altos de la administración pudiera ser bastante confuso seguir la pista de cientos de proyectos diferentes, cada uno de los cuales se lleva a cabo de distinta manera. Por ejemplo, si el proyecto A define la actividad de análisis de sistemas de diferente forma que el proyecto B y el B no incluye una fase de diseño, ¿cómo puede darse cuenta el administrador de segundo o tercer nivel de cuál proyecto se está rezagando y cuál continúa según lo previsto?[3]

2 Existen varios de estos paquetes en el mercado, que cuestan entre $10,000 y $100,000 dólares estadounidenses (o su equivalente en moneda nacional), o más. Algunos de los ejemplos más conocidos son Spectrum (de Spectrum International Corp.), SDM-70 (de AGS Software), y Method/1 (de Arthur Andersen). No comentaré acerca de ningún paquete de administración de proyectos en particular; sólo le sugiero que tenga en mente los conceptos presentados en este libro si su organización utiliza un paquete obtenido en el mercado.

3 Miller en [Miller, 1978], señala que éste es un fenómeno comúnmente observado; de hecho, lo presenta como una "hipótesis" general aplicable a todos los sistemas en activo:

> HIPOTESIS 2-1: Los componentes de un sistema incapaces de asociarse, o que carecen de la experiencia que haya formado tales asociaciones, deben funcionar de acuerdo con una programación rígida o con reglas de operación altamente estandarizadas. Se sigue que si la rotación de los componentes rebasa el ritmo con que se están desarrollando las asociaciones necesarias para su operación, aumenta la rigidez en la programación.

El tercer objetivo de un ciclo de vida de proyecto normal se refiere a la necesidad de la administración de controlar un proyecto. En los proyectos triviales, el único punto de revisión probablemente esté al final del proyecto: ¿se concluyó a tiempo y centro de los márgenes del presupuesto acordado? (o, más simple aún, ¿se concluyó siquiera?) ¿Y cumplió con los requisitos del usuario? Pero, para proyectos más grandes, debería contarse con varios puntos intermedios de revisión, que permitieran determinar si el proyecto se estuviera retrasando o si fueran necesarios recursos adicionales. Además, el usuario inteligente también necesitará puntos de revisión en diversas etapas del proyecto para que pueda determinar si quiere continuar financiándolo.[4]

Dicho todo esto, no queda más que subrayar que el ciclo de vida del proyecto definitivamente no está a cargo del proyecto; no le evitará al administrador del proyecto la difícil labor de tomar decisiones, sopesar alternativas, librar batallas políticas, negociar con usuarios recalcitrantes, animar a programadores deprimidos, ni ninguna de las demás tribulaciones relacionadas con los proyectos. El administrador del proyecto todavía tiene que *administrar,* en todo el sentido de la palabra. La única ayuda que puede proporcionar el ciclo de vida del proyecto es que pueda *organizar* las actividades del administrador, aumentando la probabilidad de que se aborden los problemas pertinentes en el momento adecuado.

5.2 EL CICLO DE VIDA DEL PROYECTO CLASICO

El tipo de ciclo de vida de proyecto que se usa actualmente en la mayoría de las organizaciones difiere de aquel al que estaremos dedicando la mayor parte de nuestra atención en la parte III. En la figura 5.1 se muestra el ciclo de vida clásico o convencional. Cada proyecto atraviesa por algún tipo de análisis, diseño e implantación, aunque no se haga exactamente como se muestra en el diagrama. El ciclo de vida de proyecto utilizado, por ejemplo, en la organización de usted, pudiera diferir del que se muestra en la figura 5.1 en una o en todas las formas siguientes:

4 De hecho, los procedimientos de la mayor parte de los proyectos de proceso de datos son tales que existe sólo un punto de control desde el cual el usuario tiene una manera obvia y limpia de arrepentirse: al final de la fase de encuesta o del estudio de factibilidad. En teoría, sin embargo, el usuario debería tener la oportunidad de cancelar un proyecto de proceso de datos al final de cualquier fase si piensa que está desperdiciando su dinero.

Figura 5.1(a): **El ciclo de vida del proyecto clásico**

- Las fases de exploración y análisis pudieran juntarse en una sola (sobre todo en organizaciones en las cuales se considera factible desde el inicio cualquier cosa que quiera el usuario).
- Puede no haber fase de estudio de hardware si se cree que cualquier sistema nuevo pudiera instalarse con las computadoras existentes sin causar mayor problema operacional.
- Las fases de diseño preliminar y de diseño de detalles pudieran juntarse en una sola llamada simplemente de diseño.
- Diversas fases de prueba pueden juntarse en una sola; de hecho, podrían incluirse con la codificación.

De aquí que el ciclo de vida del proyecto en una organización sola puede tener cinco fases o siete o doce, pero seguir siendo todavía de tipo clásico.

¿Qué es lo que realmente caracteriza el ciclo de vida de un proyecto como clásico? Se distinguen dos aspectos: una fuerte tendencia a la implantación ascendente del sistema y la insistencia en la progresión lineal y secuencial de una fase a la siguiente.

5.2.1 Implantación ascendente

El uso de la implantación ascendente es una de las grandes debilidades del ciclo de vida de los proyectos clásicos. Como se podrá ver en la figura 5.1(a), se espera que los programadores lleven a cabo primero sus pruebas modulares, luego las pruebas del subsistema, y finalmente las pruebas del sistema mismo. Este enfoque también se conoce como el ciclo de vida de cascada, y está basado en el diagrama presentado originalmente en [Royce, 1970], y popularizado posteriormente por Barry Boehm [Boehm, 1981]. Se muestra en la figura 5.1(b).

No está claro de dónde surgió originalmente este enfoque, pero pudiera haberse tomado de las líneas de montaje de las industrias manufactureras. La implantación ascendente es buena para el montaje de automóviles en línea, *¡pero sólo después de que el prototipo esté completamente libre de fallas!*[5] Desafortunadamente, muchas organizaciones que desarrollan sistemas todavía producen sistemas únicos, para lo cual el enfoque ascendente presenta un gran número de dificultades serias:

---

5 Muchos creen que el enfoque ascendente pudiera provenir de la incustria computacional del *hardware* porque muchos de los programadores y administradores de programación de los años 50 y 60 eran ingenieros eléctrónicos que habían tenido que ver previamente con el desarrollo de hardware.

- Nada está hecho hasta que *todo* esté terminado. Por eso, si el proyecto se atrasa y la fecha límite cae precisamente en medio del proceso de prueba del sistema, no habrá nada que mostrarle al usuario más que una enorme pila de listados de programas, los cuales, vistos en su totalidad, no le ofrecen nada de valor.

- Las fallas más triviales se encuentran al comienzo del periodo de prueba y las más graves al final. Esto es casi una tautología: las pruebas modulares dejan al descubierto fallas relativamente simples dentro de los módulos individuales. Las pruebas del sistema, por otra parte, descubren errores grandes de interfaz entre subsistemas. La cuestión es que los errores de interfaz no son lo que el programador desea descubrir al final de un proyecto de desarrollo; tales fallas pueden obligar a la recodificación de un gran número de módulos, y pueden tener un impacto devastador sobre el calendario, justo en un momento en el cual es probable que todo el mundo esté algo cansado y molesto tras haber trabajado duro durante tantos meses.

- La eliminación de fallas suele ser extremadamente difícil durante las últimas etapas de prueba del sistema. Nótese que se puede distinguir entre *pruebas* y *eliminación de fallas*. La eliminación de fallas es el arte de descubrir dónde está la falla (y subsecuentemente cómo arreglarla) después de que el proceso de prueba ha determinado que la falla de hecho existe. Cuando la falla se descubre durante la fase de prueba del sistema en un proyecto ascendente, a menudo suele ser extremadamente difícil determinar cuál módulo la contiene; pudiera tratarse de cualquiera de los cientos (o miles) de módulos que se han combinado por primera vez. Localizar una falla a menudo es como hallar una aguja en un pajar.

- La necesidad de prueba con la computadora aumenta exponencialmente durante las etapas finales de prueba. Para ser más específicos, el administrador del proyecto a menudo descubre que necesita una gran cantidad de horas-máquina para probar el sistema; tal vez 12 horas de labor ininterrumpida diaria. Dado que suele ser difícil obtener tanto tiempo de uso de la computadora,[6] el proyecto suele retrasarse mucho.

5.2.2 Progresión Secuencial

La segunda debilidad más importante del ciclo de vida de un proyecto clásico es su insistencia en que las fases se sucedan secuencialmente. Querer esto es una tendencia natural humana: deseamos decir que hemos *terminado* la fase de análisis

6 Estoy convencido de que aquí se aplica otra más de las leyes de Murphy: Entre más grande y más crítico sea el proyecto, más probable es que la fecha límite coincida con el proceso de fin de año o alguna otra crisis organizacional que monopoliza todo el tiempo de computadora disponible.

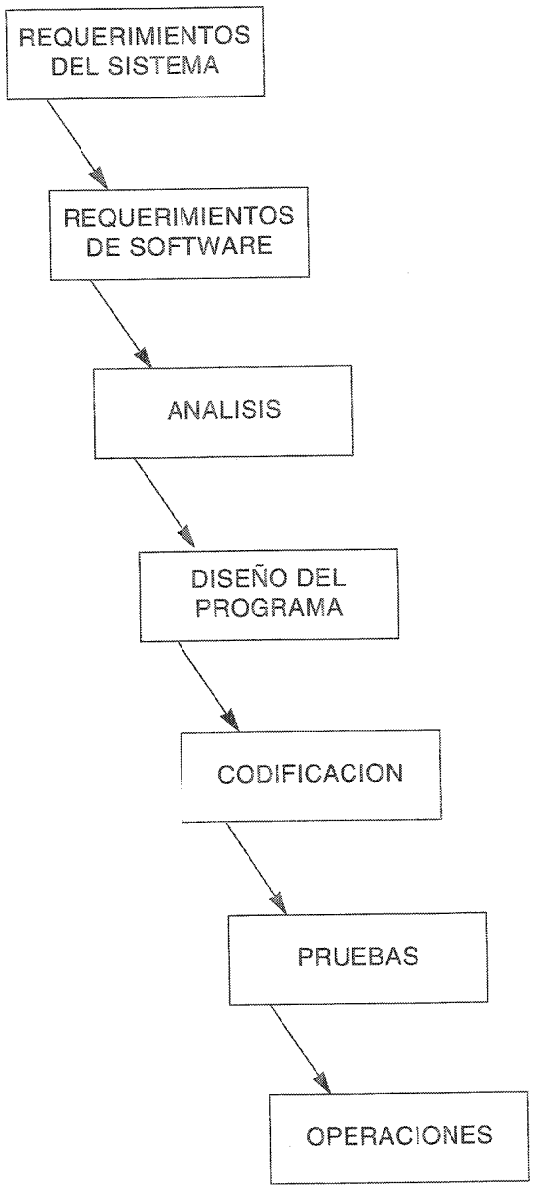


Figura 5.1(b): **El modelo de cascada del desarrollo de sistemas**

del sistema y que nunca tendremos que volver a preocuparnos por ella. De hecho, muchas organizaciones formalizan esto con un ritual conocido como "congelar" la especificación o congelar el documento de diseño.

El único problema que trae consigo este deseo de progreso ordenado es que no es nada realista. En lo particular, el enfoque secuencial no permite el tratamiento de fenómenos reales como los relacionados con el personal, la política de la compañía, o la economía. Por ejemplo, la persona que hizo el trabajo, el analista o el diseñador, pudieron haber cometido un error y haber elaborado un producto con fallas. De hecho, como humanos, rara vez atinamos a hacer bien un trabajo al primer intento, pero se suelen hacer repetidas mejoras del trabajo imperfecto. También pudiera ser que la persona que revisa el trabajo o, como caso particular, el usuario que revisa el trabajo del analista del sistema pudiera haber cometido un error. O tal vez el encargado de llevar a cabo la labor asociada con cada fase no haya tenido tiempo suficiente para terminar, pero no quiera admitirlo. Esta es una manera amable de decir que, en la mayoría de los proyectos complejos, la labor de análisis, de diseño y de prueba concluye cuando alguien decide que se ha agotado el tiempo, no cuando se quisiera.

Comúnmente surgen otros problemas asociados con el ciclo de vida del proyecto clásico o secuencial: durante los meses (o años) que toma desarrollar el sistema, el usuario pudiera cambiar de parecer respecto a lo que debe hacer el sistema. Durante el período que transcurre para desarrollar el sistema, pueden cambiar ciertos aspectos del ambiente del usuario (por ejemplo, la economía, la competencia, los reglamentos gubernamentales que afectan a las actividades del usuario).

Una característica adicional del ciclo de vida del proyecto clásico es que se apoya en técnicas anticuadas. Es decir, tiende a ignorar el uso del análisis estructurado la programación estructurada, o cualquier otra técnica moderna de desarrollo de sistemas.[7] Pero el hecho de que el ciclo de vida clásico *ignore* estas técnicas no significa que el administrador del proyecto no pueda utilizarlas. Desafortunadamente, muchos programadores, analistas y jefes de proyecto sienten que el ciclo de vida del proyecto es un mandato de la administración de alto nivel; y si la administración no dice nada al respecto del uso de la programación estructurada, entonces creen que no están obligados a utilizar métodos no clásicos.

5.3 EL CICLO DE VIDA SEMIESTRUCTURADO

Los comentarios de la sección anterior pueden hacer que parezca que la mayoría de las organizaciones de proceso de datos todavía viven en la Edad Media. De hecho, esto es injusto: no todas las organizaciones utilizan el ciclo de vida clásico. Desde fines de los años 70 y principios de los 80, ha crecido la tendencia a reconocer al diseño estructurado, la programación estructurada y la implantación descendente como parte del ciclo de vida del proyecto. Este reconocimiento ha llevado al ciclo de vida del proyecto semiestructurado que se muestra en la figura 5.2. Se muestran dos detalles obvios no presentes en el enfoque clásico:

1. La secuencia ascendente de codificación, la prueba de módulos y prueba del sistema se reemplazan por una implantación de arriba hacia abajo, que es un enfoque en el cual los módulos de alto nivel se codifican y prueban primero, seguidos por los de bajo nivel, más detallados. También hay fuertes indicios de que la programación estructurada debe usarse como método para codificar el sistema.

2. El diseño clásico se reemplaza por el diseño estructurado, que es un enfoque de diseño formal de sistemas tratado en textos tales como [Yourdon y Constantine, 1989] y [Page-Jones, 1988].

Aparte de estas diferencias obvias, hay algunos detalles sutiles acerca del ciclo de vida modificado. Por ejemplo, considere que la implantación descendente significa que se pondrán en ejecución paralelamente parte de la codificación y de las pruebas. Esto difiere mucho de las fases secuenciales que vimos en el ciclo de vida clásico. En lo particular, puede darse una *retroalimentación* entre la codificación, la prueba y la eliminación de las fallas. Cuando el programador prueba la versión de alto nivel del sistema, a veces se le puede llegar a oír exclamar: "¡Vaya, no tenía idea de que la instrucción FRAMMIS de doble precisión funcionara de esa manera!". Desde luego, se puede tener la seguridad de que en el futuro usará de manera muy diferente esta instrucción.

Tal vez sea aún más importante el hecho de que la implantación descendente pone en tentación a los ejecutores del sistema (y a los analistas si aún no han abandonado el proyecto) de no hablar con los usuarios sino hasta *después* de haberse congelado las especificaciones. Por eso, es posible que el usuario señale errores o malentendidos en la especificación, o incluso pudiera expresar el deseo de *cambiarla* y, si la conversación se da directamente entre el usuario y el que implanta, la modificación pudiera hacerse antes de que el administrador del proyecto se dé cuenta siquiera. En resumen, a menudo la implantación descendente ofrece retroalimentación entre el proceso de implantación y el de análisis, aunque esto no se muestre específicamente en la Figura 5.2, y aunque el usuario y el administrador del proyecto de proceso de datos pudieran negar que esté sucediendo.

Como último punto a tratar acerca del ciclo de vida semiestructurado, tenemos que una gran parte del trabajo que se realiza bajo el nombre de "diseño estructurado" es en realidad un esfuerzo manual para enmendar especificaciones erróneas. Esto se puede apreciar en la figura 5.3, que muestra los detalles del diseño estructurado. (Nótese que esta Figura consiste en los detalles del proceso 3 de la figura 5.2)

En la figura 5.3, la actividad 3.1 (con el título de CODIFICAR LA ESPECIFICACION FUNCIONAL) representa la labor que han tenido que desempeñar desde hace

7 Resumiremos estas técnicas modernas de desarrollo en el capítulo 7.

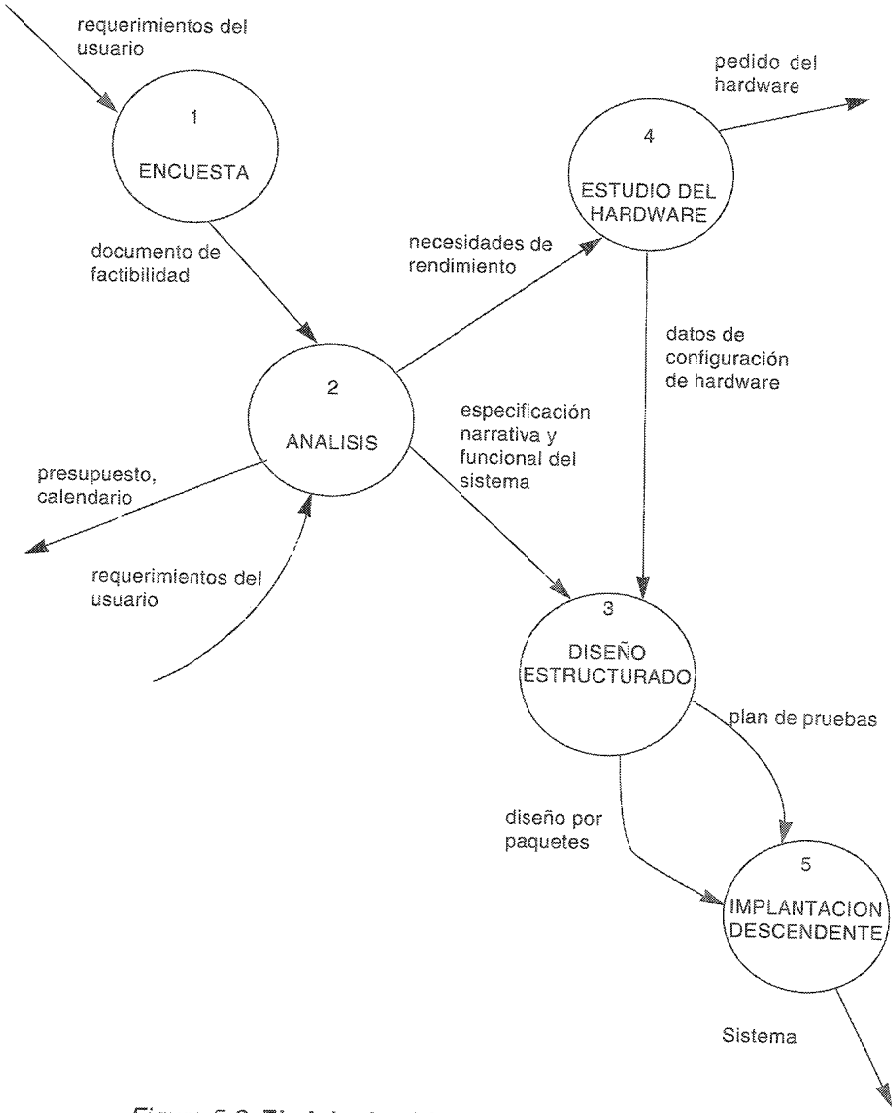


Figura 5.2: **El ciclo de vida del proyecto semiestructurado**

mucho los diseñadores: traducir un documento narrativo, ambiguo, monolítico y redundante a un modelo útil y no de procedimientos, para que sirva de base para derivar la jerarquía de módulos que ejecutarán los requisitos del usuario. En otras palabras, los que llevan a cabo el diseño estructurado han supuesto tradicionalmente que se les daría una especificación clásica; en consecuencia, su primera tarea, desde su punto de vista, es transformar la especificación en un paquete de diagramas de flujo de datos, de diccionarios de datos, de diagramas de entidad relación y de especificaciones de procesos.

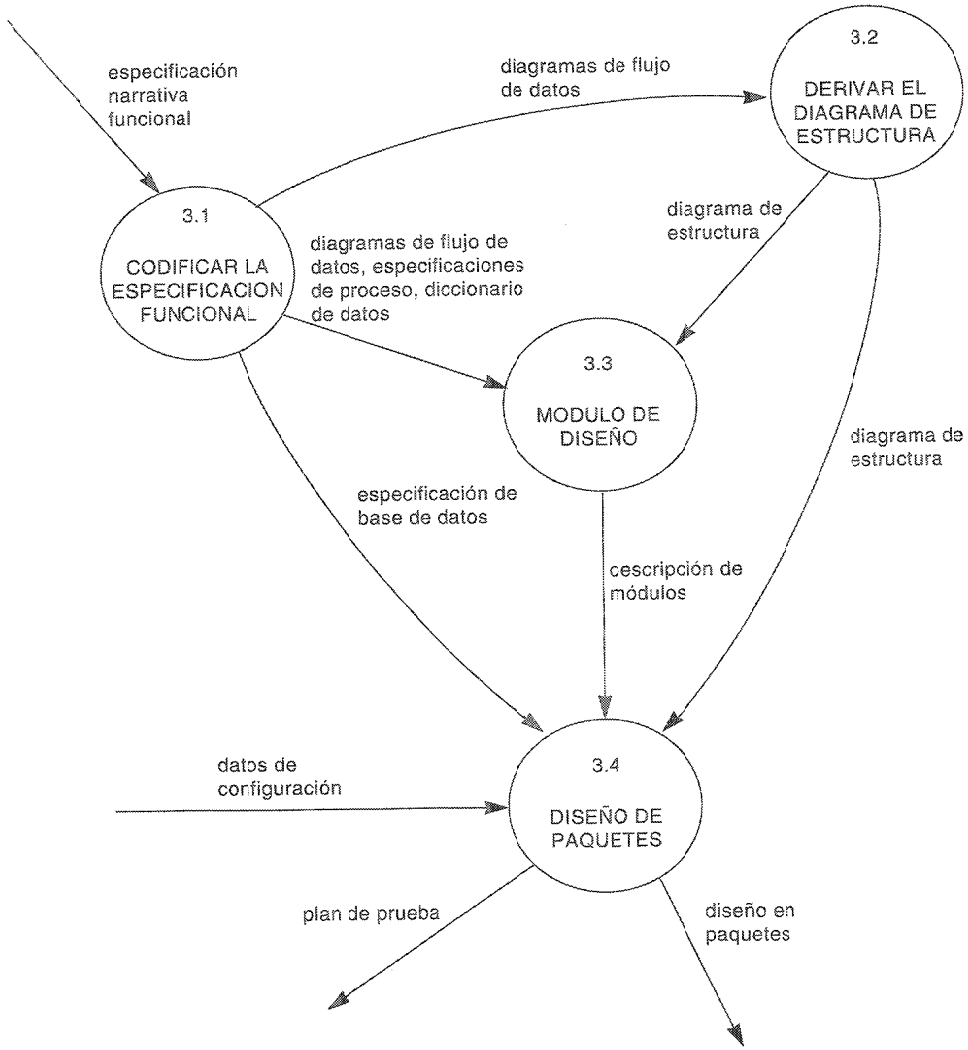


Figura 5.3: **Detalles de la actividad de diseño**

Esta labor es más difícil de lo que se pudiera imaginar: históricamente se ha llevado a cabo en el vacío. En general, los diseñadores tenían poco contacto con el analista que escribía la especificación y definitivamente *¡no* tenían contacto con el usuario!

Es obvio que esta situación amerita un cambio. Al presentar el análisis estructurado, que es el enfoque moderno de análisis de sistemas que se maneja en este libro, además de extenderse con la idea de la retroalimentación entre una parte del proyecto y otra, se crea un tipo totalmente distinto de ciclo de vida del proyecto. Este es el ciclo de vida estructurado del proyecto que discutiremos a continuación.

5.4 EL CICLO DE VIDA ESTRUCTURADO DEL PROYECTO

Ahora que ya hemos visto los ciclos de vida del proyecto clásico y semiestructurado, estamos listos para examinar el ciclo de vida estructurado, que se muestra en la figura 5.4.

Examinaremos brevemente las nueve actividades y los tres terminadores del ciclo de vida del proyecto, como se muestra en la figura 5.4. Los terminadores son los usuarios, los administradores y el personal de operaciones; como se recordará, discutimos sus papeles en el capítulo 3. Se trata de individuos o grupos que proporcionan las entradas al equipo del proyecto, y son los beneficiados finales del sistema. Ellos interactúan con las nueve actividades que se muestran en la figura 5.4 . En las siguientes secciones se resume cada una de las actividades.

5.4.1 Actividad 1: La encuesta

Esta actividad también se conoce como el estudio de factibilidad o como el estudio inicial de negocios. Por lo común, empieza cuando el usuario solicita que una o más partes de su sistema se automaticen. Los principales objetivos de la encuesta son los siguientes:

- *Identificar a los usuarios responsables y crear un "campo de actividad" inicial del sistema.* Esto puede comprender la conducción de una serie de entrevistas para determinar qué usuarios estarán comprendidos en (o serán afectados por) el proyecto propuesto.[8] Pudiera también implicar el desarrollo de un diagrama inicial de contexto, que es un diagrama de flujo de datos sencillo del tipo que se muestra en la figura 4.2, en el cual se representa el sistema completo con un solo proceso.[9]

8 Las técnicas de encuesta se discuten en el Apéndice E.

9 El diagrama de contexto es parte del modelo ambiental que se discutirá con mayor detalle en el capítulo 18. Su principal propósito, como se indica aquí, es definir cuánto abarca el sistema, así como los diversos terminadores (personas, unidades organizacionales, otros sistemas de cómputo, etc.) con los que el sistema interactuará.

- *Identificar las deficiencias actuales en el ambiente del usuario.* Esto en general comprenderá la lista de funciones que hacen falta o que se están llevando a cabo insatisfactoriamente en el sistema actual. Por ejemplo, esto pudiera incluir declaraciones como las siguientes:

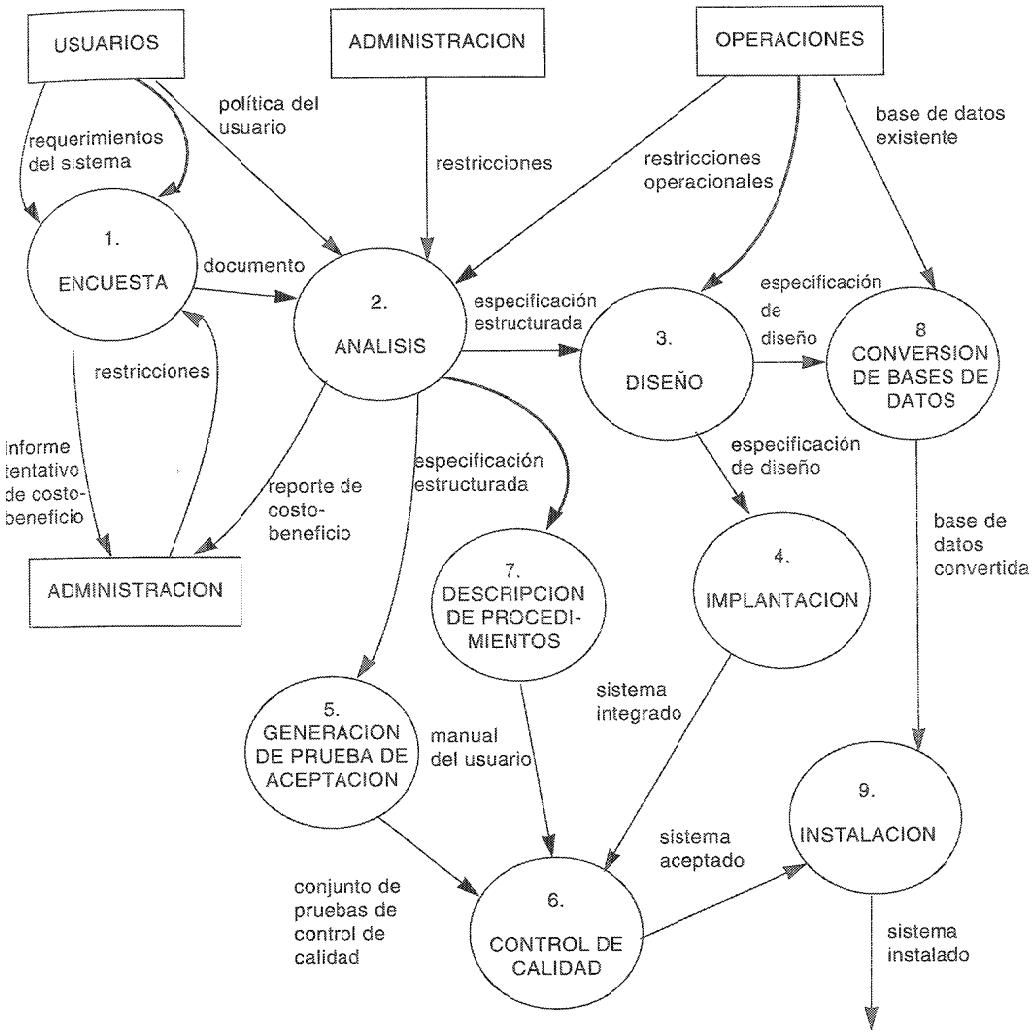


Figura 5.4: **El ciclo de vida del proyecto estructurado**

  - El hardware del sistema actual no es confiable y el vendedor se acaba de declarar en quiebra.
  - El software del sistema actual no se puede mantener, y no podemos ya contratar programadores de mantenimiento dispuestos a darle mantenimiento en el lenguaje que originalmente se utilizó para desarrollarlo.
  - El tiempo de respuesta del sistema telefónico de pedidos actual es tan malo que muchos clientes cuelgan frustrados antes de hacer su pedido.
  - El sistema actual no es capaz de producir los informes requeridos por la modificación a los impuestos decretada el año anterior.
  - El sistema actual no es capaz de recibir los informes sobre límites de crédito del departamento de contabilidad, y no puede producir los informes de promedio de volumen de pedidos que requiere el departamento de mercadotecnia.

- *Establecer metas y objetivos para un sistema nuevo.* Esto puede ser también una simple lista narrativa que contenga las funciones existentes que deben reimplantarse, las nuevas que necesitan añadirse y los criterios de desempeño del nuevo sistema.
- *Determinar si es factible automatizar el sistema y de ser así, sugerir escenarios aceptables.* Esto implicará algunas estimaciones bastante rudimentarias y aproximadas del costo y el tiempo necesarios para construir un sistema nuevo y los beneficios que se derivarán de ello;[10] también involucrará dos o más escenarios (por ejemplo, el escenario con una computadora grande, el de procesamiento distribuido, etc.). Aunque a estas alturas la administración y los usuarios usualmente querrán una estimación precisa y detallada, el analista tendrá mucha suerte si logra determinar el tiempo, los recursos y los costos con un error menor del 50% en esta etapa tan temprana del proyecto.
- *Preparar el esquema que se usará para guiar el resto del proyecto.* Este esquema incluirá toda la información que se lista anteriormente, además de identificar al administrador responsable del proyecto. También pudiera describir los detalles del ciclo de vida que seguirá el resto del proyecto.

En general, la encuesta ocupa sólo del 5 al 10 por ciento del tiempo y los recursos de todo el proyecto, y para los proyectos pequeños y sencillos pudiera ni siquiera ser una actividad formal. Sin embargo, aun cuando no consuma mucho del

10 Los cálculos de costo-beneficio se discutirán en el apéndice C.

tiempo y de los recursos del proyecto, es una actividad verdaderamente importante: al final de la encuesta, la administración pudiera decidir cancelar el proyecto si no parece atractivo desde el punto de vista de costo-beneficio.

Como analista, usted podrá o no estar involucrado en la encuesta; pudiera ser que antes de que siquiera se haya enterado del proyecto, el usuario y los niveles apropiados de la administración ya la hayan hecho. Sin embargo, en proyectos grandes y complejos, la encuesta requiere trabajo tan detallado que a menudo el usuario solicitará la colaboración del analista lo más pronto posible.

No discutiremos la encuesta con mayor detalle en este libro. Si llega a tener que ver con esta actividad, encontrará de utilidad los apéndices E y C. Para detalles adicionales, consulte [Dickinson, 1981], [Gore y Stubbe, 1983] y [Yourdon, 1988].

5.4.2 Actividad 2: El análisis de sistemas

El propósito principal de la actividad de análisis es transformar sus dos entradas –o insumos o factores– principales, las políticas del usuario y el esquema del proyecto, en una especificación estructurada. Esto implica modelar el ambiente del usuario con diagramas de flujo de datos, diagramas de entidad- relación, diagramas de transición de estado y demás herramientas que se presentaron en el capítulo 4. Estas herramientas se tratan con detalle en la parte II.

El *proceso* paso a paso del análisis de sistemas (es decir, las subactividades de la actividad de análisis de la figura 5.4) se discute en la parte III. Como veremos, implica el desarrollo de un *modelo ambiental* (que se trata en el capítulo 18) y el desarrollo de un *modelo de comportamiento* (que se discute en los capítulos 19 y 20). Estos dos modelos se combinan para formar el *modelo esencial* (que se explica en el capítulo 17), que representa una descripción formal de lo que el nuevo sistema debe hacer, independientemente de la naturaleza de la tecnología que se use para cubrir los requerimientos.

Además del modelo del sistema que describe los requerimientos del usuario, generalmente se prepara un conjunto de presupuestos y cálculos de costos y beneficios más precisos y detallados al final de la actividad de análisis. Esto se discute con más detalle en el apéndice C.

Obviamente, como analista del sistema, en esto pasará la mayor parte de su tiempo. No hay nada más que se necesite decir acerca de la actividad de análisis en este momento, ya que ese es el tema que trata todo el resto del libro.

5.4.3 Actividad 3: el diseño

La actividad de diseño se dedica a asignar porciones de la especificación (también conocida como modelo esencial) a procesadores adecuados (sean máquinas o humanos) y a labores apropiadas (o tareas, particiones, etc.) dentro de cada procesador. Dentro de cada labor, la actividad de diseño se dedica a la creación de

una jerarquía apropiada de módulos de programas y de interfases entre ellos para implantar la especificación creada en la actividad 2. Además, la actividad de diseño se ocupa de la transformación de modelos de datos de entidad-relación en un diseño de base de datos; véase [Inmon, 1988] para más detalles.

Parte de esta actividad le interesará como analista: el desarrollo de algo conocido como el *modelo de implantación del usuario*. Este modelo describe los asuntos relacionados con la implantación que le importan al usuario al grado de que no se los quiere confiar a los diseñadores y programadores. Los asuntos principales que suelen preocupar al usuario son aquellos relacionados con la especificación de la frontera humano-máquina y la especificación de la interfaz hombre-máquina. Esa frontera separa las partes del modelo esencial que llevará a cabo una persona (como actividad manual), de las partes que se implantarán en una o más computadoras. De manera similar, la interfaz hombre-máquina es una descripción del formato y de la secuencia de entradas que los usuarios proporcionan a la computadora (por ejemplo, el diseño de pantallas y el diálogo en línea entre el usuario y la computadora), además del formato y la secuencia de salidas –o productos– que la computadora proporciona al usuario. El modelo de implantación del usuario se describe en el capítulo 21.

En el capítulo 22 se puede encontrar una introducción al proceso de diseño de sistemas. Se puede encontrar material adicional en [Yourdon y Constantine, 1989], [Page-Jones, 1988], [Jackson, 1975], y otros.

5.4.4 Actividad 4: implantación

Esta actividad incluye la codificación y la integración de módulos en un esqueleto progresivamente más completo del sistema final. Por eso, la actividad 4 incluye tanto programación estructurada como implantación descendente.

Como podrá imaginar, el analista típicamente no se verá involucrado en esta actividad, aunque hay algunos proyectos (y organizaciones) donde el análisis, el diseño y la implantación de sistemas los hace la misma persona. Este tema se discute más a fondo en el capítulo 23.

5.4.5 Actividad 5: generación de pruebas de aceptación

La especificación estructurada debe contener toda la información necesaria para definir un sistema que sea aceptable desde el punto de vista del usuario. Por eso, una vez generada la especificación, puede comenzar la actividad de producir un conjunto de casos de prueba de aceptación desde la especificación estructurada.

Dado que el desarrollo de las pruebas de aceptación puede suceder al mismo tiempo que las actividades de diseño e implantación, pudiera ser que al analista le sea asignada esta labor al término del desarrollo del modelo esencial en la actividad 2. En el capítulo 23 se discute con más detalle el proceso de prueba.

5.4.6 Actividad 6: garantía de calidad

La garantía de calidad también se conoce como la prueba final o la prueba de aceptación. Esta actividad requiere como entradas los datos de la prueba de aceptación generada en la actividad 5 y el sistema integrado producido en la actividad 4.

El analista pudiera estar involucrado con la actividad de garantía de calidad, pero por lo regular no lo está. Pueden tomar la responsabilidad uno o más miembros de la organización usuaria, o pudiera llevarla a cabo un grupo independiente de prueba o un departamento de control de calidad. Consecuentemente, no se discutirá con más detalle la función de garantía de calidad.

Nótese, por cierto, que algunas personas le llaman a esta actividad "control de calidad" en lugar de "garantía de calidad". Sin importar la terminología, se necesita una actividad que *verifique* que el sistema tenga un nivel apropiado de calidad; le hemos llamado garantía de calidad en este libro. Nótese también que es importante llevar a cabo actividades de garantía de calidad *en cada una* de las actividades anteriores para asegurar que se hayan realizado con un nivel apropiado de calidad. Por eso, se esperaría que esto se haga durante toda la actividad de análisis, diseño y programación para asegurar que el analista esté desarrollando especificaciones de alta calidad, que el diseñador esté produciendo diseños de alta calicad y que el programador este escribiendo códigos de alta calidad. La actividad de garantía de calidad que se menciona aquí es simplemente la prueba *final* de la calidad del sistema.

5.4.7 Actividad 7: descripción del procedimiento

A lo largo de todo este libro nos preocupamos por el desarrollo de un sistema completo: no sólo de la porción automatizada, sino también de la parte que llevarán a cabo las personas. Por ello, una de las actividades importantes a realizar es la generación de una descripción formal de las partes del sistema que se harán en forma manual, lo mismo que la descripción de cómo interactuarán los usuarios con la parte automatizada del nuevo sistema. El resultado de la actividad 7 es un manual para el usuario.

Como podrá imaginar, esta también es una actividad en la que pudiera verse involucrado como analista. Aunque no se discutirá más a fondo en este libro, podría consultar libros acerca de redacción técnica para obtener mayor información sobre la escritura de manuales para el usuario.

5.4.8 Actividad 8: conversión de bases de datos

En algunos proyectos, la conversión de bases de datos involucraba más trabajo (y más planeación estratégica) que el desarrollo de programas de computadora para el nuevo sistema. En otros casos, pudiera no haber existido una base de datos que convertir. En el caso general, esta actividad requiere como entrada la base de datos actual del usuario, al igual que la especificación del diseño producida por medio de la actividad 3.

Según sea de la naturaleza del proyecto, el analista podría tener que ver con la actividad de conversión de la base de datos. Sin embargo no discutiremos esta actividad con mayor detalle en este libro.

5.4.9 Actividad 9: instalación

La actividad final, desde luego, es la instalación; sus entradas son el manual del usuario producido en la actividad 7, la base de datos convertida que se creó con actividad 8 y el sistema aceptado producido por la actividad 6. En algunos casos, sin embargo, la instalación pudiera significar simplemente un cambio de la noche a la mañana al nuevo sistema, sin mayor alboroto; en otros casos, la instalación pudiera ser un proceso gradual, en el que un grupo tras otro de usuarios van recibiendo manuales y entrenamiento y comenzando a usar el nuevo sistema.

5.4.10 Resumen del ciclo de vida del proyecto estructurado

Es importante ver la figura 5.4 como lo que es: un *diagrama de flujo de datos. No* es un diagrama de flujo; nada implica que toda la actividad N debe concluir antes de comenzar la actividad N + 1. Por el contrario, la red de flujos de datos que conectan las actividades hace ver con claridad que pudieran estarse llevando a cabo diversas actividades paralelamente. Debido a este aspecto no secuencial, usamos la palabra *actividad* en el ciclo de vida del proyecto estructurado en lugar de *"fase"*, que es más convencional. El término fase tradicionalmente se refiere a un periodo particular en un proyecto en el cual se estaba desarrollando una, y sólo una, actividad.

Hay otra cosa que debe recalcarse acerca del uso de un diagrama de flujo de datos para describir el ciclo de vida del proyecto: un diagrama de flujo de datos clásico, como el que se muestra en la figura 5.4, no muestra en forma explícita la retroalimentación, ni el control.[11] Prácticamente todas las actividades de la figura 5.4 pueden y suelen producir información que puede llevar a modificaciones adecuadas de una o más de las actividades precedentes. De aquí que la actividad de diseño puede producir información que acaso cambie algunas de las decisiones de costo-beneficio en la actividad de análisis; de hecho, el conocimiento que se obtiene a partir de la actividad de diseño pudiera incluso llevar a revisar decisiones anteriores acerca de la factibilidad básica del proyecto.

Más aún, en casos extremos, ciertos eventos que pudieran darse en cualquier actividad pueden causar que todo el proyecto termine repentinamente. Las entradas de la administración se muestran sólo para la actividad de análisis pues ésta es la única que requiere datos de la administración; sin embargo, se supone, que la administración ejerce *control* sobre todas las actividades.

11 En realidad, hay maneras de mostrar la retroalimentación y el control en los diagramas de flujo de datos, como se verá en el capítulo 9. Las notaciones adicionales (para flujos de control y de procesos de control) normalmente se utilizan para modelar sistemas de tiempo real y hemos evitado su uso en este modelo del "sistema para construir sistemas".

En resumen, la fFigura 5.4 sólo señala la o las entradas requeridas por cada actividad, y la o las salidas o productos que se generan. La secuencia de las actividades sólo puede suponerse en la medida en que la presencia o ausencia de datos haga posible comenzar una determinada actividad.

5.5 IMPLANTACION RADICAL CONTRA IMPLANTACION DESCENDENTE CONSERVADORA

En la sección anterior señalé que el ciclo de vida del proyecto estructurado permite que más de una actividad se lleve a cabo a la vez. Pongámoslo de otra manera: en la situación más extrema, *todas* las actividades del ciclo de vida estructurado pudieran estarse realizando simultáneamente. En el otro extremo, el administrador del proyecto pudiera decidir adoptar el enfoque secuencial, que implica terminar *completamente* una actividad antes de emprender la siguiente.

Es conveniente tener terminología para discutir estos extremos así como los términos medios entre ellos. El enfoque *radical* del ciclo de vida del proyecto estructurado es aquel en el que las actividades 1 a 9 se llevan a cabo paralelamente desde el principio del proyecto: la codificación se inicia el primer día del proyecto, y la encuesta y el análisis continúan hasta el último. En cambio, en el enfoque *conservador* del ciclo de vida del proyecto estructurado, la actividad N completa se termina antes de comenzar con la actividad N + 1.

Obviamente, ningún administrador en sus cabales adoptaría cualquiera de estos dos extremos. La clave para reconocer esto consiste en que los extremos radical y conservador definidos anteriormente son los puntos extremos de una gama de opciones; esto se ilustra en la figura 5.5. Existe un infinito número de opciones entre los extremos radical y conservador. Un administrador de proyecto pudiera decidir terminar el 75% de la actividad de encuesta, seguido por la terminación del 75% del análisis del sistema, y luego del 75% del diseño para poder producir un esqueleto razonable de un sistema cuyos detalles pudieran posteriormente refinarse al pasar por segunda vez por el ciclo de vida entero del proyecto. O bien, el administrador pudiera decidir terminar todas las actividades de encuesta y de análisis, seguido por la terminación del 50% del diseño y el 50% de la implantación. Las posibilidades son interminables.

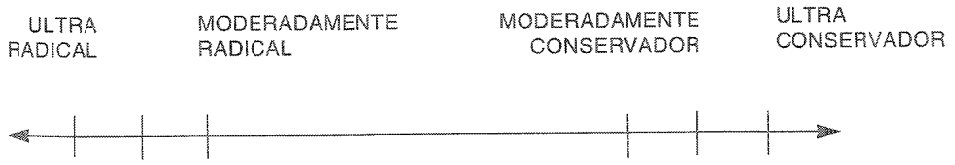


Figura 5.5: **Elecciones de implantación radical y conservadora**

¿Cómo decide un administrador de proyecto si adoptar un enfoque radical o conservador? Básicamente, no hay respuesta; la decisión suele basarse en los siguientes factores:

- ¿Qué tan voluble es el usuario?
- ¿Bajo qué presión labora el equipo del proyecto para producir resultados tangibles e inmediatos?
- ¿Bajo qué presión labora el administrador del proyecto para producir un presupuesto, programa, y estimación de personas y otros recursos?
- ¿Cuáles son los peligros de cometer un error técnico importante?

Como podrá apreciarse, ninguna de estas preguntas puede responderse claramente. Por ejemplo, uno no puede preguntarle al usuario, en una conversación informal, "¿Qué tan voluble andas hoy?". Por otro lado, el administrador del proyecto debiera poder juzgar la situación basándose en la observación, sobre todo si es un veterano que ha lidiado anteriormente con muchos usuarios y administradores de alto nivel.

Si el administrador del proyecto juzga que está tratando con un usuario voluble cuya personalidad es tal que retrasa la toma de decisiones hasta estar seguro de que el sistema va a funcionar, entonces probablemente optaría por un enfoque más radical. Lo mismo si trata con un usuario sin experiencia, a quien le hayan creado pocos sistemas. ¿Por qué pasar años desarrollando un conjunto perfecto de especificaciones tan sólo para descubrir que el usuario no comprendió su significado?

Por otro lado, si el administrador trata con un usuario veterano que está absolutamente seguro de lo que quiere, y si éste último trabaja en un área estable y con poca probabilidad de cambiar radicalmente de un mes a otro, entonces puede darse el lujo de adoptar un enfoque más conservador. Desde luego, hay muchas situaciones intermedias: el usuario puede estar seguro de *algunas* de las funciones de negocios que deberán llevarse a cabo, pero al mismo tiempo no estar seguro del tipo de informes administrativos que desea que el sistema le proporcione. O bien, si el usuario está familiarizado con sistemas computacionales por lotes *(batch)*, podría no estar seguro del impacto que pudiera tener en la empresa un sistema en línea.

Además de la volubilidad, existe un segundo factor que se debe considerar: la presión a la que se está sometido para producir resultados tangibles e inmediatos. Si, debido a las políticas u otras presiones externas, el equipo que realiza el proyecto *debe* concluirlo forzosamente para una fecha determinada, entonces se requiere un enfoque un tanto radical. El administrador del proyecto aún corre el riesgo de que el sistema sólo esté completo en un 90 por ciento para la fecha límite, pero por lo menos será un esqueleto operante completo en un 90 por ciento que puede mostrarse y tal vez incluso ponerse a producir. Eso generalmente es mejor que haber

terminado todo el análisis de sistemas, todo el diseño y toda la codificación, pero nada de las pruebas.

Desde luego, *todos* los proyectos llegan a verse apremiados a llegar a resultados tangibles; la cuestión es el del apremio. Es un asunto que puede ser algo dinámico: un proyecto que comienza holgadamente con un programa cómodo puede de repente volverse de alta prioridad y la fecha límite adelantarse seis meses o un año. Una de las ventajas de hacer el análisis, diseño, codificación e implantación del sistema en forma descendente es que se puede suspender una actividad en cualquier momento y dejar los detalles restantes para consideración posterior; mientras tanto, el análisis de alto nivel que se haya terminado puede usarse para comenzar el diseño de alto nivel, y así para los demás casos.

Otro factor más en la administración de proyectos es el requisito siempre presente en la mayoría de las organizaciones grandes de que se tienen que producir programas, estimaciones, presupuestos, etc. En algunas organizaciones, esto suele hacerse de manera bastante informal, normalmente porque los proyectos son relativamente pequeños y porque la administración siente que cualquier error en la estimación tendrá poco impacto en la organización global. En tales casos se puede adoptar un enfoque radical, aunque cualquier intento de hacer una estimación se tendrá que reducir al nivel de conjeturas viscerales. En cambio, la mayoría de los proyectos requieren estimaciones relativamente detalladas de necesidades de personal, recursos computacionales, etc., y esto sólo se puede realizar tras un sondeo, análisis y diseño bastante detallados. En otras palabras, entre más detalladas y precisas tengan que ser las estimaciones, más probable es que el proyecto siga un enfoque conservador.

Finalmente, el administrador del proyecto debe considerar el peligro de cometer un error técnico importante. Por ejemplo, suponga que toda su experiencia pasada en desarrollo de proyectos ha sido con una pequeña computadora de procesamiento por lotes IBM/36. Ahora, de repente, está a cargo de desarrollar un sistema de multiprocesamiento en línea para administración de bases de datos distribuidas, en tiempo real, que procesará 2 millones de transacciones diarias desde 5000 terminales distribuidas en todo el mundo. En tal situación, uno de los peligros del enfoque radical es descubrir algún error importante en el diseño tras haber realizado una buena parte del esqueleto de alto nivel del sistema.

Pudiera descubrir, por ejemplo, que para que su gran sistema funcione se requiere que un módulo de bajo nivel lleve a cabo su función en 19 microsegundos, pero sus programadores de repente le informan que es imposible codificar un módulo con tanta eficiencia, ni en COBOL, ni en C, ni siquiera (¡uf!) en lenguaje ensamblador. Por lo tanto, debe estar alerta al hecho de que seguir un enfoque radical requiere que sus analistas y diseñadores escojan un "tope máximo" para su sistema en etapa relativamente temprana, y que siempre existe el peligro de descubrir, ya cerca del final, que escogieron un máximo equivocado.

Sin embargo, considere otra situación: el administrador del proyecto ha decidido construir un sistema electrónico de proceso de datos con equipo nuevo, sistema operativo nuevo, sistema de administración de bases de datos nuevo (producido por alguien que no sea el vendedor), y un paquete de telecomunicaciones nuevo (producido por otra empresa más). Todos los proveedores tienen manuales brillantes e impresionantes que describen sus productos, pero nunca han probado la interfaz entre sus respectivos productos de hardware y software. ¿Quién sabe si siquiera funcionarán juntos? ¿Quién sabe si las funciones prometidas por un proveedor queden anuladas por los recursos del sistema que utiliza el otro? Ciertamente, en un caso como éste, el administrador del proyecto pudiera elegir un enfoque radical, para que la versión esqueleto o primaria del sistema pueda utilizarse para explorar los posibles problemas de interacción e interfaz entre los componentes de los diferentes proveedores.

Si el administrador del proyecto está a cargo de un tipo familiar de sistema como, por ejemplo, su nonagésimo noveno sistema de nóminas, probablemente tenga bastante idea de qué tan realistas sean sus metas. Es posible que recuerde, de su proyecto anterior, qué tipo de módulos necesitará a nivel detallado, y probablemente recuerde con claridad cómo se veía la estructura de alto nivel del sistema. En tal caso, pudiera estar dispuesto a correr el riesgo de cometer un error dados los demás beneficios que puede traerle un enfoque racical.

En resumen, el enfoque radical es el más adecuado para intentos apenas disfrazados de investigación y desarrollo, en los que nadie está muy seguro de qué es lo que se supone que debe hacer el sistema final. Y es bueno para los casos en los que para determinada fecha algo *tiene que* estar ya funcionando, y en situaciones en las que la percepción del usuario respecto a lo que desea que el sistema haga esté sujeta a posibles cambios. El enfoque conservador, por otro lado, suele usarse en proyectos más grandes, en los que se invierten cantidades enormes de dinero y para los cuales se requiere un análisis y diseño muy detallados para evitar desastres subsecuentes. Sin embargo, cada proyecto es diferente y requiere de su propia combinación de implantación descendente conservadora y radical. Para tratar la naturaleza individual de cada proyecto, el administrador debe estar dispuesto a modificar su enfoque en mitad del camino si es necesario.

5.6 EL CICLO DE VIDA DE PROTOTIPOS

Se ha vuelto popular en los últimos años una variación del enfoque descendente antes discutido. En general se le conoce como el enfoque de *prototipos* y lo popularizaron Bernard Boar, James Martin y otros. Como lo describe Boar [Boar, 1984]:

> Una alternativa de enfoque para la definición de los requerimientos consiste en capturar un conjunto inicial de necesidades e implantarlas rápidamente con la intención declarada de expandirlas y refinarlas iterativamente al ir aumentando la comprensión que del sistema tienen el usuario y quien lo desarrolla. La definición del sistema se realiza mediante el descubrimiento evolutivo y gradual y no a través de la previsión omnisciente... Este tipo de enfoque se llama "de prototipos". También se le conoce como modelado del sistema o desarrollo heurístico. Ofrece una alternativa atractiva y practicable a los métodos de especificación para tratar mejor la incertidumbre, la ambigüedad y la volubilidad de los proyectos reales.

Por muchas razones, esto suena exactamente como el enfoque descendente radical que se discutió en la sección anterior. La principal diferencia es que el enfoque estructurado que se discute a lo largo de este libro supone que tarde o temprano se construirá *un modelo en papel completo del sistema* (es decir, un juego completo de diagramas de flujo de datos, de diagramas entidad-relación, de diagramas de transición de estados, de especificaciones de procesos, etc.). El modelo se completará más pronto con un enfoque conservador y más tarde con uno radical, pero para el final del proyecto habrá un juego formal de documentos que deberán permanecer siempre con el sistema, a lo largo de su corrección y mantenimiento.

El enfoque de prototipos, por otro lado, casi siempre supone que el modelo será operante, es decir, una colección de programas de computadora que simularán algunas o todas las funciones que el usuario desea. Pero dado que se pretende que dichos programas sean sólo de modelo, también se supone que *al concluirse el modelado, los programas se descartarán y se reemplazarán con programas REALES.* Quienes hacen prototipos generalmente usan los siguientes tipos de herramientas de software:

- Un diccionario de datos integrado
- Un generador de pantallas
- Un generador de reportes no guiado por procedimientos
- Un lenguaje de programación de cuarta generación
- Un lenguaje de consultas no guiado por procedimientos
- Medios poderosos de administración de bases de datos

El ciclo de vida de prototipos propuesto por Boar se muestra en la figura 5.6. Comienza con una actividad de sondeo, similar a la que propone este libro. De esto sigue inmediatamente una determinación de si el proyecto es un buen candidato para un enfoque de prototipos. Los buenos candidatos son aquellos que tienen las siguientes características:

- El usuario no puede o no está dispuesto a examinar modelos abstractos en papel, tales como diagramas de flujo de datos.
- El usuario no puede o no está dispuesto a articular (o "pre-especificar") sus requerimientos de ninguna forma y sólo se pueden determinar sus requerimientos mediante un proceso de tanteo, o ensayo y error. O, como

lo dice mi colega Bob Spurgeon, es la situación en la que el usuario dice: "No sé qué es lo que quiero, pero lo reconoceré cuando lo vea".

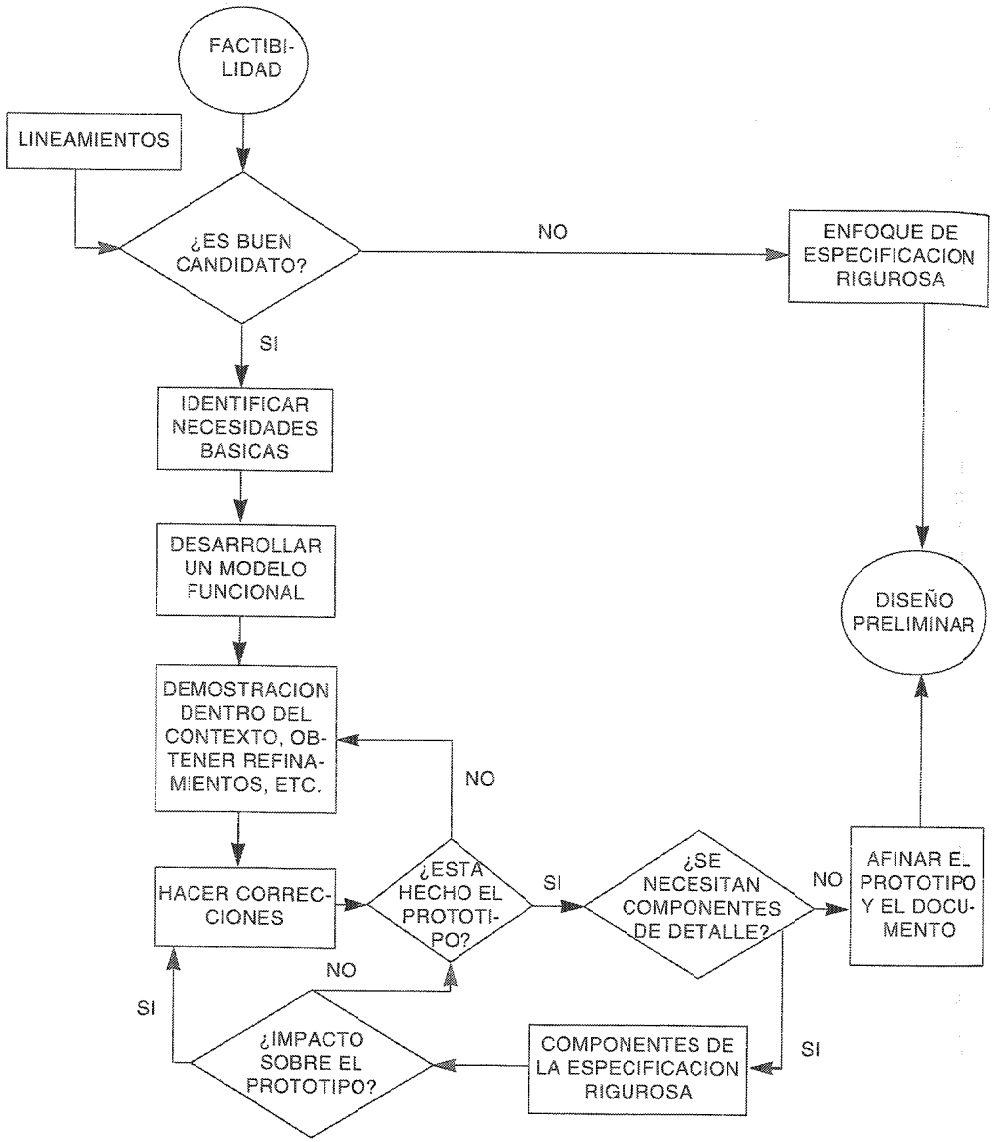


Figura 5.6: **El ciclo de vida por prototipos**

- Se tiene la intención de que el sistema sea en línea y con operación total por la pantalla, en contraposición con los sistemas de edición, actualización y reportes operados por lotes. (Casi todas las herramientas de software de prototipos apuntan al enfoque orientado a terminales en línea y manejadas por bases de datos; existen pocas herramientas de software en el mercado para ayudar a la creación de prototipos de sistemas de procesamiento por lotes.)

- El sistema *no* requiere la especificación de grandes cantidades de detalles algorítmicos, ni de muchas especificaciones de procesos para describir los algoritmos con los cuales se obtienen los resultados. Por ello, los sistemas de apoyo a decisiones, de recuperación ad hoc (a propósito) y de administración de registros son buenos candidatos para el prototipo. Los buenos candidatos suelen ser sistemas en los cuales el usuario se preocupa más por el formato y distribución de los datos de entrada y salida en la pantalla, y por los mensajes de error, que por los cómputos que realiza el sistema para lograrlo.

Es importante notar que el ciclo de vida de prototipos que se muestra en la figura 5.6 concluye con una fase de diseño de un ciclo de vida estructurado "tradicional" como los que describe este libro. Específicamente, esto significa que *no se tiene la intención* de que el prototipo haga las veces de un sistema operacional; la intención es tan sólo que modele los requerimientos del usuario.

El enfoque de prototipos ciertamente tiene su mérito en muchas situaciones. En algunos casos, el administrador del proyecto tal vez quiera utilizar este enfoque como alternativa al de análisis estructurado que se presenta en este libro; en otros casos, pudiera desear utilizarlo *en conjunto* con la creación de modelos en papel, como los diagramas de flujo de datos. Tenga en mente lo siguiente:

- El enfoque descendente descrito en la sección anterior es otra manera de hacer un prototipo, pero en vez de usar herramientas que se pueden obtener en el mercado, como generadores de pantallas y lenguajes de cuarta generación, el equipo que realiza el proyecto utiliza el sistema mismo como su propio prototipo. Es decir, las diversas versiones de un esqueleto del sistema proveen un modelo operativo con el cual el usuario puede interactuar y darse así una idea más realista de las funciones del sistema que la que se pudiera formar a partir de un modelo en papel.

- El ciclo de vida de prototipos, como se describió anteriormente, involucra el desarrollo de un modelo funcional, que luego se descarta y se reemplaza con un sistema de producción. Existe un peligro considerable de que el usuario o el equipo que desarrolla el sistema traten de convertir al prototipo mismo en un sistema de producción. Esto suele resultar un desastre, pues el prototipo no puede trabajar eficientemente con grandes volúmenes de transacciones, y porque carece de detalles operacionales

tales como recuperación de errores, auditorías, características de respaldo/reinicio, documentación para el usuario y procedimientos de conversión.

- Si de hecho se descarta el prototipo y se reemplaza con el sistema de producción, existe el peligro real de que pudiera concluirse el proyecto sin dejar un registro permanente de los requerimientos del usuario. Esto probablemente dificulte cada vez más el mantenimiento con el paso del tiempo (por ejemplo, diez años después de la construcción del sistema, será difícil que los programadores de mantenimiento incorporen algún cambio, pues nadie, incluyendo a los usuarios de "segunda generación" que están trabajando actualmente con el sistema, recordará lo que se suponía en primer lugar que debía hacer). El ciclo de vida que se presenta en este libro se basa en la idea de que los modelos en papel desarrollados durante la actividad de análisis no sólo serán una entrada para la actividad de diseño, sino que también se conservarán (y se modificarán según vaya siendo necesario) durante el mantenimiento. De hecho, los modelos pudieran sobrevivir más allá del sistema en el cual se implantaron, y pudieran servir como especificación para el sistema de reemplazo.

5.7 RESUMEN

El principal propósito de este capítulo fue proporcionar una visión global de los ciclos de vida de los proyectos en general. Si examina el ciclo de vida formal de proyectos en cualquier organización de desarrollo de sistemas, debería poder distinguir si se trata de uno clásico, semiestructurado, estructurado, o de prototipos.

Si su proyecto sólo permite una actividad a la vez, la discusión sobre implantación descendente radical y conservadora de la Sección 5.6 puede haberlo perturbado. Este fue mi propósito, y el principal objetivo de esa sección fuéehacerle pensar acerca de la *posibilidad* de traslapar algunas de las principales actividades en el proyecto de desarrollo de un sistema. Obviamente, es más difícil administrar un proyecto en el cual diversas actividades se llevan a cabo en paralelo, pero, hasta cierto punto, eso sucede en todo proyecto. Aún si el administrador decide que su gente se concentrará en una actividad a la vez, de todos modos habrá varias subactividades que se llevarán a cabo en paralelo. Múltiples analistas de sistemas estarán entrevistando simultáneamente a múltiples usuarios; diversas piezas del producto final del análisis se encontrarán en diversas etapas de progreso a lo largo de toda la fase de análisis. Una labor del administrador es tener el suficiente control sobre dichas subactividades como para asegurar que se coordinen propiamente. Y en casi cualquier proyecto de proceso electrónico de datos, este tipo de actividad paralela se da también a alto nivel; es decir, a pesar de lo que pueda haber recomendado el ciclo de vida formal del proyecto de una organización dada, la realidad es que muchas de las principales actividades del proyecto sí se traslapan hasta cierto punto. No obstante, si el administrador decide insistir en una progresión de actividades estrictamente secuencial, aún funcionará el ciclo de vida presentado por este libro.

Para obtener mayores detalles acerca de ciclos de vida de proyectos, consulte [Dickinson, 1981] y [Yourdon, 1988]. También cubren este concepto una variedad de libros de ingeniería de software y de libros de administración de proyectos.

REFERENCIAS

1. Edward Yourdon y Larry L. Constantine, *Structured Design: Fundamentals and Applications in Software Engineering,* 2a. edición, Englewood Cliffs, N.J.: YOURDON Press, 1988.
2. Meilir Page-Jones, *The Practical Guide to Structured Systems Design,* 2ª edición Englewood Cliffs, N.J.:YOURDON Press, 1988.
3. Bernard Boar, *Application Prototyping.* Reading, Mass.: Addison-Wesley, 1984.
4. James Grier Miller, *Living Systems.* Nueva York: McGraw- Hill, 1978.
5. Brian Dickingson, *Developing Structured Systems.* Nueva York: YOURDON Press, 1981.
6. Edward Yourdon, *Managing the Systems Life Cycle,* 2ª edición, Englewood Cliffs, N.J.: Prentice-Hall, 1988.
7. James Grier Miller, *Living Systems.* Nueva York: McGraw- Hill, 1978.
8. Michael Jackson, *Principles of Program Design.* Nueva York: Academic Press, 1975.
9. Winston W. Royce, "Managing the Development of Large Software Systems", *Proceedings, IEEE Wescon,* agosto 1970, pp. 1-9.
10. Barry Boehm, *Software Engineering Economics.* Englewood Cliffs, N.J.: Prentice-Hall, 1981.
11. Bill Inmon, *Information Engineering for the Practitioner: Putting Theory into Practice.* Englewood Cliffs, N.J.: Prentice-Hall, 1988.
12. Marvin Gore y John Stubbe, *Elements of Systems Analysis,* 3ª edición, Dubuque, Iowa: William Brown, 1983.

PREGUNTAS Y EJERCICIOS

1. Mencione dos sinónimos de metodología.
2. ¿Cuál es la diferencia entre una herramienta, como se utiliza en este libro, y una metodología?
3. ¿Cuáles son los tres principales propósitos del ciclo de vida de un proyecto?

4. Proyecto de investigación: Encuentre el precio de tres productos para metodología comerciales ofrecidos por proveedores de software o empresas consultoras.

5. ¿Por qué normalmente las organizaciones pequeñas de proceso de datos no usan metodologías formales?

6. ¿Por qué es útil para los administradores nuevos una metodología?

7. ¿Por qué es importante tener una metodología en una organización en la que se estén llevando a cabo muchos proyectos diferentes?

8. ¿Cómo es que una metodología es útil para controlar proyectos?

9. ¿Cuáles son las principales características que distinguen el ciclo de vida clásico?

10. ¿Qué significa la expresión implantación ascendente?

11. ¿Cuáles son las cuatro principales dificultades con la estrategia de implantación ascendente?

12. ¿Qué tipo de ambiente es el adecuado para un enfoque de implantación ascendente?

13. ¿Por qué es importante que "nada está hecho hasta que todo esté hecho", que además es lo que caracteriza al enfoque ascendente?

14. ¿Por qué debieran encontrarse los errores triviales primeramente en la fase de prueba de un proyecto?

15. ¿Qué diferencia hay entre prueba y eliminación de errores?

16. ¿Por qué es difícil la eliminación de errores en una implantación ascendente?

17. ¿Qué se entiende por la frase "progresión secuencial" cuando se describe el ciclo de vida de un proyecto?

18. ¿Cuáles son los dos principales problemas de la progresión secuencial?

19. ¿Cuáles son las principales diferencias entre el ciclo de vida semiestructurado y el clásico?

20. ¿Cuáles son las dos principales consecuencias del enfoque de la implantación descendente?

21. ¿Por qué, en el ciclo de vida semiestructurado, el diseño a menudo involucra trabajo redundante?

22. ¿Cuáles son las principales diferencias entre los ciclos de vida semiestructurado y estructurado?

23. Nombre las nueve actividades del ciclo de vida estructurado del proyecto.

24. ¿Quiénes son los tres tipos de personas que proveen de entradas primarias al ciclo de vida del proyecto?

25. ¿Cuáles son los cinco principales objetivos de la actividad de la encuesta?

26. ¿Qué es un diagrama de contexto?

27. ¿Cuál es el principal propósito de la actividad de análisis?

28. ¿Cuáles son los tipos de modelos producidos por la actividad de análisis?

29. ¿Cuál es el propósito de la actividad de diseño?

30. ¿Cuáles son los dos asuntos principales que normalmente le preocupan al usuario en la actividad de diseño?

31. ¿Cuándo puede comenzar la generación de pruebas de aceptación? (actividad 5)

32. ¿Cuál es el propósito de la actividad de descripción del procedimiento?

33. ¿Por qué se utilizó un diagrama de flujo de datos en la Figura 5.4 para mostrar el ciclo de vida del proyecto?

34. ¿Cuál sería un posible sinónimo para la palabra actividad?

35. ¿Por qué es importante la retroalimentación en el ciclo de vida estructurado del proyecto?

36. ¿Qué diferencia hay entre los enfoques radical y conservador para el ciclo de vida estructurado del proyecto?

37. ¿Cuáles son los cuatro principales criterios para elegir el enfoque radical vs. el enfoque conservador?

38. ¿Se le ocurre algún criterio adicional para elegir un enfoque radical vs. un enfoque conservador?

39. ¿Qué tipo de enfoque (radical vs. conservador) debe escoger el administrador de un proyecto si es probable que el usuario cambie de opinión respecto a los requerimientos del sistema?

40. ¿Qué tipo de enfoque (radical vs. conservador) debe escoger el administrador del proyecto si tiene una gran presión de tiempo?

41. ¿Qué tipo de enfoque (radical vs. conservador) debe escoger el administrador del proyecto si se encuentra con riesgos técnicos importantes?

42. ¿Qué diferencia existe entre el ciclo de vida de prototipos y el radical?

43. ¿Qué características tiene el proyecto de prototipos ideal?

44. ¿Qué clase de herramientas se requieren típicamente para un proyecto de prototipos?

45. ¿Por qué no son generalmente buenos candidatos para proyectos de prototipo los sistemas por lotes?

46. ¿Cuáles son los peligros del enfoque de prototipos?

47. ¿Cómo pueden utilizarse en combinación en un proyecto los ciclos de vida estructurado y de prototipos?

6 ASPECTOS IMPORTANTES EN EL DESARROLLO DE SISTEMAS

Los dogmas del tranquilo pasado son inadecuados para el borrascoso presente. La ocasión está atiborrada de dificultades y debemos estar a la altura. Como nuestro caso es nuevo, debemos pensar y actuar en forma novedosa. Debemos desenredarnos, y entonces salvaremos a nuestra nación.

Abraham Lincoln,
Segundo Mensaje Anual al Congreso

En este capítulo se aprenderá:

1. Por qué la productividad es un asunto importante
2. Las soluciones comunes al problema de la productividad
3. El número de errores en un sistema típico
4. La relación entre la edad de un sistema y el número de errores encontrados

Como analista de sistemas, formará parte de un equipo de personas cuyo propósito es desarrollar un sistema de información útil y de alta calidad, que cubrirá las necesidades del usuario final. Al llevar a cabo su trabajo, usted y sus compañeros miembros del equipo sin duda se verán influenciados por las siguientes importantes cuestiones:

- Productividad
- Confiabilidad
- Mantenibilidad

Desde luego, todo mundo está a favor de la productividad; es un término utilizado de igual forma que la maternidad y la lealtad a la patria. Pero hace una generación, cuando se estaban creando la mayoría de los sistemas operativos, la productividad no era algo a lo que se hiciera mucho caso. Los analistas y programadores de los años 60 trabajaban largas horas (aunque no siempre en horarios predecibles), pero nadie estaba seguro jamás de cuánto trabajo lograrían hacer en una semana, o cuánto les tomaría construir un sistema completo. El sentir común era que el equipo de desarrollo del proyecto trabajaría Muy Duro cada día, y un día, ¡voila!, ¡magia! el sistema quedaría terminado.

Hoy en día, la productividad es un asunto mucho más serio. Asimismo lo es el asunto de la confiabilidad: una falla del sistema en un sistema grande y complejo probablemente tendría consecuencias devastadoras. Y la mantenibilidad se ha convertido en un asunto de importancia también: ahora es claro que muchos de los sistemas construidos hoy deberán durar por lo menos 20 años o más antes de que puedan volverse a construir, y tendrán que someterse a constantes revisiones y modificaciones durante su existencia.

Cada uno de estos asuntos se explora con más detalle en este capítulo. Algunos, como el asunto del mantenimiento, aparentan tener poco o nada que ver con el análisis de sistemas, pero, como se verá, el analista juega un papel crucial en el logro de la productividad mejorada, una mayor confiabilidad y mejor mantenibilidad.

6.1 PRODUCTIVIDAD: EL RETRASO EN LAS APLICACIONES

Tal vez el problema más visible al que se enfrenta actualmente la profesión de desarrollo de sistemas sea el de la productividad. La sociedad y los negocios modernos parecen exigir cada vez más: más sistemas, más complejidad y todo más rápido. Como analista, verá los dos principales aspectos de este problema: el retraso en los nuevos sistemas que se necesita desarrollar, y el tiempo que se requiere para construir un sistema individual nuevo.

En casi cualquier organización de los Estados Unidos donde exista un grupo centralizado responsable del desarrollo de nuevos sistemas, existe un retraso de varios años de trabajo en espera de que se le lleve a cabo;[1] de hecho, muchas organizaciones tienen un retraso de cuatro a siete años o más. El retraso se presenta en tres tipos diferentes:

1 Las conversaciones informales que he sostenido con administradores de proceso de datos en Canadá, Europa, Australia, Sudamérica y otras partes del mundo me llevan a pensar que este problema no es único de los Estados Unidos.

- Retraso *visible*. Sistemas nuevos que los usuarios han pedido oficialmente y que han sido aprobados y financiados por comités apropiados de administración. Sin embargo, los proyectos no se han iniciado porque no existen los recursos adecuados (esto es, analistas, programadores, etc.).
- Retraso *invisible*. Existen sistemas nuevos que los usuarios saben que necesitan, pero no se han molestado en pedirlos "oficialmente", porque aún están aguardando a que se concluyan proyectos del retraso visible.
- Retraso *desconocido*. Estos son sistemas nuevos que los usuarios ni siquiera saben que quieren todavía, pero que serán identificados en cuanto se termine alguno de los sistemas del retraso visible o del invisible.

En un estudio clásico del retraso y de la demanda de sistemas de información [Alloway y Quillard, 1982], los investigadores Robert Alloway y Judith Quillard, de la escuela Sloan del Instituto de Tecnología de Massachusetts, encontraron que el retraso invisible era típicamente 5.35 veces mayor que el retraso visible de los nuevos sistemas. Esto sugiere que el problema del retraso es más bien como el proverbial iceberg: sólo una pequeña porción es visible, con una gran parte oculta bajo el agua. Esto, desde luego, es un problema de primera importancia para la organización de desarrollo de sistemas que lleva a cabo su planeación y sus presupuestos basándose sólo en las exigencias conocidas y visibles de sus servicios.

Un segundo aspecto del problema de la productividad es el tiempo necesario para desarrollar un sistema individual.[2] Claro está que, si el proyecto de desarrollo de sistemas promedio pudiera reducirse de tres años a uno, el problema del retraso desaparecería rápidamente. Pero aquí el asunto es que los usuarios generalmente no se preocupan por el problema *global* del retraso, sino también por el problema *local* de productividad asociado con un proyecto individual. Se preocupan por si, para cuando el nuevo sistema esté listo, habrán cambiado las condiciones de negocios tan drásticamente que los requerimientos originales ya no sean relevantes.

O, poniéndolo en otras palabras, un nuevo sistema se asocia con una oportunidad de negocios que el usuario percibe, y esa oportunidad a menudo tiene un margen de oportunidad, es decir, un periodo tras el cual ésta habrá desaparecido y ya no se necesitará el sistema nuevo.

Existe un tercer motivo del problema de la productividad en muchas organizaciones: algunos proyectos resultan ser inútiles y la administración los cancela antes de que se terminen. De hecho, varias encuestas han mostrado que hasta un 25% de los proyectos en organizaciones grandes de sistemas de información jamás se concluyen. Existen, desde luego, muchas razones por las cuales puede fallar un pro-

2 Para dar una idea de hasta dónde abarca este problema, Caper Jones encontró, en una encuesta de aproximadamente 200 organizaciones grandes en EUA, que el proyecto típico se retrasaba un año y se excedía en un 100% del presupuesto. Véase [Jones, 1986].

yecto: problemas técnicos, problemas administrativos, personal sin experiencia, falta de tiempo para llevar a cabo un buen trabajo de análisis y diseño (lo cual usualmente es un problema de la administración), y falta de participación por parte de la administración o de los usuarios. Para una excelente exposición de la razón de las fallas de los proyectos, véase el agradable libro de Robert Block, *The Politics of Projects* [Block, 1980].

El problema de la productividad ha existido por muchos años en la profesión de desarrollo de sistemas, y muchas organizaciones están buscando de manera agresiva la forma de reducir radicalmente su retraso en las aplicaciones y disminuir el tiempo requerido para desarrollar un nuevo sistema. Entre las técnicas más comúnmente utilizadas están las siguientes:

- *Contratar más programadores y analistas.* Esto es muy común en las organizaciones nuevas y que crecen (por ejemplo, una organización creada como resultado de una fusión, o una organización nueva formada para explotar mercados nuevos y nuevos negocios).[3] Para organizaciones maduras, sin embargo, este enfoque usualmente se descarta; de hecho, muchas organizaciones sienten que tienen demasiados programadores y analistas ya y que en realidad lo que se ocupa es hacerlos más productivos. [4]

- *Contratar programadores y analistas más talentosos y darles mejores condiciones de trabajo.* En vez de armar un gran ejército de mediocres creadores de sistemas, algunas organizaciones se concentran en crear un grupo más pequeño de profesionales de gran talento, altamente capacitados y fuertemente apoyados (y se esperaría que bien pagados). Este enfoque se basa en la disparidad bien conocida en el rendimiento de programadores con experiencia: en un estudio clásico hecho en 1968 [Sackman, Erickson y Grant, 1968] se documentó por primera vez el hecho de que algunos programadores son 25 veces más productivos que otros. Una forma extrema de este concepto es el "superprogramador" o el "equipo con programador en jefe", que fue un concepto que popularizó IBM en los años 70 (véase [Aron, 1976], [Baker, 1972] y [Mills y Baker, 1973]), que consiste en un equipo de especialistas del proyecto (bibliotecarios, creadores de herramientas, etc.) que apoya a un profesional de talento extraordinario que llevaba a cabo tanto el análisis como el diseño del sistema. Desde luego, la mayoría de las organizaciones no puede construir toda una organización de desarrollo de sistemas en torno a una persona diez veces superior al promedio.[5] Sin embargo, tiene bastante de positivo el tratar de volver lo más productivo posible al personal existente, dándole cursos de actualización, herramientas modernas de desarrollo de software (que se tratan posteriormente con más detalle), y condiciones apropiadas de trabajo.[6]

- *Permitir a los usuarios desarrollar sus propios sistemas.* Es interesante notar que muchos otros adelantos tecnológicos interponían a alguien entre el usuario y el aparato mismo: el chofer del automóvil y la operadora de conmutador telefónico son dos ejemplos obvios. Desde luego, la mayoría de nosotros no tenemos choferes y la mayoría no necesitamos operadoras para realizar nuestras llamadas; el automóvil y el teléfono son lo suficientemente "amistosos" con el usuario como para que los podamos operar nosotros mismos. De igual manera, la combinación de computadoras personales, centros de información y lenguajes de programación de cuarta generación, todo lo cual fue introducido en muchas organizaciones estadunidenses a mediados de los años 80, hicieron posible para muchos usuarios (incluyendo, como se vio en el capítulo 2, a una generación de usuarios que aprendieron los fundamentos de la computación en la secundaria, preparatoria o facultad) el desarrollar sus propias aplicaciones sencillas. Los informes ad hoc, las bases de datos, las aplicaciones de hojas de cálculo y ciertos cambios de mantenimiento a los programas existentes son algunos de los proyectos que un usuario motivado y letrado en computación puede desarrollar por sí solo.

- *Mejores lenguajes de programación.* Los lenguajes de programación han sufrido enormes cambios desde los años 50, cuando los programadores creaban los programas codificando laboriosamente los ceros y unos binarios que el hardware ejecuta. Esta *primera generación del lenguaje de máquina* dio lugar a una *segunda generación* de lenguaje ensamblador en los años 60, y es interesante notar que el lenguaje *ensamblador* aún se utiliza hoy en día en algunos proyectos. Sin embargo, los lenguajes de

3 Un buen ejemplo de esto ocurrió a mediados de la década de los años 80, cuando varios países liberaron su banca y su bolsa. Australia, Japón e Inglaterra están entre los países que de repente encontraron que docenas de bancos y bolsas extranjeros abrían sus puertas. De éstos, Londres fue tal vez la localidad más visible, con su desreglamentación "Big Bang" de 27 de octubre de 1986. Estas actividades requirieron del desarrollo de nuevos sistemas grandes de información, lo cual generalmente se lograba empleando a tantos programadores y analistas nuevos como se pudiera, en el tiempo más corto posible.

4 Esto contrasta con las predicciones de una *escasez* nacional de programadores hechas por el Departamento de Comercio de los E.U., y la Fundación Nacional de Ciencia. Para mayores detalles véase el Capítulo 28 de [Yourdon, 1986].

5 Para una exposición detallada del por qué no es práctico el concepto de superprogramador en la mayoría de las organizaciones, véase *Managing the Structured Techniques* [Yourdon 1988].

6 Una interpretación obvia de las condiciones de trabajo apropiadas es dar a cada miembro del proyecto una oficina privada, o para dos personas, o un cubículo aislado de ruidos para permitir la privacidad y la concentración. Esto por sí solo es probable que mejore la productividad del analista/programador en un 10 por ciento o más, en comparación con el que trabaja en un área abierta y grande con música a todo volumen. Otra interpretación es esta: déjelos trabajar en casa. Para ahondar más sobre este concepto de la "cabaña electrónica", véase el capítulo 3 de [Yourdon, 1986].

procedimientos de la *tercera generación* empezaron a prevalecer en los años 70 y siguen siendo el tipo más común de lenguaje; como ejemplos tenemos COBOL, FORTRAN, BASIC, Pascal, C, MODULA-2 y Ada. No obstante que la industria de software continúa mejorando estos lenguajes (por ejemplo, la versión actual de FORTRAN es mucho mejor que la que usaban los programadores a comienzos de los años 70), el enfoque se ha dirigido a una nueva *cuarta generación* de lenguajes que eliminan la necesidad de preocuparse por engorrosos y morosos detalles de edición y valición de entradas, formato de los reportes y manejo de archivos. Ejemplos de esto son FOCUS, RAMIS, MAPPER y MARK V (al igual que lenguajes como PC-FOCUS, dBASE-III y Rbase-5000 para computadoras personales). Muchos arguyen que estos nuevos lenguajes pueden incrementar la productividad de la actividad de programación hasta en un factor de diez. Sin embargo, dado que la programación representa típicamente sólo del 10 al 15 por ciento del proyecto global de desarrollo del sistema, la ganancia global en productividad es a menudo mucho menos substancial.

- *Atacar el problema del mantenimiento.* El mantenimiento es un asunto de primera importancia en el campo del desarrollo de sistemas, como se discutirá en la sección 6.4. Sin embargo, la mayor parte de la atención está enfocada actualmente (como era de esperarse) a la mantenibilidad de los sistemas nuevos; mientras tanto, como se mencionó anteriormente, muchas organizaciones están dedicando un 50 a 70 por ciento de sus recursos al mantenimiento de sistemas viejos. Así, la interrogante es ¿qué puede hacerse para volver más fáciles de mantener estos sistemas viejos, aparte de la idea obvia de desecharlos y construir reemplazos nuevos? Un enfoque que crece en popularidad es el de *reestructuración*, es decir, la traducción mecánica de los programas anteriores (cuya lógica ha sido parchada y cambiada tantas veces que a menudo es completamente ininteligible) a programas nuevos, estructurados y bien organizados.[7] Una idea relacionada con esto es el uso de paquetes automatizados de documentación que pueden producir listados de referencias cruzadas, diccionarios de datos, diagramas de flujo de datos detallados, diagramas de estructura, o diagramas de flujo del sistema directamente desde el programa (a esto algunos encargados de mantenimiento le llaman ingeniería en reversa). Otro enfoque, como se mencionó anteriormente, es motivar a los usuarios a realizar sus propias modificaciones de mantenimiento.[8]

7 Existen diversos productos comerciales en esta area. Entre los más conocidos están *Superstructure*, de Group Operation, Inc; *Structured Retrofit*, comercializada por Peat, Marwick; y *Recorder*, de Language Technology, Inc.

8 Esto es particularmente relevante, pues de acuerdo con un estudio [Lientz y Swanson, 1980] aproximadamente el 42 por ciento de la actividad de mantenimiento en una organización típica consiste en "mejorías del usuario", en comparación con el 12 por ciento para "composturas de emergencia", el 9 por ciento para "correcciones de rutina", el 6 por ciento para el acomodo de "cambios de hardware", etc. De la porción invertida en mejorías, el 40 por ciento se invirtió en tratar de hacer reportes nuevos adicionales, el 27% en añadir datos a los reportes existentes, el 10 por ciento en modificar el formato de reportes sin cambiar el contenido de datos, y el 6 por ciento en consolidar datos de reportes existentes y lenguajes de cuarta generación. Es probable que muchos de estos cambios los hubiera podido hacer directamente el usuario.

Otro enfoque más es el de documentar cuidadosamente la naturaleza específica del trabajo de mantenimiento: a menudo resulta que tan sólo un 5 por ciento de la codificación en un sistema operacional es responsable de un 50 por ciento o más del trabajo de mantenimiento.

- *Disciplinas de ingeniería de software.* Otro enfoque más para la mejor productividad es un conjunto de herramientas, técnicas y disciplinas generalmente conocidas como ingeniería de software o técnicas estructuradas que incluyen la programación estructurada, el diseño estructurado y el análisis estructurado,[9] además de disciplinas relacionadas con esto, tales como métrica de software, pruebas de corrección de programas, y control de calidad de software. En general, las técnicas estructuradas han tenido un impacto modesto, típicamente una mejora de un 10 a 20 por ciento, sobre la productividad de los *profesionales* del desarrollo de sistemas *durante la fase de desarrollo del proyecto*. Sin embargo, los sistemas desarrollados utilizando técnicas estructuradas en general tienen costos de mantenimiento substancialmente más bajos y una confiabilidad considerablemente mayor, a menudo por un factor de 10 o más. Esto tiende a liberar recursos que de otra manera estarían acaparados por el mantenimiento o el arreglo de fallas, con lo cual se mejora la productividad de toda la organización.

- *Herramientas automatizadas para el desarrollo de sistemas.* Por último, observamos que una razón del problema de la productividad es que mucho del trabajo de desarrollo de un sistema automatizado de información se realiza, irónicamente, de manera manual. Así como los hijos del zapatero son los últimos en recibir zapatos, los programadores y analistas tradicionalmente han sido los últimos en gozar de los beneficios de la automatización en su propio trabajo. Desde luego, se puede argüir que un compilador es una herramienta automatizada para la programación, así como los paquetes de prueba y los auxiliares en corrección de errores proporcionan una forma de automatización. Pero, hasta recientemente, ha habido poca ayuda automatizada para el diseñador de sistemas y casi nada para el analista. Ahora hay estaciones de trabajo gráficas que auto-

9 El enfoque del análisis discutido en este libro representa la forma actual del análisis estructurado. Como veremos en el capítulo 7, ha habido cambios desde que se introdujera por primera vez en los textos a fines de los años 70.

matizan la mayor parte de la tediosa labor de desarrollar y mantener diagramas de flujo de datos, diagramas de entidad-relación y otros modelos gráficos que vimos en el Capítulo 4. Estas herramientas automatizadas también se encargan de una variedad de actividades de revisión de errores, lo cual asegura que la especificación producida por el analista sea completa, no ambigua, e internamente firme. Y, en algunos casos, las herramientas automatizadas pueden incluso generar código directamente de la especificación, eliminando de esta manera la actividad manual de manera absoluta. En el apéndice A se tratan detalles de estas herramientas automatizadas para el análisis de sistemas.

Muchos de estos enfoques de productividad pueden usarse en conjunto, pues comprenden conceptos y técnicas complementarias. Individualmente, cada uno de los enfoques antes expuestos puede llevar a un 10 a 50 por ciento de mejoría; tomados en conjunto, fácilmente pueden doblar la productividad de la organización y, en casos especiales, tal vez incluso mejorar la productividad por un factor de 10. En [Jones, 1986] se puede encontrar una exposición excelente sobre el impacto cuantitativo, de estos métodos así como de un gran número de factores de productividad.

Como analista su reacción a todo esto pudiera ser: "¿Y qué? ¿Por qué es relevante esto?" De hecho, parece que la mayoría de los asuntos relacionados con la productividad tiene que ver con la programación, la prueba y el mantenimiento. Ninguno parece tener que ver con el análisis de sistemas. Sin embargo, existen tres razones por las cuales, como analista, debe ser muy sensible respecto al asunto de la productividad:

1. La calidad del trabajo hecho por el analista puede tener un impacto tremendo sobre el tiempo que se invierte en pruebas, dado que el 50 por ciento de los errores (y el 75 por ciento del costo de su eliminación) usualmente se asocian con errores en el análisis. Pudiera culparse a los programadores de la baja productividad por el tiempo que invierten en realizar pruebas, pero a menudo esto es un indicic de la poca calidad de la labor realizada por el analista.

2. Algunas de las técnicas de productividad –más y mejor gente, mejores condiciones de trabajo y, sobre todo, las herramientas automatizadas–, son de relevancia directa para el analista. Vale la pena pensar qué puede hacerse para volver a *usted* y a *su trabajo* más productivos.

3. La productividad del análisis de sistemas es un asunto políticamente delicado, pues a menudo le parece al usuario (y a veces a los administradores del grupo de desarrollo de sistemas y de otras partes de la organización) que se hace muy poco durante la fase de análisis. Con frecuencia se escucha el comentario: "¿Cuándo van a comenzar con la programación? ¡No podemos darnos el lujo de estar aquí sentados para siempre platicando acerca del sistema; ya necesitamos que se realice!" Y los usuarios no le atribuyen gran valor a la especificación funcional, que es el producto del análisis del sistema. La reacción que se da a la especificación a veces será: "¿Para qué tanto alboroto con todas estas imágenes y palabras? Le dijimos lo que queremos que haga el sistema. ¿Para qué tuvo que escribir todo esto?".

El hecho es que no se puede elaborar un sistema que dé buen resultado, de alta calidad y mantenible si no se sabe precisamente, y con suficiente detalle, qué se supone que debe poder hacer. Así que, a pesar de que algunos usuarios y administradores pudieran quejarse de que el análisis es meramente un periodo de "descanso" mientras se prepara para la *verdadera* labor del proyecto (la programación), el hecho es que debe hacerse de manera cuidadosa y rigurosa. Pero también debe hacerse con tanta eficiencia y productividad como sea posible; así que conviene al analista no pensar que el análisis es simple cosa de programación.

6.2 CONFIABILIDAD

Otro gran problema al que se enfrentan los que desarrollan sistemas es el de la confiabilidad. El largo tiempo que se invierte en la prueba y la corrección de errores –típicamente un 50 por ciento del proyecto de desarrollo de un sistema–, y la muy poca productividad (que muchos sienten que se relaciona con el tiempo que se invierte en probar) pudieran ser aceptables si el resultado fuesen sistemas altamente confiables y fácilmente mantenibles. La evidencia de los últimos 30 años es justo lo contrario: los sistemas producidos por las organizaciones en todo el mundo están llenos de errores y son casi imposibles de modificar.

Estar "llenos de errores" significa cosas diferentes para diferentes personas. En promedio, el software desarrollado en los Estados Unidos tiene entre tres y cinco errores por cada 100 instrucciones del programa. Esto *después* de que el software ha sido probado y entregado al cliente, –véase [Jones, 1986]–. Unos cuantos proyectos ejemplares de desarrollo de software han reportado sólo de tres a cinco errores por cada 10 mil instrucciones, desde el proyecto del superprogramador de IBM [Baker, 1972]. Además, ha habido reportes pesimistas, como [Sanger, 1985], que sugieren que el software estadounidense puede tener hasta de tres a cinco errores por cada 10 instrucciones del programa.

Los errores de software van desde los sublimes hasta los ridículos. Un error trivial pudiera consistir en salidas (resultados) correctas, pero que no se imprimen o presentan tan bien como el usuario desearía. Un error moderadamente serio pudiera ser un caso en el cual el sistema se rehusa a reconocer ciertos tipos de entradas, pero el usuario final puede hallar alguna forma de saltar el problema. Los errores serios son aquellos que ocasionan que todo el programa deje de funcionar, con una pérdida asociada de dinero o de vidas humanas. Algunos ejemplos de errores serios relacionados con software que se han ido documentando en el transcurso de los años son los siguientes:

- En 1979, el sistema SAC/NORAD (siglas en inglés de Comando Aéreo Estratégico/Defensa Aérea Norteamericana) registró 50 falsas alarmas, incluyendo un ataque simulado, cuyos resultados o salidas comenzaron accidentalmente una "escaramuza" en vivo.
- Un error en el programa simulador de vuelo del F16 hacía que el avión volara en posición invertida (cabeza abajo), cada vez que cruzaba el ecuador.
- Un proyectil F18 comenzó su propulsión estando todavía unido al avión e hizo que éste perdiera 6 000 metros de altitud.
- Las puertas del sistema de trenes BART, controlado por computadora en San Francisco, se abren a veces en tramos largos entre estaciones.
- Una señal de alerta de la NORAD, del Sistema de Advertencia Rápida de Proyectiles Balísticos (BMEWS), detectó a la Luna como un proyectil que llegaba.
- El índice de la Bolsa de Vancouver perdió 574 puntos en un periodo de 22 meses debido a redondeos (por ejemplo, redondear 3.14159 a 3.1416)
- El 28 de noviembre de 1979, un avión de Air New Zealand se estrelló en una montaña. Las investigaciones posteriores mostraron que se había detectado y corregido un error en los datos de curso de la computadora, pero que jamás se informó de esto al piloto.

Desafortunadamente, la lista sigue y sigue. Para ver más ejemplos, remítase a [Neumann, 1985]. Muchos errores de software jamás se reportan porque el individuo o la organización "culpables" preferirían no admitirlo públicamente. Al momento de escribirse este libro, había gran preocupación por el hecho de que errores de este tipo pudieran llegar a consecuencias lamentables con el programa Guerra de las Galaxias, del Departamento de Defensa de Estados Unidos, o con algunos otros sistemas complejos controlados por software de defensa aérea; véase [Jacky, 1985] y [Adams y Fischetti, 1985]. De hecho, aun si la confiabilidad del software de la Guerra de las Galaxias fuese 100 veces mayor que la de los sistemas promedio desarrollados en Estados Unidos, pudiera todavía tener alrededor de 10 000 errores, lo cual no es una perspectiva tranquilizante si se considera que cualquiera de esos errores pudiera acabar con la vida en este planeta.

En muchos casos, nadie está muy seguro respecto a cuántos errores tiene un sistema, pues 1) algunos errores jamás se encuentran antes de que caduque el sistema y, 2) el proceso de documentación y registro de errores es tan descuidado que *la mitad* de los errores encontrados no se reportan,[10] aun dentro de la organización misma de desarrollo de sistemas. En cualquier caso, el fenómeno típico de descubrimiento de errores, en un periodo de varios años de utilidad de un sistema de software, usualmente toma la forma mostrada en la figura 6.1.

10 Esto se basa en la encuesta hecha por Lientz y Swanson [Lientz y Swanson, 1980].

La forma de esta curva recibe influencias de buen número de factores. Por ejemplo, cuando el sistema se entrega por primera vez a los usuarios finales, a menudo no pueden ponerlo a trabajar a su máxima capacidad. Lleva tiempo convertir su sistema anterior (que pudiera haber sido un sistema manual) y capacitar a su personal. Además, desconfían un poco de la computadora y no la quieren usar demasiado, por lo que se descubren pocos errores. Al convertir su operación anterior al nuevo sistema, a medida que su personal operacional ya está mejor preparado y que dejan de sentirse intimidados, empiezan a usar más el software y se encuentran cada vez más errores. Desde luego, si esto continuara indefinidamente, si se encontraran cada día más errores, los usuarios finalmente dejarían de usar el software y lo desecharían.[11] En la mayoría de los casos, los programadores arreglan frenéticamente nuevos errores al irlos descubriendo los usuarios. En la mayoría de los casos, llega un momento en el cual el sistema empieza a estabilizarse y los usuarios encuentran cada vez menos errores.

Existen tres aspectos deprimentes en la figura 6.1. Primero, la curva jamás regresa a cero; es decir, casi nunca encontramos una situación donde transcurra el tiempo sin encontrar nuevos errores. Segundo, el área bajo la curva, que representa

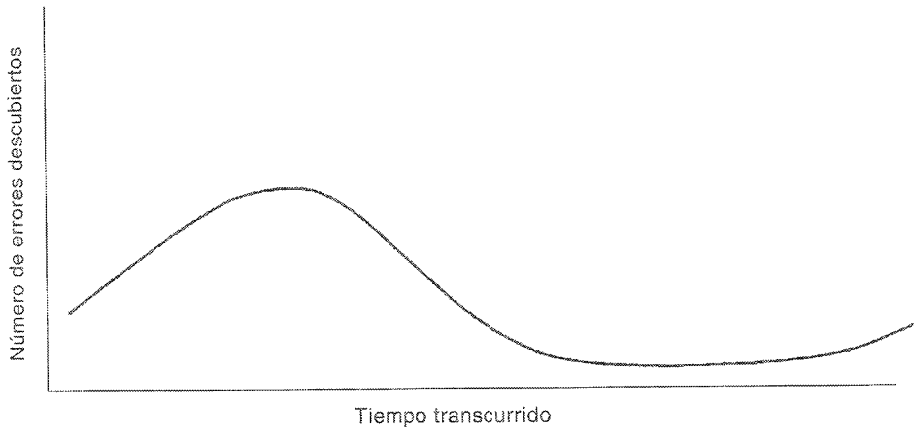


Figura 6.1: **Errores descubiertos como función del tiempo**

11 Desde luego, hay excepciones al acoplamiento gradual, sobre todo cuando un nuevo sistema tiene que aceptar de una vez todo el trabajo (transacciones) del anterior. Como un ejemplo interesante, en el cual se renovó el sistema de bonos de cobertura nacional de Dinamarca, véase [Hansen, 1984].

el número total de errores descubiertos en función del tiempo, es atrozmente grande; su promedio es de tres a cinco errores por cada cien instrucciones del programa. Y, tercero, la curva finalmente comienza a subir de nuevo, por lo común después de varios años. Por último, todos los sistemas de software llegan a un estado de parchado tal que cualquier intento de eliminar un error introduce otros dos nuevos y las modificaciones necesarias para eliminarlos introducirán cuatro, y así en adelante.

Existe un último problema que cabe señalar acerca de los errores de software: no son fáciles de enmendar. Cuando alguien, ya sea programador, usuario final o algún otro intermediario, descubre que el software no funciona correctamente, deben suceder dos cosas: 1) el programador debe identificar el origen y la naturaleza del error y, 2) debe encontrar la manera de corregir el error (ya sea cambiando algunas instrucciones de programación existentes, quitando otras o añadiendo nuevas) sin afectar algún otro aspecto de la operación del sistema. Esto no es fácil de hacer; de hecho, el programador tiene menos de un 50 por ciento de probabilidades de éxito en su primer intento, y éstas probabilidades se reducen rápidamente si el programador debe modificar más de 10 a 20 instrucciones (véase [Yourdon, 1986]).

6.3 MANTENIBILIDAD

La corrección de errores sobre la marcha es un aspecto del mantenimiento. Lientz y Swanson [Lientz y Swanson, 1980] encontraron que en esto consistían más del 21 por ciento de los esfuerzos de mantenimiento global en las organizaciones estadounidenses de proceso de datos.[12] El mantenimiento también involucra la modificación de un sistema para reflejar cambios en el hardware, modificaciones para apresurar ciertos aspectos operacionales o modificaciones para reflejar cambios en los requerimientos del usuario final del sistema.

El mantenimiento de software es un problema primordial en muchas organizaciones, entre el 50 y el 80 por ciento del trabajo que se hace en la mayoría de las organizaciones de desarrollo de sistemas se asocia con la revisión, modificación, conversión, mejoramiento o corrección de errores en algún programa de computadora que otra persona escribió hace mucho. Y es caro: a comienzos de los años 70, la Secretaría de la Defensa de Estados Unidos informó que el costo promedio de desarrollar programas de computadora en un proyecto era de 75 dólares estadounidenses por instrucción de computadora; el costo de mantener el sistema llegaba hasta los 4 000 dólares estadounidenses por instrucción.

Para poner esto de una manera mucho más clara considere los siguientes ejemplos de la Administración del Seguro Social de los Estados Unidos:

12 Dado que la industria de la computación representa aproximadamente un 8 o 10 por ciento del PIB de los EUA, esto significa que se están gastando aproximadamente 75 dólares estadounidenses por cabeza al año en mantenimiento de software.

- Calcular el incremento en el costo de la vida de 50 millones de receptores de los beneficios del Seguro Social implica el uso de 20 000 horas de tiempo de máquina en los sistemas más viejos de cómputo del Sistema del Seguro Social (véase [Rochester y Gantz, 1983].

- Cuando el Sistema de Seguro Social aumentó sus sistemas de cómputo en 1981 de cheques de cinco cifras a cheques de seis cifras, requirió de 20 000 horas-persona de trabajo y 2 500 horas de tiempo de máquina para modificar 600 programas distintos.

- La moral del departamento de mantenimiento del Seguro Social estaba tan baja en un momento dado que se sorprendió a uno de los programadores orinando sobre un paquete de discos en la sala de computadoras. Aunque esto definitivamente es una manera novedosa de descargar la frustración, no resulta muy bueno para el paquete de discos.

El resultado, en más de alguna organización de proceso de datos, es que los sistemas existentes que se crearon hace 10 o 20 años simplemente no pueden modificarse para ajustarlos a las nuevas exigencias del gobierno, de la economía, del clima o de la volubilidad del usuario. Al paso que las compañías y la sociedad se vuelven cada vez más dependientes de las computadoras, encontramos una analogía interesante: en la medida en que se estanca el software se estancará la compañía o la sociedad a la que sirve dicho software.

El problema es peor aún. Si fuera simplemente un caso de que el software fuese malo, se podría considerar desecharlo o reemplazarlo. Pero muchas organizaciones jamás han capitalizado su software (los costos se determinan cada año), y sus políticas de administración y de negocios hacen que se vuelva prohibitivamente caro reemplazar los sistemas antiguos. Y existe un problema aún más fundamental: en la mayoría de las organizaciones, no existe una descripción coherente de lo que se supone que los sistemas deben hacer. La documentación existente suele ser obsoleta y confusa. En cualquier caso, proporciona, cuando más, alguna idea de *cómo* funciona el software, pero no de *cuál* es su propósito fundamental, o qué política de negocios se supone que debe realizar.

Por eso, aunque se pueda argumentar que la mantenibilidad es enteramente función de la implantación (es decir, algo que preocupa a los programadores), es imposible mantener un sistema si no existe un modelo preciso y actualizado de sus requerimientos. Esto, a fin de cuentas, es labor del analista. Como se verá en el capítulo 8, las especificaciones funcionales desarrolladas por los analistas han progresado gradualmente, desde novelas victorianas absolutamente inmantenibles (miles de hojas de texto narrativo) a modelos gráficos del sistema hechos a mano, y hasta modelos generados y mantenidos por la computadora. También se discutirá el asunto del mantenimiento continuo de las especificaciones del sistema en el capítulo 24.

6.4 OTROS ASUNTOS

¿De qué tiene que preocuparse el analista además de la productividad, la confiabilidad y la mantenibilidad? La lista varía de una organización a otra y de un proyecto a otro, pero por lo común incluye lo siguiente:

- *Eficiencia.* Un sistema debe operar mediante salidas o productos (o resultados) apropiados (usualmente medidas en transacciones por segundo) y con un tiempo de respuesta adecuado para las terminales en línea. Este no suele ser asunto del que se tenga que preocupar el analista, puesto que los diseñadores y programadores tendrán la influencia principal en la eficiencia global del sistema ya realizado. De hecho, se está volviendo un asunto cada vez menos importante para los sistemas modernos, dado que los costos de hardware se vuelven menores cada año, mientras que la potencia y la rapidez aumentan continuamente.[13]

- *Transportabilidad.* La mayoría de los sistemas nuevos se realizan en una marca de computadora, pero pudiera haber necesidad de desarrollar el sistema de tal manera que se le pueda mudar fácilmente entre computadoras. Nuevamente, éste no es asunto que deba preocupar al analista, aunque pudiera requerir que se especifiquen las necesidades de transportabilidad en el modelo de implantación.

- *Seguridad.* Dado que los sistemas modernos de computación son cada vez más accesibles (pues tienden a estar en línea), y dado que se utilizan para administrar bienes cada vez más delicados de la organización, la seguridad se está convirtiendo en un asunto de importancia para muchos proyectos de desarrollo de sistemas. El nuevo sistema debe evitar el acceso no autorizado, además de la actualización y la eliminación no autorizadas de datos delicados.

6.5 RESUMEN

Varios expertos predicen que la razón matemática o proporción precio/rendimiento del hardware de computadora mejorará por un factor de mil y posiblemente hasta de un millón, en los próximos 10 a 15 años. Desafortunadamente, la historia del desarrollo de software en las últimas tres décadas llevaría al observador promedio a concluir que la tecnología de software mejorará sólo en una cantidad modesta.

13 Hay algunas excepciones a esta afirmación optimista. Para algunas aplicaciones críticas (predicción del clima, investigación nuclear, modelado de propiedades aerodinámicas de aeroplanos y automóviles, etc.), aún no es adecuada la tecnología computacional actual. Para una mayor exposición de esto véase [Yourdon, 1986]. Para muchos sistemas de tiempo real e interconstruidos, la tecnología sí es adecuada, pero los diseñadores y programadores deben trabajar duro para lograr un nivel de eficiencia aceptable. En algunos casos la tecnología de hardware parece adecuada, pero el nuevo software (por ejemplo, lenguajes de cuarta generación) resultan ser ineficientes a tal grado que el sistema global no es aceptablemente eficiente.

Dado que el software se ha vuelto el principal costo y la "ruta crítica" de muchos sistemas, una mejora tan modesta no puede considerarse aceptable. En toda la industria computacional existe un esfuerzo enorme y concertado para realizar mejoras de un orden de magnitud en el proceso de desarrollo de software.

Las técnicas de análisis presentadas en este libro son parte de dicho esfuerzo. Como se ha visto, parte del esfuerzo es asunto de la programación o diseño del sistema; pero una buena especificación del sistema crea los cimientos en los cuales deben sostenerse el diseño y la programación.

REFERENCIAS

1. Robert Alloway y Judith Quillard, "User Manager's Systems Needs", *CISR Working* Paper 86. Cambridge Mass.: MIT Sloan School for Information Systems Research, abril 1982.

2. Harold Sackman, W.J. Erickson, y E.E. Grant, "Exploratory Experimental Studies Comparing Online and Offline Programming Performance", *Communications of the ACM*, enero 1968, pp. 3-11.

3. J. Aron, "The Super-Programmer Project", *Software Engineering Concepts and Techniques*. eds. J.M. Buxton, P. Naur y B. Randell. Nueva York: Petrocelli/Charter, 1976, pp. 188-190.

4. F.T. Baker, "Chief Programmer Team Managment of Production Programming", *IBM Systems Journal*, volumen 11, número 1 (enero 1972), pp. 56-73.

5. H.D. Mills y F.T. Baker, "Chief Programmer Teams", *Datamation*, volumen 19, número 12 (diciembre, 1973), pp. 58-61.

6. Edward Yourdon, *Managing the Structured Techniques: Strategies for Software Development in the 1990s*, 3ª edición, Nueva York: YOURDON Press, 1986.

7. Bennett P. Lientz y E. Burton Swanson, *Software Maintenance Management*. Reading, Mass: Addison Wesley, 1980.

8. T. Capers Jones, *Programming Productivity*. Nueva York: McGraw-Hill, 1986.

9. T. Capers Jones, "A Software Productivity Survey", discurso en la Primera Conferencia Nacional sobre Calidad de Software y Productividad, Williamsburg, Virginia, marzo 1985.

10. Edward Yourdon, *op cit.*

11. F.T. Baker, *op cit.*

12. David Sanger, "Software Fears on Star Wars", *New York Times*, 4 de julio de 1985.

13. Peter G. Neumann, "Some Computer-Related Disasters and Other Egregious Horrors", *ACM SIGSOFT Software Engineering Notes*, enero 1985.

14. Jonathan Jacky, "The Star Wars Defense Won't Compute", *Atlantic Monthly*, junio de 1985.

15. John A. Adams y Mark A. Fischetti, "Star Wars-SDI: The Grand Experiment", *IEEE Spectrum*, septiembre de 1985, pp. 34-64.

16. Artículo del *New York Times*, alrededor del 16 de septiembre de 1986, que comentaba el número de errores en la Guerra de las Galaxias.

17. Dines Hansen, *Up and Running*. Nueva York: YOURDON Press, 1984.

18. Edward Yourdon, *op cit.*

19. Bennett P. Lientz y Burton Swanson, *op cit.*

20. Jack Rochester y John Gantz, *The Naked Computer*. Nueva York: William Morrow, 1983.

21. Edward Yourdon, *Nations at Risk*. Nueva York: YOURDON Press, 1986.

22. Robert Block, *The Politics of Projects*, Nueva York: YOURDON Press, 1981.

EJERCICIOS Y PROBLEMAS

1. Examine un reporte financiero de una compañía pública grande para ver si puede determinar cuánto se gasta en desarrollo de sistemas. ¿Cuánto dinero se ahorraría si se pudiera doblar la productividad de la organización de desarrollo de sistemas?

2. Escoja una compañía pública grande que sea obvio que dependa de computadoras para su operación diaria. Estime cuántos días, semanas o meses podría continuar funcionando la empresa si se detuvieran sus sistemas de computación.

3. ¿Cuáles son los tres principales aspectos del desarrollo de sistemas actualmente?

4. ¿Por qué es probable que la productividad sea el problema más visible en el desarrollo actual de sistemas?

5. ¿Cuáles son los tres tipos de retraso que se pueden encontrar en una organización típica actualmente?

6. Proyecto de investigación: haga una encuesta en su organización para ver qué tan grande es el retraso en el desarrollo de sistemas. ¿Se conocen estas cifras entre los usuarios y administradores en su organización?

7. ¿Cuál es la diferencia entre retraso visible e invisible?

8. ¿Cuándo existe un retraso desconocido?

9. ¿Por qué es probable que el retraso invisible sea mucho mayor que el visible?

10. ¿Cuáles son las siete principales soluciones que siguen las organizaciones para resolver sus problemas de productividad? ¿Puede sugerir otras?

11. ¿Cómo cree que una empresa deba medir la productividad de su organización de desarrollo de sistemas?

12. ¿Qué tan práctico resulta solucionar el problema de la productividad contratando más programadores y analistas? ¿Cuáles son las ventajas y desventajas?

13. ¿Cree que sea práctico resolver el problema de productividad empleando programadores o analistas con más talento? ¿Por qué sí o por qué no?

14. Suponga que hubiese un programador diez veces más productivo que el programador promedio que percibe 250 mil dólares de EUA, anuales. ¿Cree que la administración de una organización típica esté dispuesta a gastar 250 mil dólares de EUA, anuales en el programador con talento? ¿Cree que debieran estar dispuestos a ello? ¿Por qué sí o por qué no?

15. ¿Qué tan práctico cree que sea resolver el problema de la productividad dejando a los usuarios desarrollar sus propios sistemas? ¿Cuáles son las ventajas y desventajas?

16. ¿Qué tipo de sistemas serían más apropiados para que los usuarios los desarrollaran por sí solos? ¿Qué porcentaje de proyectos en una organización típica cree que sea probable que entren en esta categoría?

17. ¿Qué tan práctico cree que sea utilizar nuevos lenguajes de programación, ya sea de tercera generación, como Ada, o de cuarta como Focus, RAMIS o NOMAD, para resolver el problema de la productividad? ¿Cuáles son las ventajas y desventajas de este enfoque?

18. Proyecto de investigación: ¿Cuánto mejoraría la productividad en su organización durante los próximos cinco años si se comenzara a utilizar un nuevo lenguaje de programación? ¿Cómo se vería afectado esto por el código *existente* y por la "cultura" existente de los programadores y analistas?

19. ¿Por qué un lenguaje de programación de cuarta generación proporciona una mayor mejoría en la productividad que uno convencional de tercera generación?

20. ¿Cuánto mejoraría la productividad en una organización típica si el mantenimiento pudiera reducirse en un factor de 10?

21. Proyecto de investigación: Utilice un producto comercial de tipo "máquina de estructuración" para reestructurar un programa existente en su organización, y mida la reducción de los costos de mantenimiento. ¿Qué le dice esto acerca de los beneficios potenciales de una máquina o método de estructuración?

22. ¿Cree que la reestructuración pueda convertir programas buenos en malos? ¿Por qué sí o por qué no? Si su respuesta es no, entonces, ¿cuál es el propósito de la reestructuración de programas?

23. ¿Pueden los usuarios llevar a cabo su propio mantenimiento de software? ¿Qué se ocupa para que esto funcione? Hablando en forma realista, ¿cuánto trabajo de mantenimiento de software cree que los usuarios pudieran ser capaces de hacer?

24. ¿Por qué la ingeniería de software puede mejorar la productividad?

25. ¿Por qué pueden mejorar la productividad las herramientas automatizadas de desarrollo de software?

26. ¿En qué puede afectar al trabajo del analista la productividad en un proyecto de desarrollo de sistemas?

27. ¿Qué tanto de un proyecto se invierte en pruebas y corrección de errores?

28. Proyecto de investigación: ¿Cuál es el número promedio de errores en los sistemas desarrollados por su organización? ¿Cuál es la varianza: la diferencia entre el pecr y el mejor?

29. Proyecto de investigación: encuentre por lo menos dos ejemplos documentados de fallas en el último año en sistemas que hayan causado pérdidas de vidas o que hayan costado más de 1 millón de dólares de EUA. ¿Cómo pudieron haberse evitado estas fallas?

30. ¿Por qué tiende a *aumentar* el número de errores en un sistema después de que se pone a funcionar por primera vez?

31. ¿Por qué tiende a aumentar el número de errores gradualmente tras haber estado funcionando varios años el sistema?

32. Proyecto de investigación: ¿qué porcentaje de los recursos de su organización de desarrollo de sistemas se invierten en mantenimiento? ¿Está al tanto la administración superior de estas cifras?

33. ¿Qué factores, además de la productividad, calidad y confiabilicad, son importantes en la organización típica actual de desarrollo de sistemas?

34. ¿Qué papel representa el analista en la determinación de la eficiencia de un sistema de información actualmente?

35. ¿Qué papel juega el analista en la determinación de la transportabilidad de un sistema de información en la actualidad?

36. ¿Qué papel tiene el analista en la determinación de la seguridad en un sistema de información actualmente?

7 CAMBIOS EN EL ANALISIS DE SISTEMAS

Se han acuñado nuevos términos, tales como tecno-stress y tecno-shock, para definir las manifestaciones psicológicas de un público avasallado por toda clase de cosas, desde hornos de microondas hasta juegos caseros de Pac-Man. Desafortunadamente, estos términos no describen adecuadamente el progreso en la industria del procesamiento de datos, en lo relacionado con el desarrollo de software. Para muchos profesionales del proceso de datos, el tecno-stress se define mejor como la frustración que trae el lento cambio que se está dando en los métodos de desarrollo de software, ante la siempre creciente demanda de servicios de procesamiento de datos.

Aunque no hay duda de que se ha tenido algún progreso orientado hacia mejores métodos de desarrollo de sistemas durante los últimos 30 años, tampoco la hay de que, en general, cualquier proceso de cambio es lento y discontinuo. Desde una perspectiva histórica, parece ser que para que se logre un verdadero progreso debe haber un replanteamiento periódico y colectivo de las ideas básicas. Los lapsos que hay entre cada gran salto de avance pueden ser de decenas a cientos de años.

F.J. Grant, "El software del siglo XXI"
Datamation, 1º de abril de 1985

En este capítulo se aprenderá:

1. Los problemas del análisis clásico de sistemas.
2. Los cambios que se han dado en el análisis estructurado clásico.
3. Por qué las herramientas automatizadas son importantes para el futuro del análisis de sistemas.
4. Por qué los problemas del análisis estructurado clásico han llevado a la elaboración de prototipos.

Los métodos y herramientas de análisis de sistemas que se presentan en este libro representan un enfoque de lo más avanzado que se usará en la mayoría de las organizaciones de desarrollo de sistemas durante el resto de los años 80 y comienzos de los 90. Para mediados de los 90 es probable que el análisis de sistemas haya evolucionado sustancialmente; en el capítulo 25 se discute la probable naturaleza de dicha evolución.

Pero no es suficiente estar al tanto de las técnicas *actuales* de análisis de sistemas. También deben comprenderse los cambios que han sucedido en los últimos cinco a diez años, que han conducido a las técnicas y herramientas que exploraremos con mayor detalle en las partes II y III. Existen tres razones por las cuales debe estar familiarizado con la evolución del análisis de sistemas.

Primero, pudiera ser que encuentre que la organización de desarrollo de sistemas para la que trabaja no ha evolucionado y que nadie tiene intención de hacer cambios. Aunque los cambios que se discuten en este capítulo ocurrieron en aproximadamente un 80 % de las organizaciones de proceso de datos de Estados Unidos, Canadá, Europa y otras partes del mundo, aún quedan aquellos baluartes de la mediocridad que no ven razón alguna para cambiar la forma en la que han venido trabajando desde hace 20 años. Si se encuentra en esta situación, lo lógico sería buscarse otro empleo; o, si se siente con valor, pudiera adoptar un papel de líder y ayudar a su organización a entrar al mundo moderno del análisis de sistemas.[1]

Segundo (y más probablemente), pudiera encontrar que su organización ha comenzado a implantar algunos de los cambios que se tratan en este capítulo, pero que el proceso de cambio durará otros cuantos años más. Un buen ejemplo de esto es el desarrollo de herramientas automatizadas para el análisis de sistemas. Casi todos los analistas le cirán que estas herramientas basadas en PC (computadoras personales) son una buena idea, y algunos administradores de proceso de datos están comenzando a apoyar el concepto. Pero las herramientas son relativamente nuevas; casi no existían antes de 1984. Y las organizaciones son lentas en hacer el cambio; para fines de 1987 se estimaba que menos del 2 por ciento de los analistas de sistemas en los Estados Unidos tenían acceso a las herramientas que se tratan en este libro; para 1990 se estima que aproximadamente el 10% de los analistas las estarán utilizando, y probablemente no ocurrirá una verdadera saturación del mercado hasta mediados de los años 90. Por ello, aunque usted, y otros miembros de su organización sepan qué herramientas y técnicas se instalarán dentro de tres años, es posible que el enfoque *actual* del análisis de sistemas sea algo diferente. Es importante que sepa qué enfoque estuvo utilizando anteriormente la organización y qué tipo de transición está ocurriendo.

1 Esta sugerencia no es frívola. Si estas organizaciones no cambian, quedarán fuera del mercado en algún periodo de los años 90.

Tercero, la noción de *transición* es importante aun si trabaja en una de las organizaciones "líderes" que han implantado completamente el enfoque de análisis que se presenta en este libro. El campo del análisis de sistemas, como todos los demás aspectos de las computadoras, es dinámico; la manera en la que se lleve el análisis de sistemas en 1995 indudablemente será diferente de como se hace ahora. Para dilucidar los cambios que sucederán durante la última mitad de los años 90 se requiere una buena apreciación del origen de este campo de actividades, así como de hacia dónde se dirige.

7.1 EL MOVIMIENTO HACIA EL ANALISIS ESTRUCTURADO

Hasta fines de los años 70, la gran mayoría de los proyectos de desarrollo de sistemas empezaban con la creación de una "novela victoriana" de requerimientos del usuario. Es decir, el analista escribía lo que entendía de los requerimientos del usuario en un enorme documento que consistía primariamente en una narrativa. Los primeros autores de textos de "análisis estructurado", sobre todo [De Marco, 1978], [Gane y Sarson, 1977] y [Weinberg, 1978], señalaron que estos pesados tomos (a menudo conocidos como "especificación funcional") se veían afectados por diversos problemas importantes:

- Eran *monolíticos:* había que leer completamente la especificación, de principio a fin, para poder entenderla. Como en una novela victoriana, si no se leía la última página, se tendría poca idea de cómo terminaría la historia.[2] Esta es una falla importante, pues existen muchas situaciones en las que el analista quisiera leer y comprender una parte de la especificación sin tener necesariamente que leer las demás.

- Eran *redundantes:* a menudo se repetía la misma información en diversas partes del documento.[3] El problema con esto es que si cambia cualquier aspecto de los requerimientos del usuario durante la fase de análisis (o peor aún, después de declararse terminada ésta), el cambio debe reflejarse en diversas partes del documento. Este sería un problema menor hoy, pues hasta la organización más primitiva cuenta con amplio acceso a instrumentos y medios de procesamiento de palabras; pero en los años 60 y 70, la mayoría de las organizaciones creaban sus especificaciones funcionales con nada más elaborado que una máquina de escribir eléctrica.[4]

2 O, para ponerlo de otro modo, nunca hay sexo sino hasta la última página.

3 Existen diversas teorías en cuanto a por qué la redundancia resulta ser una característica tan común. En mi propia experiencia, la especificación funcional servía para tres propósitos en la organización (y por lo tanto la misma información se presentaba de tres maneras distintas): *primero,* era la declaración "oficial" de los requerimientos del usuario; *segundo,* era un manual extraoficial de entrenamiento, con la intención de explicar cómo los "tontos" usuarios operarían el sistema; y *tercero,* era una herramienta de ventas orientada políticamente, con la intención de fijar en las mentes de la administración la impresión de que el sistema sería tan maravilloso que bien valdría su costo.

4 Tal vez uno de los mejores ejemplos de este problema se dio en la organización de proceso de datos de un banco neoyorquino importante a mediados de los años 70. Embarcado en un proyecto característico de desarrollo de sistema tipo "horda mongoliana", el grupo de análisis entrevistó a docenas de usuarios en todo el banco y gradualmente desarrolló una especificación tipo novela victoriana de gigantescas proporciones. Transcribir el documento le llevó dos semanas al grupo de mecanografía y todas las copiadoras se monopolizaron durante días para poder hacer duplicados suficientes para todos los usuarios. Se le dio una semana a la comunidad usuaria para leer toda la especificación funcional e indicar los cambios o correcciones deseados. Un tanto para su sorpresa (pero también para su gran alivio) los analistas no recibieron comentario alguno de los usuarios para la fecha indicada. Por lo tanto, se declaró "congelada" la especificación y se dio inicio al diseño y a la programación. Tres semanas después, seis miembros de la comunidad anunciaron que finalmente habían logrado leer toda la especificación y, sí, tenían unos cuantos pequeños cambios qué señalar. De aquí siguió un leve pánico: ¿Qué se le debería hacer a la especificación? Tras dos acaloradas juntas en las que los usuarios y analistas insultaron mutuamente a sus respectivas parentelas e inteligencias en términos que no pueden repetirse en un libro como éste, se llegó a una decisión: los cambios no se introdujeron en la especificación escrita (pues eso resultaría demasiado difícil), pero sí se incorporarían al sistema mismo. O, para ponerlo de otro modo: el equipo encargado del proyecto encontró que era más fácil modificar el COBOL que el inglés.

5 Véase James Martin, *An Information Systems Manifesto* (Erglewood Cliffs, N.J.: Prentice-Hall, 1984).

Debido a que era tan difícil actualizar y revisar un documento, la redundancia inherente llevaba a un problema aún peor: la *inconsistencia.* Así como es poco probable que una persona que tenga muchos relojes sepa exactamente qué hora es, una especificación funcional que repita tres o cuatro veces la misma información es probable que tenga información errónea en diversos casos.

- Eran *ambiguas:* el reporte detallado de los requerimientos podía ser (y a menudo era) interpretado de diferente manera por el usuario, el analista, el diseñador y el programador. En estudios hechos a fines de los años 70[5] se encontró que el 50 por ciento de los errores que finalmente aparecían en un sistema operacional y el 75 por ciento del costo de su eliminación podían atribuirse a malos entendidos o errores en la especificación funcional.

- Eran *imposibles de mantener:* por todas las razones descritas anteriormente, la especificación funcional era casi obsoleta para cuando llegaba el final del proceso de desarrollo del sistema (es decir, para cuando el sistema se ponía en operación), y a menudo era obsoleto para el final de la fase de análisis. Esto significa que la mayoría de los sistemas que se desarrollaron durante los años 60 y 70, que ahora tienen 20 años o más de existir, no tienen definiciones actualizadas de las políticas de negocios que se supone que llevan a cabo. Y dado que los analistas y usuarios originales del sistema han desaparecido, la realidad es que *nadie sabe lo que la mayoría de los principales sistemas de cómputo están haciendo actualmente.*

Mientras se debatían todos estos problemas, ya se estaba adoptando un conjunto complementario de ideas en el área de programación y diseño. Estas ideas, normalmente conocidas como *diseño y programación estructurados,* prometían grandes mejoras en la organización, codificación, prueba y mantenimiento de los programas de computadora. Y, de hecho, sí han demostrado ser útiles, aunque cada vez más organizaciones de procesamiento de datos han empezado gradualmente a darse cuenta de que no tenía caso escribir programas brillantes y diseñar sistemas altamente modulares *si nadie sabía realmente qué era lo que se suponía que el sistema debería hacer.* En realidad, se podría argumentar que el diseño y la programación estructurados les permitían a algunos equipos de encargados de proyectos llegar a un desastre más rápidamente que antes, al construir una brillante solución al problema equivocado.

Como resultado, ha habido un movimiento gradual (puesto que aceptarlo le ha llevado a la profesión de desarrollo de sistemas alrededor de diez años) tendiente a hacer especificaciones funcionales que sean:

- *Gráficas:* compuestas de una variedad de diagramas, apoyadas con material textual detallado que, en muchos casos, sirve de material de referencia más que como cuerpo principal de la especificación.
- *Particionadas:* de tal manera que se puedan leer independientemente porciones individuales de la especificación.
- *Mínimamente redundantes:* de tal manera que los cambios en los requerimientos del usuario puedan incorporarse normalmente en sólo una parte de la especificación.

Este enfoque, al que por lo general se conoce como análisis estructurado, se utiliza ahora en la mayoría de las organizaciones de desarrollo de sistemas orientados a los negocios, al igual que en gran número de las orientadas hacia la ingeniería. Se pueden encontrar aún algunas organizaciones que produzcan especificaciones tipo novela victoriana, pero son minoría y, como los dinosaurios, se extinguirán.

7.2 CAMBIOS EN EL ANALISIS ESTRUCTURADO CLASICO

Como se mencionó anteriormente, el análisis tradicional de sistemas (que se caracteriza por especificaciones tipo novela victoriana) empezó a cambiar a fines de los años 70. La mayoría de las organizaciones que ahora usan el análisis estructurado basan su enfoque en los primeros textos de DeMarco, Gane y Sarson, y Weinberg y otros, al igual que en seminarios, videos y otros materiales de capacitación basados en dichos libros.

Sin embargo, varios años de experiencia práctica con el análisis estructurado clásico han señalado un buen número de áreas en las que es necesario hacer cambios o extensiones. Los principales cambios son:

- El énfasis en la construcción de modelos "físicos actuales" y "lógicos actuales" del sistema del usuario se han demostrado que es políticamente peligroso. A menudo, el equipo encargado del proyecto pasaba tanto tiempo (a veces hasta seis meses, un año o más) estudiando el sistema *anterior,* que todo mundo sabía que iba a desecharse y reemplazarse con el nuevo, que el proyecto acababa siendo cancelado por un usuario impaciente antes de que el equipo pudiera darse a la tarea de estudiar el *nuevo* sistema propuesto. Esto no quiere decir que hayamos decidido evitar modelar el sistema actual del usuario en todos los casos, sino que simplemente lo reconocemos como una actividad políticamente peligrosa, a la que con toda probabilidad se tendrá que minimizar, si no eliminar del todo en el mundo real. Trataremos esto nuevamente en el capítulo 17.
- El análisis estructurado clásico hacía una distinción difusa y poco definida entre los modelos físicos (que hacen suposiciones acerca de la tecnología de la implantación o están predispuestos por ésta) y los modelos lógicos (que son completamente independientes de la tecnología de implantación); de hecho, aun los términos lógico y físico confunden a muchos. [McMenamin y Palmer, 1984] contribuyeron con ideas importantes a esta área, e incluso ha cambiado parte de la terminología: ahora nos referimos a modelos esenciales (modelos de la "esencia" del sistema) en lugar de modelos lógicos, y a modelos de implantación en lugar de modelos físicos.
- Cada vez son más las organizaciones que están usando las técnicas del análisis estructurado para construir sistemas en *tiempo real.*[6] Sin embargo, el análisis estructurado clásico no tiene manera de modelar el comportamiento dependiente del tiempo de un sistema; carece de la notación necesaria para mostrar interrupciones y señales, y para mostrar la sincronización y coordinación de distintas labores de proceso. Para resolver este problema se ha añadido la notación necesaria y una herramienta nueva completa, que incluye flujos de control, procesos de control y diagramas de transición de estados. Esto se trata con más detalle en los capítulos 9 y 13.
- El análisis estructurado clásico se concentraba casi totalmente en modelar las *funciones* que deberían llevarse a cabo en un sistema; el modelado de *datos* se hacía de una manera primitiva[7] y a menudo se lo desenfatiza-

6 Recuerde la definición y ejemplos de sistemas de tiempo real del capítulo 2.

7 Esto tal vez sea un poco injusto, dado que [DeMarco, 1978] y [Gane y Sarson, 1977] dedican un capítulo o más a las estructuras de datos. Pero su notación ("diagramas de estructura de datos") ahora se considera obsoleta; además, la mayor parte del énfasis de los primeros textos era con respecto a la conversión de un conjunto arbitrario de estructuras de datos a la tercera forma normal, la cual es 1) bastante sencilla como para que el proceso se haya mecanizado y, 2) más bien una cuestión de implantación que de análisis de sistemas.

ba o incluso se lo ignoraba. Mientras tanto, más y más organizaciones han encontrado que sus sistemas comprenden funciones, relaciones entre datos y características de tiempo real, todas ellas complejas. Como hemos visto, los *diagramas de transición de estados* se han añadido al análisis estructurado para permitir el modelado de sistemas en tiempo real. Y para permitir el modelado de sistemas con relaciones complejas entre datos se introdujeron los *diagramas de entidad-relación.* E hecho de que las tres herramientas importantes puedan *integrarse,* es decir, que puedan utilizarse juntas de modo que cada una apoye a las demás, es más importante que el añadir una o dos herramientas adicionales de modelado. Los diagramas de entidad-relación se explican en el capítulo 12, y el concepto de los modelos integrados se trata en el capítulo 14.

- El proceso del análisis estructurado ha cambiado asombrosamente. El análisis estructurado clásico suponía que el analista empezaría por dibujar un diagrama de contexto, es decir, un diagrama de flujo de datos con una sola burbuja que representa a todo el sistema, y luego lo dividiría en varias funciones y almacenes de datos, en una forma estrictamente descendente. Por desgracia, esto no ha funcionado bien, por los motivos que se discutirán en el capítulo 20. En consecuencia, se ha añadido un nuevo enfoque, conocido como *división de eventos.* La terminología y el concepto básico de la división de eventos los introdujeron [McMenamin y Palmer, 1984] y los extendieron [Ward y Mellor, 1985].

7.3 EL SURGIMIENTO DE HERRAMIENTAS AUTOMATIZADAS DE ANALISIS

Cuando a finales de los años 70 y comienzos de los 80 se extendieron las técnicas de modelado gráfico del análisis estructurado en las organizaciones de desarrollo de sistemas, los analistas comenzaron a percatarse de que existía un gran problema: el trabajo necesario para crear diagramas de flujo de datos, diagramas de entidad-relación, diagramas de estructura, diagramas de transición de estados y otros modelos gráficos a menudo era abrumadora. El problema, en la mayoría de los casos, no era la creación inicial de los diagramas, sino su revisión y mantenimiento. Crear el diagrama inicial consume tiempo, pero por lo menos existe la satisfacción de que es una actividad intelectual creativa y desafiante. En un proyecto típico, el analista encuentra que tiene que volver a dibujar los modelos gráficos varias veces antes de que él y los usuarios puedan llegar a algún acuerdo sobre los requerimientos del sistema.[8]

En un sistema muy grande puede haber 50, 100 o más diagramas de flujo de datos, diversos diagramas de entidad-relación y, potencialmente, varios diagramas

8 Esto pudiera no serle evidente aún, pues sólo hemos visto unos cuantos diagramas de análisis estructurado en el capítulo 4. Sin embargo, para el final de la parte II debiera ser abundantemente evidente; si no, espere a ver el final de su primer proyecto "real" de análisis estructurado.

de transición de estados; así que la cantidad de trabajo manual puede ser en verdad intimidante. La consecuencia práctica de esto, en muchas organizaciones, es que el análisis estructurado clásico no fue tan exitoso como debiera ser. Se plantearon los siguientes problemas:

- Tras la segunda y tercera correcciones de un diagrama, el analista se volvía cada vez más opuesto y renuente a hacer más cambios. Por ello, se podían encontrar diagramas "congelados" que no reflejaban los verdaderos requerimientos del usuario.

- Debido a la cantidad de trabajo requerido, el analista dejaba a veces de dividir el modelo del sistema en modelos de menor nivel; es decir, en lugar de desarrollar un modelo que consistiera por ejemplo, en cinco niveles de diagramas de flujo, se detenía en el cuarto nivel. El modelo resultante contenía funciones "primitivas" (esto es, las burbujas que se muestran en el cuarto nivel) que no eran primitivas en lo más mínimo; de hecho, resultaban ser tan complejas que el programador tenía la necesidad de llevar a cabo un análisis adicional del sistema antes de que pudiera escribir programas.[9]

- A menudo no se incorporaban en el modelo del sistema los cambios en los requerimientos del usuario sino hasta *después* de la fase de análisis del proyecto. Muchos de estos cambios se daban durante las fases de diseño, programación y prueba; otros se daban después de implantado el sistema. El resultado era una especificación obsoleta.

Además del trabajo que se necesita para crear y mantener los diagramas, el análisis estructurado clásico requiere de una gran cantidad de trabajo para *verificar* los diagramas con el fin de asegurar que sean consistentes y estén completos; estas reglas se discuten en el capítulo 14.[10] Durante los años 70 y la mayor parte de los 80, los analistas tuvieron que depender de técnicas de verificación manual (es decir, inspeccionar visualmente los diagramas para encontrar errores). Debido a que esta labor es detallada y aburrida, tiende a estar plagada de errores. En consecuencia, no se encontraban muchos de los errores de especificación que se debieran haber encontrado.

9 Como se verá en el capítulo 11, debiera haber una especificación del proceso (normalmente escrita como tabla de decisiones, diagrama de flujo o en un formato estructurado en español) para cada burbuja primitiva de último nivel del diagrama de flujo de datos. Si el sistema se ha dividido correctamente, la mayoría de las especificaciones de proceso deberían ser de menos de una página.

10 Ejemplos de reglas de verificación: todos los flujos de datos en un diagrama de flujo de datos deben tener nombre, y los nombres deben estar definidos en el diccionario de datos. Todos los nombres en el diccionario de datos deben corresponder con flujos de datos o almacenes en el diagrama. Cada burbuja en el diagrama debe tener por lo menos un flujo de datos que entra y uno que sale, etcétera.

Muchos de estos problemas se pueden resolver con apoyo automatizado adecuado. Esto era bien conocido aun desde que por primera vez se introdujo el análisis estructurado clásico, pero el costo de la automatización estaba por encima de las posibilidades económicas de la mayoría de las organizaciones. Sin embargo, el desarrollo de poderosas estaciones de trabajo gráficas a mediados de los años 80 llevó a una nueva industria llamada CASE (siglas que significan *Computer-Aided Software Engineering:* ingeniería de software auxiliada por computadora). Varios proveedores ofrecen productos (normalmente basados en PC) que dibujan diagramas de flujo de datos y otros, además de llevar a cabo una variedad de labores de revisión de errores. Las propiedades y ejemplos de estas herramientas se presentan en el apéndice A.

Como se mencionó anteriormente, sólo el 2 por ciento de los analistas de sistemas en los Estados Unidos tenían acceso a estas herramientas en 1987, y se estima que sólo el 10 por ciento lo tendrán en 1990. Sin embargo, éste es indiscutiblemente el camino del futuro, y podemos esperar que todos los analistas profesionales insistirán en tales herramientas al transcurrir el tiempo. Esto llevará primordialmente a un nivel más elevado de calidad en los modelos de sistemas producidos. También, de manera secundaria, llevará a modelos gráficos del sistema visualmente más atractivos. En la medida en que los conceptos de revisión de errores que se discutieron en el capítulo 14 sean automatizados, podrán eliminar la necesidad de que los analistas aprendan el material del capítulo 14. Y en la medida en que las herramientas CASE comiencen a generar programas (en COBOL, Pascal, o tal vez en un lenguaje de cuarta generación) *directamente* desde las especificaciones, también incluso se reducirá la necesidad de programadores.

7.4 EL USO DE PROTOTIPOS

Como se señaló en el capítulo 3, algunos usuarios tienen dificultades al tratar con los modelos gráficos del análisis estructurado y prefieren alguna otra forma de modelar los requerimientos y comportamiento del sistema. Las herramientas de generación de prototipos, que empezaron a ser muy accesibles a mediados de los años 80, se han considerado como una alternativa al análisis estructurado para tales usuarios.

Existe otra razón de la popularidad de los prototipos: en muchas organizaciones se considera que el análisis estructurado clásico consume demasiado tiempo; para cuando concluye la fase de análisis, el usuario habrá olvidado para qué quería en un principio el sistema. Esto suele ser resultado de alguno de los siguientes problemas:

- El equipo encargado del proyecto tardó demasiado en desarrollar modelos del sistema *actual* y luego se tuvo que tardar aún más modelando el nuevo. Como se mencionó anteriormente, ahora consideramos el modelado del sistema actual como una actividad políticamente peligrosa.

- La organización invirtió previamente poco o nada de tiempo, en hacer análisis, pues prefirió codificar lo antes posible. En tal ambiente, el largo trabajo de análisis, que aparentemente no produce nada fuera de muchas imágenes con círculos y rectángulos, pudiera parecerles improductivo.

- Los primeros proyectos que se han realizado utilizando el análisis estructurado pudieran consumir más tiempo de lo normal, pues los analistas están aprendiendo nuevas técnicas y discutiendo respecto a la mejor forma de aplicar dichas técnicas.

Las herramientas de generación de prototipos (herramientas de software que permiten al analista construir un simulacro del sistema) se ven, por ello, como una solución efectiva a estos problemas. Nótese también que el uso de prototipos tiende a concentrarse en el aspecto de la interfaz humana en los proyectos de desarrollo de sistemas.

Desafortunadamente, las herramientas de generación de prototipos a veces se utilizan para evitar los detalles del análisis y del diseño. En la sección 5.6 se mostró un uso apropiado de prototipos.

7.5 EL MATRIMONIO DEL ANALISIS Y EL DISEÑO DE SISTEMAS

Como se mencionó anteriormente en este capítulo, las mejorías en la ingeniería de software comenzaron con la programación y el diseño estructurados. De hecho, estos dos temas estuvieron sujetos a un considerable debate en las organizaciones de desarrollo de sistemas durante la primera mitad de los años 70. También fue durante este periodo que empezaron a aparecer los primeros textos sobre diseño estructurado (véase [Myers, 1975] y [Yourdon y Constantine, 1975]) Los primeros libros no hacían referencia al análisis estructurado (puesto que aún no se habían desarrollado los conceptos), mientras que libros posteriores, como [Page-Jones, 1980], incluían una breve reseña del tema. El trabajo en análisis estructurado comenzó a mediados de los 70 y los primeros textos empezaron a aparecer a fines de esa década; *pero había poca o no había conexión entre el discurso del análisis estructurado y el del diseño estructurado.* El principal problema era que el análisis estructurado trataba con la especificación de sistemas grandes y complejos, mientras que el diseño estructurado parecía ser más apropiado para el diseño de programas individuales que se ejecutaban en una misma computadora. El puente entre el análisis del sistema y el diseño de los programas, es decir, hacía falta el *diseño de sistemas.*

Este problema lo han tratado muchos consultores, autores y organizaciones de desarrollo de sistemas durante los años 80. Libros recientes, como el de [Ward y Mellor, 1985], así como las ediciones nuevas de [Page-Jones, 1988] y [Yourdon y Constantine, 1989], tratan ahora puntos del diseño de sistemas al igual que de programas.

7.6 RESUMEN

Como cualquier campo de la ciencia o la ingeniería, el análisis de sistemas ha pasado por una serie de cambios evolucionarios durante los últimos 20 años. Como se indicó al comienzo de este capítulo, es importante saber cuáles han sido estos cambios porque la industria de la computación es lo suficientemente amplia como para que no todo mundo practique las mismas técnicas al mismo tiempo. La organización de usted pudiera estar a la vanguardia de la tecnolcgía o al triste final de la cola.

Se puede esperar que el campo del análisis de sistemas continúe. Las técnicas que se presentan en este libro habrán evolucionaco aún más en en los próximos cinco a diez años. En el capítulo 25 se aborda la probable naturaleza de dichos cambios evolucionarios.

REFERENCIAS

1. Tom DeMarco, *Structured Analysis and System Specification.* Nueva York: YOURDON Press, 1978.
2. Chris Gane y Trish Sarson, *Structured Systems Analysis and Design.* Nueva York: Improved Systems Technologies, Inc., 1977.
3. Victor Weinberg, *Structured Analysis.* Nueva York: YOURDON Press, 1978.
4. Paul Ward y Steve Mellor, *Structured Development for Real-Time Systems.* Volúmenes 1-3. Nueva York: YOURDON Press, 1985.
5. Steve McMenamin y John Palmer, *Essential Systems Analysis.* Nueva York: YOURDON Press, 1984.
6. Glen Myers, *Reliable Systems through Composite Design,* 1ª edición. Nueva York: Petrocelli/ Charter, 1975.
7. Edward Yourdon y Larry Constantine, *Structured Design,* 1ª edición. Nueva York: YOURDON Press, 1975.
8. Meilir Page-Jones, *The Practical Guide to Structured Systems Design,* 1ª edición, Nueva York: YOURDON Press, 1980.
9. Meilir Page-Jones, *The practical Guide to Structured Systems Design,* 2ª edición, Englewood Cliffs, N.J.: Prentice-Hall, 1988.
10. Edward Yourdon y Larry Constantine, *Structured Design,* 2ª edición, Englewood Cliffs, N.J.: Prentice-Hall, 1989.

PREGUNTAS Y EJERCICIOS

1. ¿Cuáles son las tres principales razones por las cuales debe estar familiarizado con la evolución del análisis de sistemas?
2. ¿Qué cree que deba hacer si la organización para la que trabaja no ha llevado a cabo los cambios que se exponen en este capítulo?
3. Mencione cuatro problemas importantes de una especificación narrativa clásica.
4. ¿Por qué no es deseable la redundancia en una especificación de sistemas? ¿Es posible eliminar por completo la redundancia de la especificación?
5. ¿Se le ocurre alguna razón por la cual sería útil la redundancia en la especificación de un sistema?
6. ¿Cuáles son las tres principales razones por las que aparece redundancia en una especificación clásica?
7. ¿Qué porcentaje de errores en un sistema operacional pueden atribuirse a errores que ocurrieron durante la fase de análisis del proyecto?
8. Proyecto de investigación: ¿Qué porcentaje de errores puede atribuirse en *su* organización a errores que ocurrieron durante la fase de análisis del proyecto?
9. ¿Cuáles son las consecuencias de una especificación imposible de mantener?
10. Describa brevemente la programación estructurada.
11. Describa brevemente el diseño estructurado.
12. ¿Por qué algunas organizaciones no tuvieron éxito al utilizar la programación y el diseño estructurados?
13. ¿Cuáles son las tres principales características de una especificación estructurada?
14. ¿Cuáles son los cinco principales cambios que se han dado en el análisis estructurado clásico?
15. ¿Qué problemas enfrenta el análisis estructurado clásico cuando trata con sistemas de tiempo real?
16. ¿Cuáles son los peligros asociados con modelar el sistema actual del usuario? ¿Cuánto tiempo se debiera dedicar a esto?
17. ¿Cuáles son los tres principales problemas a los que es probable que el analista se enfrente si no tiene apoyo automatizado para su trabajo?

18. ¿Es importante contar con apoyo automatizado para proyectos pequeños de desarrollo de sistemas de información? ¿Por qué sí o por qué no?

19. ¿Qué problemas es probable encontrar si el analista tiene que llevar a cabo manualmente la revisión de errores?

20 ¿Por qué cree que tan sólo el 2 por ciento de los analistas de sistemas de los Estados Unidos tenían estaciones de trabajo automatizadas de análisis de sistemas en 1987?

21. ¿Qué beneficios adicionales podemos esperar al introducir una red de herramientas automatizadas de análisis de sistemas? (Por ejemplo, uno de estos beneficios es el correo electrónico.)

22. ¿Cuáles son los tres principales problemas que han surgido en las organizaciones al implantar el análisis estructurado clásico?

23. ¿Qué problemas de interfaz existían entre el análisis estructurado y el diseño estructurado en los años 70 y principios de los 80?

PARTE II: LAS HERRAMIENTAS DE MODELADO

8 CARACTERISTICAS DE LAS HERRAMIENTAS DE MODELADO

Cualquier cosa es fácil si puede asimilarla a su colección de modelos
Seymour Papert, *Mindstorms*

En este capítulo se aprenderá:

1. Por qué suelen ser gráficas las herramientas de modelado de sistemas.
2. Por qué se pueden segmentar en forma descendente las herramientas de modelado de sistemas.
3. Por qué tienen redundancia mínima las herramientas de modelado de sistemas.
4. Por qué ayudan las herramientas de modelado de sistemas a modelar el comportamiento de un sistema.

Los siguientes capítulos de este libro describen las diversas herramientas de modelado que usted usará como analista. Antes de ahondar en los detalles de los diagramas de flujo de datos, de entidad-relación, etc, existen algunos puntos introductorios que necesitamos repasar.

Se recordará del capítulo 4 que un modelo es un simulacro a bajo costo de un sistema complejo que se desea estudiar. Se construyen modelos de sistemas por tres motivos:

1. Para enfocar características importantes del sistema, a la vez que para minimizar las características menos importantes.
2. Para discutir cambios y correcciones a los requerimientos del usuario, a bajo costo y con riesgo mínimo.
3. Para verificar que se entiende el ambiente del usuario, y que se ha documentado de tal manera que los diseñadores y programadores puedan construir el sistema.

Sin embargo existen muchos tipos diferentes de modelos que se pueden construir para el usuario: modelos narrativos, modelos de prototipos, modelos gráficos diversos, etc. De hecho, el sistema final que se le construirá al usuario pudiera resultar ser un modelo, en el sentido de que puede representar, por primera vez, una manera de que el usuario visualice lo que desea.

En este libro nos concentramos en los modelos en *papel* (o en modelos en papel producidos por sistemas CASE automatizados). Pero, nuevamente, existe una gran variedad. Como se apreciará con más detalle en los siguientes capítulos, existen diagramas de flujo, diagramas HIPO, tablas de decisión, diagramas de flujo de datos, diagramas de flujo de sistemas, diagramas de transición de estados, árboles de decisiones, diagramas de entidad-relación, diagramas de Fersi, diagramas de Hamilton-Zeldin, diagramas PAD y una interminable serie de diagramas, tablas y gráficas que se le pueden presentar al usuario. ¿Cuál debemos usar?

La premisa básica de este libro es que debe usarse *cualquier* modelo que funcione en la situación en la que se encuentra. Los diferentes usuarios pudieran requerir distintas herramientas de modelado, sea por su experiencia pasada o porque ciertos tipos de diagramas los confunden o intimidan.[1] Diferentes proyectos pudieran requerir de distintas herramientas para cumplir con los estándares de documentación impuestos por organizaciones externas. Y diferentes tipos de sistemas

1 Un corolario de esto es que las buenas herramientas de modelado suelen emplear una notación sencilla, con pocas reglas, símbolos y vocabulario nuevo que el usuario tenga que aprender. Un purista pudiera argumentar incluso que una buena herramienta *no* requiere explicación ni preparación. De cualquier modo, no debería ser necesario leer un libro de 700 páginas como éste para poder aprender a leer y entender un modelo desarrollado por el analista.

pudieran requerir de modelos diversos para poder destacar adecuadamente características importantes.

Para llevar más lejos esto, la mayoría de los sistemas requieren de *múltiples* modelos: cada modelo se enfoca a un número limitado de aspectos del sistema, a la vez que minimiza (o ignora totalmente) otros de sus aspectos. Esto se da sobre todo en muchos de los sistemas que se están construyendo actualmente, pues tienen características funcionales complejas, estructuras de datos complejas, y consideraciones complejas de *tiempos*.

Cualquier herramienta que use debiera tener las siguientes características:

- Debe ser gráfica, con detalles textuales de apoyo apropiados.
- Debe permitir que el sistema sea visto en segmentos, en forma descendente.
- Debe tener redundancia mínima.
- Debe ayudar al lector a predecir el comportamiento del sistema.
- Debe ser transparente para el lector.

A continuación discutiremos con más detalle cada uno de estos puntos.

8.1 MODELOS GRAFICOS

La mayoría de los modelos populares de sistemas, y todos los que se presentan en este libro, se apoyan mucho en gráficas. *No es requisito* usar gráficas en un modelo de sistemas, pero el viejo adagio de que "una imagen vale más que mil palabras" es una buena explicación de nuestra preferencia por las gráficas en lugar de texto narrativo. Una imagen bien escogida puede transmitir de manera concisa y compacta una gran cantidad de información.

Esto no significa necesariamente que una imagen pueda describir *todo* lo referente a un sistema; hacerlo normalmente significaría tener un desorden que nadie quisiera mirar. En general, se utilizan los gráficos para identificar los *componentes* de un sistema y su *interfaz*. Todos los demás detalles (éste es, las respuestas a preguntas tales como "¿Cuántos?" y "¿En qué orden?", etc.) se presentan en documentos textuales de apoyo. Los textos de apoyo que se describen en este libro son la *especificación del proceso* y el *diccionario de datos*.

Esto no significa que todos los analistas deban usar el conjunto particular de herramientas gráficas y de textos que se presentan en este libro; el Gran Analista de Sistemas que está en el cielo no lo fulminará con sus rayos si no utiliza diagramas de flujo de datos. Sin embargo, es probable que lo fulmine el rayo si opta por uno de los extremos de *únicamente* gráficos (sin textos de apoyo) o de *sólo* texto (sin material gráfico). Y seguramente le tocaría aunque sea un pequeño relámpago si hiciera

que el texto fuera la parte dominante del modelo, y que los gráficos tuvieran un papel pequeño y subordinado. Uno o más gráficos debieran ser el documento primario al que se dirige el usuario para poder entender el sistema; los documentos textuales debieran servir de material de referencia para consulta en caso de necesidad.

8.2 MODELOS SEGMENTABLES EN FORMA DESCENDENTE

Un segundo aspecto importante de una buena herramienta de modelado es su capacidad de mostrar un sistema por partes en forma descendente. Esto no es importante para sistemas pequeños, pues de ellos se puede decir todo lo necesario en una o dos páginas, y cualquiera que necesite conocer algún aspecto del sistema bien puede conocerlo en su totalidad.

Sin embargo, los proyectos reales en el mundo real generalmente no son pequeños.[2] De hecho, muchos de los proyectos en los que es probable que se involucre serán desde medianos hasta enormes. En consecuencia, será imposible que alguien, sea usuario, analista c programador, se enfoque a todo el sistema al mismo tiempo. Tampoco será posible presentar un modelo gráfico de un sistema grande y complejo en una sola hoja de papel, a menos, claro, que se considere el extremo de tener una microficha de 2 x 3 metros. Por eso, nuestras herramientas deben permitirnos mostrar partes individuales del sistema de manera independiente, junto con *una forma sencilla de moverse de una parte a otra del modelo del sistema.*

Sin embargo, se necesita más que esto: no sería apropiado, por ejemplo, crear un modelo gráfico enorme de 30 x 30 metros y luego cortarlo en 3,600 pedazos individuales de medio metro cuadrado, con miles de conectores de hoja a hoja. Esto equivaldría, a grandes rasgos, a dibujar un mapa de las calles de todo un país en una sola hoja y luego partirla en miles de pedacitos del tamaño de una página.

De hecho, nuestra experiencia con mapas y atlas ilustra cómo debe organizarse un modelo de un sistema complejo. Por ejemplo, un atlas de los Estados Unidos empieza generalmente por un solo diagrama de una página de todo el país, como se muestra en la figura 8.1. Dicha página muestra los estados individuales, y pudiera también mostrar las principales interfaces entre los estados (ríos, autopistas interestatales, líneas de ferrocarril, rutas aéreas, etc.). Las siguientes páginas se dedican normalmente a cada estado individual, en donde cada página muestra las ciudades y condados individuales del estado, al igual que las autopistas locales que no aparecen en el mapa de nivel nacional. Podriamos imaginar mapas de nivel menor que

2 O, dicho de otro modo, los usuarios están desarrollando cada vez más proyectos "pequeños" sin necesidad de analistas o programadores. Con el gran acceso a computadoras personales, paquetes de hojas de cálculo y lenguajes de cuarta generación, muchos trabajos que hubieran requerido días (o incluso semanas) de dedicación de un profesional de las computadoras el usuario puede hacerlos actualmente en cuestión de minutos o de horas. Sin embargo, aún continúan desarrollándose muchos sistemas que requieren de más de 10 *millones* de instrucciones para llevar a cabo su propósito.

proporcionarían detalles de cada condado, de cada ciudad dentro de un condado determinado, y de cada barrio dentro de una ciudad dada.

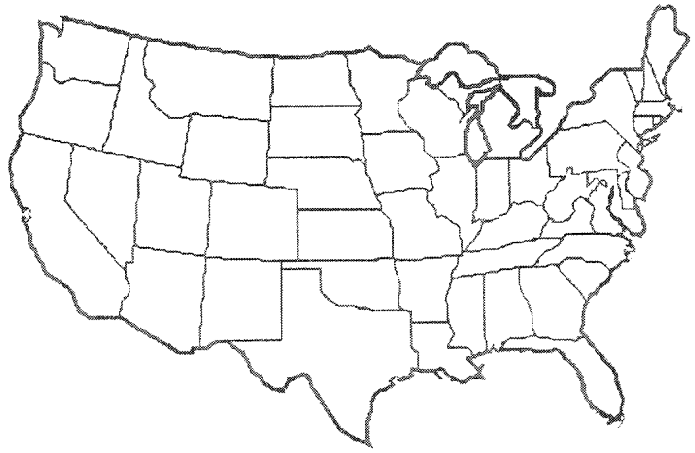


Figura 8.1: **Mapa de Estados Unidos**

Un buen modelo de un sistema complejo de información debería proceder de la misma manera descendente. Una porción de la vista global de alto nivel del modelo debe dar al lector buena idea de los principales componentes de alto nivel y de las interfaces del sistema. Las siguientes porciones del modelo deberían proporcionar información respecto a los componentes detallados de bajo nivel. Y así como el atlas proporciona un mecanismo conveniente para recorrer el conjunto completo de mapas individuales (ésto es, llegar sin mayor confusión del mapa de nivel nacional al mapa apropiado al nivel de condado), un buen modelo de un sistema de información proporciona un mecanismo conveniente para pasar tranquilamente de un nivel alto a uno bajo.

8.3 MODELOS MINIMAMENTE REDUNDANTES

Los modelos son representación de algún sistema del mundo real y el sistema mismo pudiera ser estático (no cambiante) o dinámico. Un mapa de Estados Unidos, por ejemplo, es una representación gráfica del país. Mientras que muchos de sus aspectos son obviamente muy dinámicos, podría decirse que los aspectos que modela un mapa son relativamente estáticos: los estados individuales no aparecen o desaparecen muy a menudo y las fronteras entre ellos han permanecido constantes durante un tiempo bastante largo. (En cambio, no puede decirse esto de un mapa de todo el mundo).

¿Por qué importa esto a quien construye un modelo? Simplemente, porque es deseable *mantenerlo* en forma actualizada y precisa. Si el sistema cambia, entonces debe cambiar el modelo, si ha de tenerse actualizado. Obviamente, si sólo cambia un aspecto local del sistema, preferiríamos cambiar sólo un aspecto local correspondiente del modelo, sin tener por fuerza que cambiar algún otro. De hecho, si se *requieren* múltiples cambios, existe una buena probabilidad de que no se hagan o de que se harán desordenadamente. Y esto significa que el modelo se volverá gradualmente menos preciso.

Para ilustrar esto, considere nuevamente nuestro ejemplo del atlas de los Estados Unidos . Podemos imaginar, en el caso más simple, una página que muestre todo el país, y 50 páginas siguientes que muestren los detalles de cada estado. Ahora, imagine qué sucedería si desapareciera el estado de Nueva Jersey:[3] el mapa de nivel nacional tendría que volver a dibujarse para mostrar el nuevo país de 49 estados, y el anterior mapa de nivel estatal de Nueva Jersey tendría que descartarse.

Sin embargo sería un poco más difícil con los atlas de verdad, pues, como es característico de muchos modelos de sistemas, existe alguna redundancia. Cada mapa de nivel estatal muestra no sólo el estado que se describe sino también parte de los que tienen frontera con él. Esta información se tiene en el mapa de nivel nacional, pero es útil en el nivel estatal también. Esto significa que si desapareciera Nueva Jersey, probablemente tendríamos que redibujar los mapas de Nueva York y Pennsylvania, e incluso tal vez Delaware y Maryland. Qué molestia.

Los cartógrafos profesionales pudieran objetar esto y argumentar que se necesita una cierta cantidad de redundancia para hacer el atlas fácil de leer. Pero debería ser evidente que cuanto más redundante sea el modelo más difícil será de mantener. Imagine, por ejemplo, que nuestro atlas mítico muestra las autopistas interestatales en el mapa nacional y en todos los mapas de nivel estatal. Imagine también que algún emprendedor fabricante de mapas ha decidido mostrar la longitud completa de cada autopista interestatal *en cada mapa estatal que atraviesa*. De esta forma, la carretera interestatal 95, que va de Maine a Florida, aparecería en alrededor de una docena de estados, y en cada uno se escribiría el hecho (redundante) de que la autopista mide 2,700 kilómetros. Y ¿ahora qué sucede si descubrimos que esta cifra estaba equivocada o que parte de la autopista se extendió o se desvió de ruta? Obviamente, se tendrían que modificar una docena de mapas estatales.

8.4 MODELOS TRANSPARENTES

Finalmente, un buen modelo debe ser tan fácil de leer que el lector no tenga que detenerse a pensar siquiera que se trata de la *representación* de un sistema y no del sistema mismo. Esto no siempre es fácil de lograr y a menudo requiere de práctica y preparación por parte del lector. Piense por ejemplo en un mapa: ¿qué tan

3 Si vivió en Nueva Jersey o tiene alguna otra conexión patológica con este estado, siéntase libre de usar cualquier otro para este ejemplo. Mis disculpas a Bruce Springsteen.

a menudo se pone a pensar en que está mirando una representación abstracta del estado de Nueva Jersey y no la realidad misma? Por otro lado, observe a un niño pequeño que esté viendo un mapa mientras sus padres o su maestro pretenden explicarle que el estado de Nueva Jersey tiene frontera con el estado de Nueva York y que Newark está a quince kilómetros de la ciudad de Nueva York. "No, no es cierto", dirá el niño, "Newark está a dos centímetros de Nueva York".

Al crecer, nos familiarizamos cada vez más con el concepto de las representaciones abstractas, *siempre que nos parezcan cómodas menialmente.* Los científicos han estudiado el comportamiento y la organización del cerebro humano y han encontrado que el hemisferio izquierdo realiza los procesos secuenciales. También se hace cargo de los textos; por ejemplo, las palabras que está leyendo, una tras otra, en esta página. El hemisferio derecho trata con las imágenes y el procesamiento asincrónico, donde "todo sucede a la vez". Esto indica que si estamos tratando de modelar algo que es intrínsecamente lineal y secuencial, como el flujo de control en un programa de computadora, debemos usar una herramienta de modelado textual que quepa cómodamente en el hemisferio izquierdo, que será el mejor equipado para tratarlo. Y si estamos tratando de modelar algo que es intrínsecamente multidimensional, con muchas actividades que se dan a la vez, debemos usar una herramienta gráfica.

8.5 RESUMEN

Sin duda estará tan ocupado aprendiendo las herramientas de modelado que se presentan en este libro que no pensará en la posibilidad de otras. Sin embargo, si existen, y en el capítulo 15 examinaremos brevemente varias otras.

Más importante aún, como analista se verá ante una variedad de herramientas de modelado en proyectos del mundo real. Aunque los detalles (y formas) de estas herramientas varían mucho, verifique con cuidado si siguen los principios básicos que se presentan en este capítulo.

PREGUNTAS Y EJERCICIOS

1. ¿Cuáles son las tres principales razones por las que se hacen modelos de un sistema?
2. Describa tres tipos diferentes de modelos de sistemas.
3. ¿Cuáles son las características principales que debe tener una herramienta de modelado de un sistema?
4. ¿Por qué se prefieren generalmente las herramientas gráficas a las textuales?
5. ¿Es *necesario* usar herramientas de modelado gráfico para desarrollar un sistema de información? ¿Se le ocurren situaciones donde no quisiera usar dichas herramientas?

6. ¿Qué cosas normalmente *no* muestran los modelos gráficos acerca de un sistema?

7. ¿Por qué es importante que una herramienta de modelado muestre el sistema de manera descendente? ¿Existen situaciones donde no importe?

8. ¿Requiere la *descripción* descendente de un sistema que éste se diseñe de manera descendente?

9. Describa una situación en la que debiera ser aceptable incluir redundancia en el modelo del sistema.

9 DIAGRAMAS DE FLUJO DE DATOS

La forma siempre sigue a la función.
Louis Henri Sullivan
"El gran edificio de oficinas, considerado artísticamente",
Lippincott's Magazine, marzo de 1896

En este capítulo se aprenderá:

1. Los componentes de un diagrama de flujo de datos.
2. Cómo dibujar un diagrama de flujo de datos sencillo.
3. Guía para dibujar diagramas eficaces de flujo de datos.
4. Cómo volver a dibujar diagramas *nivelados* de flujo de datos.

En este capítulo exploraremos una de las tres herramientas gráficas de modelado más importantes del análisis estructurado: el *diagrama de flujo de datos*. Esta es una herramienta que permite visualizar un sistema como una red de procesos funcionales, conectados entre sí por "conductos" y "tanques de almacenamiento" de datos. En la literatura computacional, y en sus conversaciones con otros analistas y usuarios, puede utilizar cualquiera de los siguientes términos como sinónimos de diagrama de flujo de datos:

- Carta de burbujas
- DFD (Abreviatura que se usará en todo este libro)
- Diagrama de burbujas
- Modelo de proceso
- Diagrama de flujo de trabajo
- Modelo de función
- "una imagen de lo que está sucediendo aquí"

El diagrama de flujo de datos es una de las herramientas más comúnmente usadas, sobre todo por sistemas operacionales en los cuales *las funciones* del sistema son de gran importancia y son más complejas que los datos que éste maneja. Los DFD se utilizaron por primera vez en la ingeniería de software como notación para el estudio del diseño de sistemas (por ejemplo, en los libros y artículos de diseño estructurado tales como [Stevens, Myers y Constantine, 1974], [Yourdon y Constantine, 1975], [Myers, 1975], y otros). A su vez, la notación se tomó prestada de artículos anteriores sobre teoría de gráficas, y continúa siendo utilizada por los ingenieros de software que trabajan en la implantación directa de modelos de los requerimientos del usuario.

Estos son antecedentes interesantes, pero con toda probabilidad no serán muy relevantes para los usuarios a quienes usted mostrará los modelos de DFD del sistema; de hecho, probablemente lo *peor* que pueda usted hacer sea decir, "Sr. Usuario, quisiera mostrarle un modelo gráfico-teórico descendente y por partes de su sistema". En realidad, muchos usuarios estarán familiarizados con el concepto básico de DFD, pues la misma notación ha sido empleada por investigadores de operaciones durante los últimos 70 años para construir modelos de flujo de trabajo de organizaciones. Es importante tener esto en mente: los DFD no sólo se pueden utilizar para modelar sistemas de sistemas de proceso de información, sino también como manera de modelar organizaciones enteras, es decir, como una herramienta para la planeación estratégica y de negocios.

Empezaremos nuestro estudio de los DFD examinando los componentes de un diagrama típico de flujo de datos: el proceso, el flujo, el almacén y el terminador. Utilizaremos una notación bastante común, siguiendo los textos clásicos de [DeMarco, 1978], [Gane y Sarson, 1977], y otros. Sin embargo, también incluiremos la notación de DFD para modelar sistemas de tiempo real (es decir, flujos de control y procesos de control). Esta notación adicional generalmente no se ocupa en los sistemas dirigidos a los negocios, pero es crucial cuando se modela una variedad de sistemas científicos y de ingeniería.

En seguida, revisaremos algunas de las guías de elaboración de diagramas de flujo de datos para minimizar las posibilidades de construir un DFD confuso, incorrecto o inconsistente. Finalmente, discutiremos el concepto de DFD *nivelado* como método de modelar sistemas complejos.

Tenga en mente que el DFD es tan sólo una de las herramientas de modelado disponibles y que únicamente proporciona un punto de vista de un sistema, el orientado a las funciones. Si se está desarrollando un sistema en donde las relaciones entre datos son más importantes que las funciones, tal vez se dé menos importancia al DFD (o incluso ni nos molestemos en elaborarlo), para concentrarse más bien en desarrollar un conjunto de diagramas de entidad-relación, como se expone en el capítulo 12. De otra manera, si el comportamiento dependiente del tiempo de un sistema domina sobre cualquier otro factor, tal vez nos concentremos más en el diagrama de transición de estados que se discute en el capítulo 13.

9.1 LOS COMPONENTES DE UN DFD

La figura 9.1 muestra un DFD típico para un sistema pequeño. Antes de examinar sus componentes en detalle, nótese lo siguiente:

- Prácticamente no requiere explicación; se puede simplemente mirar el diagrama y entenderlo. La notación es sencilla y clara y, en cierto sentido, intuitivamente obvia. Esto es particularmente importante cuando recordamos quién se supone que está viendo la figura 9.1: no el analista, sino el usuario. Si el usuario necesita una enciclopedia para poder leer y entender el modelo de su sistema, probablemente no se tomará la molestia de hacer ninguna de las dos cosas.
- El diagrama cabe fácilmente en una página. Esto significa dos cosas: 1) alguien puede mirarlo sin ofuscarse y 2) el sistema que se está modelando no es muy complejo. ¿Qué se hace si el sistema es intrínsecamente complejo, tanto –por ejemplo– que hubiera literalmente cientos de círculos y líneas en el diagrama? Discutiremos esto en la sección 9.4.
- El diagrama se dibujó con computadora. Nada tiene de malo un diagrama hecho a mano, pero la figura 9.1 y muchos de los otros DFD que se muestran en este libro se hicieron con la ayuda de un programa de la Macintosh llamado MacDraw. Esto significa que el diagrama probablemente se hará de manera más ordenada y estándar que lo que en general sería posible a mano. También significa que se pueden hacer cambios y producir nuevas versiones en cuestión de minutos.[1]

1 Sin embargo, la desventaja de MacDraw (y de otros programas genéricos por el estilo) es que no sabe nada de la naturaleza especial de los DFD o de otros modelos de sistemas que se presentan en este libro. Sólo maneja figuras primitivas como rectángulos, círculos y líneas. Los paquetes de herramientas para analistas que se discuten en el Apéndice A son más poderosos, pues manejan mucho acerca de DFD.

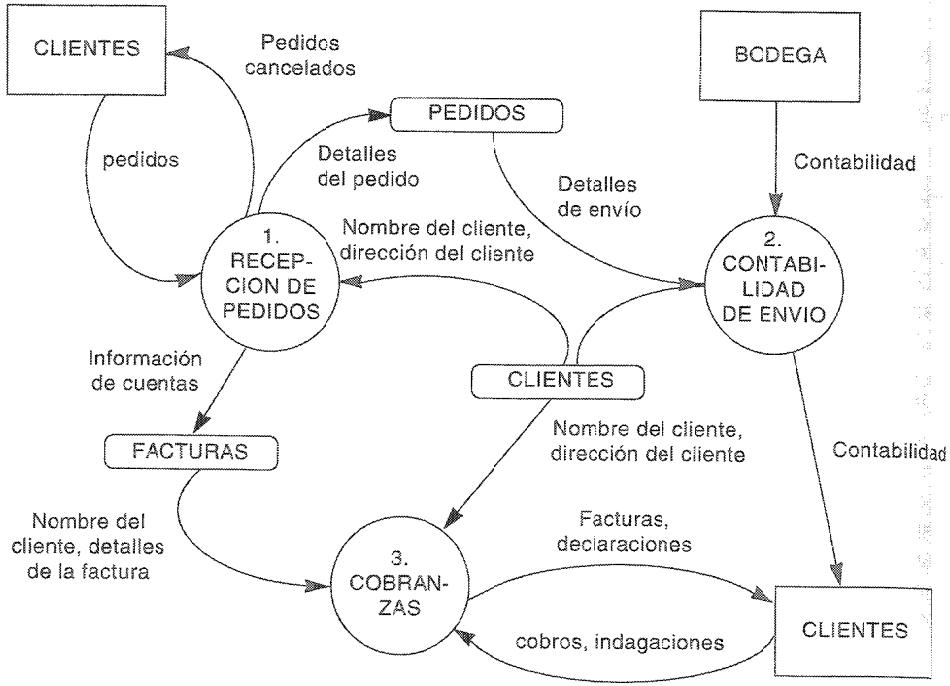


Figura 9.1: **DFD típico**

9.1.1 El proceso

El primer componente del DFD se conoce como *proceso*. Los sinónimos comúnes son *burbuja, función* o *transformación*. El proceso muestra una parte del sistema que transforma entradas en salidas; es decir, muestra cómo es que una o más entradas se transforman en salidas. El proceso se representa gráficamente como un círculo, como se muestra en la figura 9.2(a). Algunos analistas prefieren usar un óvalo o un rectángulo con esquinas redondeadas, como se muestra en la figura 9.2(b); y otros prefieren usar un rectángulo, como se muestra en la figura 9.2(c). Las diferencias entre estas tres formas son puramente cosméticas, aunque obviamente es importante usar la misma forma de manera consistente para representar todas las funciones de un sistema. A lo largo de este libro utilizaremos el círculo o burbuja.[2]

2 La figura que el analista utilice para el proceso a menudo se asocia con una "escuela" particular de análisis estructurado. El círculo se asocia con la "escuela Yourdon/DeMarco", pues se utiliza en varios textos publicados por YOURDON Press, lo mismo que en las actividades de consulta y adiestramiento de YOURDON Inc. De manera similar, el óvalo se asocia a menudo con la "escuela Gane/Sarson", pues lo introdujeron Chris Gane y Trish Sarson en su libro [Gane y Sarson, 1977] y fue usado por McDonnell Douglas Automation Company (McAuto) y varias organizaciones más. La figura rectangular suele asociarse con la escuela "SADT", pues la popularizaron diversos artículos acerca de la técnica de Softech para Diseño y Análisis Estructurado (SADT); véase, por ejemplo, [Ross y Schoman, 1977].

Figura 9.2(a): **Ejemplo de un proceso**

Figura 9.2(b): **Representación alternativa de un proceso**

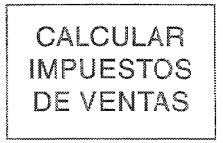


Figura 9.2(c): **Una representación más de un proceso**

Nótese que el proceso se nombra o describe con una sola palabra, frase u oración sencilla. En casi todos los DFD que se discutirán en este libro, el nombre del proceso describirá *lo que hace*. En la sección 9.2 hablaremos más acerca de la nomenclatura correcta para burbujas de proceso. Por ahora es suficiente decir que un buen nombre generalmente consiste en una frase verbo-objeto tal como **VALIDAR ENTRADA** o **CALCULAR IMPUESTO.**

En algunos casos, el proceso contendrá el nombre de una persona o un grupo (por ejemplo, un departamento o una división de una organización), o de una computadora o un aparato mecánico. Es decir, el proceso a veces describe quién o *qué* lo está efectuando, más que describir el proceso mismo. Discutiremos esto con más detalle en el capítulo 17 cuando veamos el concepto de modelo esencial, y más adelante en la parte IV, cuando veamos los modelos de implantación.

9.1.2 El flujo

Un *flujo* se representa gráficamente por medio de una flecha que entra o sale de un proceso; un ejemplo se muestra en la figura 9.3. El flujo se usa para describir el movimiento de bloques o paquetes de información de una parte del sistema a otra. Por ello, los flujos representan datos en movimiento, mientras que los almacenes (que se describen en la sección 9.1.3) representan datos en reposo.

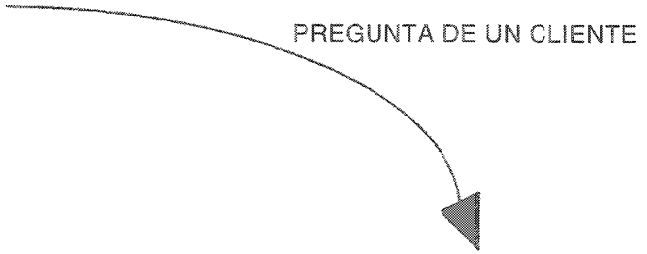


Figura 9.3: **Ejemplo de un flujo**

En la mayoría de los sistemas que modele como analista, los flujos realmente representarán datos, es decir, bits, caracteres, mensajes, números de punto flotante y los diversos otros tipos de información con los que las computadoras pueden tratar. Pero los DFD también pueden utilizarse para modelar otros sistemas aparte de los automatizados y computarizados; pudiera escogerse, por ejemplo, usar un modelo de DFD para modelar una línea de ensamblado en la que no haya componentes computarizados. En tales casos, los paquetes o fragmentos mostrados por los flujos serán típicamente *materiales físicos.* Un ejemplo de esto se muestra en la figura 9.4. Para muchos sistemas complejos del mundo real, el DFD mostrará el flujo de materiales *y* datos.

Nótese que los flujos de las figuras 9.3 y 9.4 tienen *nombre.* El nombre representa el significado del paquete que se mueve a lo largo del flujo. Un corolario de esto es que el flujo sólo lleva un tipo de paquete, como lo indica su nombre. El analista no debe dar al flujo un nombre como **MANZANAS Y NARANJAS Y ARTICULOS Y VARIAS COSAS MAS.** Sin embargo, veremos en la parte III que hay excepciones a este convenio: a veces es útil consolidar varios flujos elementales en

uno solo. Por ello, se pudiera ver un solo flujo llamado **VEGETALES** en lugar de diversos flujos llamados **PAPAS, COLES DE BRUSELAS** y **CHICHAROS.** Como veremos, esto requerirá de alguna explicación en el *diccionario de datos,* que se discutirá en el capítulo 10.

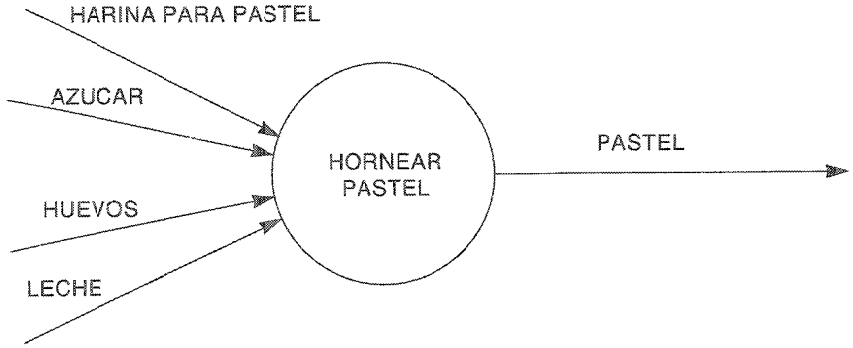


Figura 9.4: **DFD con flujo de materiales**

Aunque parezca obvio, tenga en mente que el mismo contenido pudiera tener distintos significados en distintas partes del sistema. Por ejemplo, considere el fragmento de sistema que se muestra en la figura 9.5.

El mismo fragmento de datos por ejemplo, 212-410-9955) tiene distinto significado cuando viaja a lo largo de un flujo llamado **NUMERO TELEFONICO** que cuando viaja a lo largo de uno llamado **NUMERO TELEFONICO VALIDO.** En el primer caso, significa un número telefónico que pudiera ser o no válido; en el segundo caso, significa un número telefónico que, dentro del contexto de este sistema, se sabe que es válido. Otra forma de verlo es que el flujo denominado "número telefónico" es como un ducto, lo suficientemente poco discriminador como para permitir el paso de números no válidos al igual que válidos; el flujo denominado **NUMERO TELEFONICO VALIDO** es más estrecho, o más discriminador, y permite pasar datos definidos más estrechamente.

Nótese también que los flujos muestran la *dirección:* una cabeza de flecha en cualquier extremo (o posiblemente ambos) del flujo indica si los datos (o el material) se está moviendo hacia adentro o hacia afuera de un proceso (o ambas cosas). El flujo que se muestra en la figura 9.6(a), por ejemplo, indica claramente que el número se está mandando *hacia* el proceso denominado **VALIDAR NUMERO TELEFONICO.** Y el flujo denominado **HORARIO DE ENTREGA DE CHOFERES** de la figura 9.6(b) claramente indica que es una salida generada por el proceso **GENERAR HORARIO DE ENTREGA DE CHOFERES.** Los datos que se mueven a lo largo de di-

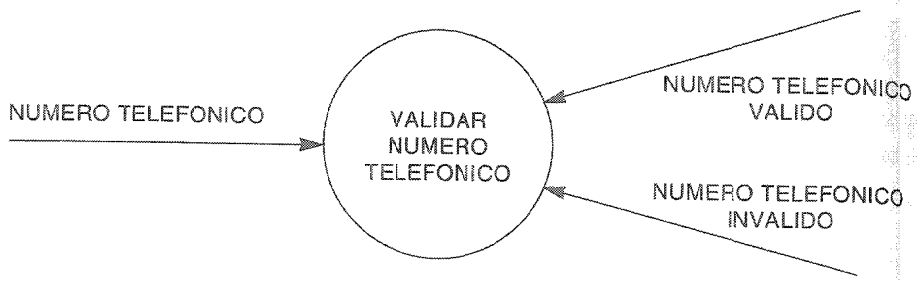


Figura 9.5: **DFD típico**

cho flujo viajarán ya sea a otro proceso (como entrada) o a un almacén (como se discutirá en la sección 9.1.3) o a un terminador (como se indica en la sección 9.1.4). El flujo de dos cabezas que se muestra en la figura 9.6(c) es un *diálogo,* es decir, un empacado conveniente de dos paquetes de datos (una pregunta y una respuesta) en el mismo flujo. En el caso de un diálogo, los paquetes en cada extremo de la flecha deben nombrarse, como se ilustra en la figura 9.6(c).[3]

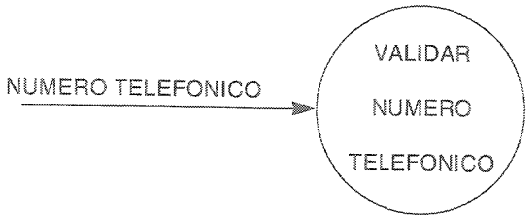


Figura 9.6(a): **Flujo de entrada**

Los flujos de datos pueden divergir o converger en un DFD; conceptualmente esto es algo así como un río principal que se divide en varios más pequeños, o varios pequeños que se unen. Sin embargo, esto tiene un significado especial en un DFD típico en el cual hay paquetes de *datos* que se mueven a través del sistema: en

3 Una alternativa aceptable en lugar del diálogo es el uso de dos flujos diferentes, uno que muestre las entradas (preguntas) a un proceso y otro que muestre las salidas (respuestas). Esto es, de hecho, una mejor forma de manejar las cosas si una entrada puede llevar a diversas acciones (y respuestas) del proceso. En el caso de una situación sencilla de pregunta-respuesta, el uso de un flujo de diálogo o de flujos de entrada y salida separados es cosa de gustos. La mayoría de los analistas prefieren la notación de diálogo porque 1) recalca al lector que los flujos de entrada y salida están relacionados entre sí y 2) reduce la complejidad del diagrama.

el caso de un flujo divergente, esto significa que se están mandando copias por duplicado de un paquete de datos a diferentes partes del sistema, *o bien* que un paquete complejo de datos se está dividiendo en varios paquetes individuales más, cada uno de los cuales se está mandando a diferentes partes del sistema, o que el ducto de flujo de datos lleva artículos con distintos valores (por ejemplo, vegetales cuyos valores pudieran ser "papa", "col de bruselas" o "ejote") que están siendo separados. De manera inversa, en el caso de un flujo convergente, significa que varios paquetes elementales de datos se están uniendo para formar agregados más complejos de paquetes de datos. Por ejemplo, la figura 9.7(a) muestra un DFD en el cual el flujo denominado **DETALLES DE ORDENES** diverge y lleva copias de los mismos paquetes a los procesos **GENERAR DOCUMENTOS DE ENVIO, ACTUALIZAR INVENTARIO** y **GENERAR FACTURAS.** La figura 9.7(b) muestra como el flujo **DOMICILIO DE CLIENTE** se divide en los paquetes más elementales **NUMERO TELEFONICO, CODIGO POSTAL,** y **CALLE Y NUMERO,** los cuales se mandan a tres procesos de validación diferentes.[4]

Figura 9.6(b): **Flujo de salida**

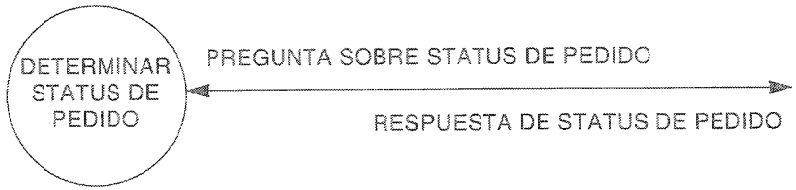


Figura 9.6(c): **Flujo de diálogo**

4 Cómo se realiza exactamente este proceso de duplicado o descomposición de paquetes de datos se considera asunto de la implantación, es decir, algo de lo que se tendrá que preocupar el diseñador pero que el analista no necesita mostrar en el modelo del sistema. A la larga pudiera llevarse a cabo por hardware o software, manualmente, o por magia negra. Si el analista está modelando un sistema ya existente, puede haber tentación de mostrar el mecanismo (es decir, el proceso) que lleva a cabo la duplicación/ descomposición de datos. Se discutirá esto más a fondo en la parte III.

Nótese que el flujo no responde a muchas dudas de procedimiento que pudiera tener cuando esté viendo un DFD: no responde a dudas acerca de petición de entradas o de flujos de salidas, por ejemplo. La figura 9.8(a) muestra el caso sencillo de un flujo de entrada que sale del proceso denominado PROCESAR ORDEN. ¿Pero cómo sucede esto? ¿PROCESAR ORDEN pide explícitamente al usuario las entradas? ¿O se mueven los paquetes a lo largo del flujo por su propia voluntad, sin ser pedidos? Similarmente, la figura 9.8(b) muestra un flujo de salidas sencillo que emana de **GENERAR REPORTE DE FACTURAS.** ¿Acaso las **FACTURAS** se mueven a lo largo del flujo cuando **GENERAR REPORTE DE FACTURAS** los quiera mandar, o cuando alguna otra parte del sistema pide el paquete? Finalmente, considere la situación más común que se muestra en la figura 9.8(c), en donde hay múltiples flujos de entrada y de salida: ¿en qué *secuencia* llegan los paquetes de datos y en qué secuencia se generan los paquetes de salida? Es decir, ¿el proceso Q requiere exactamente un paquete de los flujos A, B y C para producir exactamente un paquete de salida para los flujos X, Y y Z? ¿O existen dos Aes para cada tres Bes?

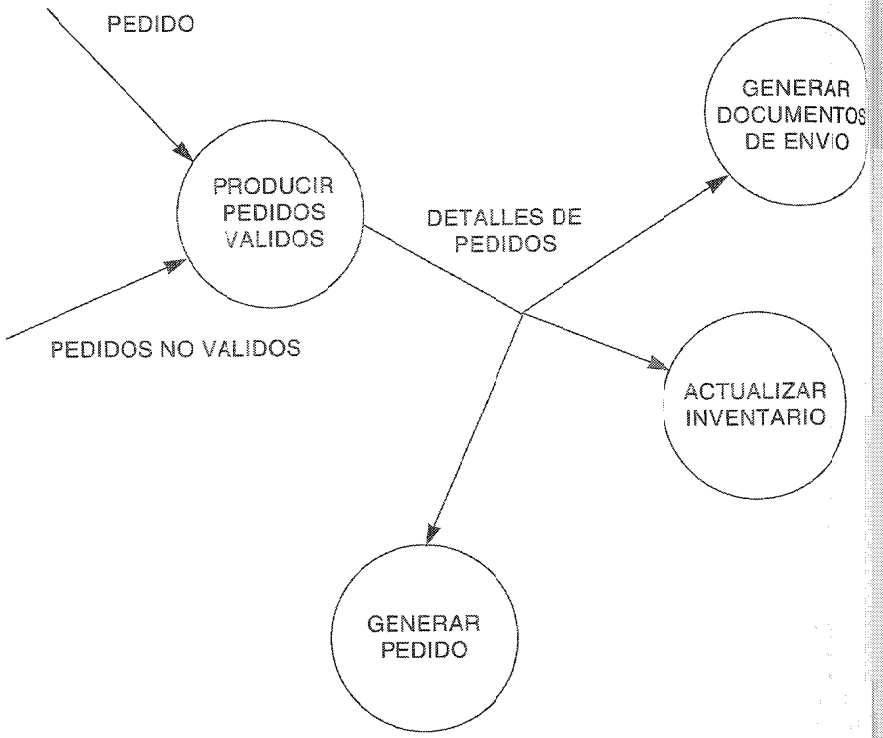


Figura 9.7(a): **Flujo divergente**

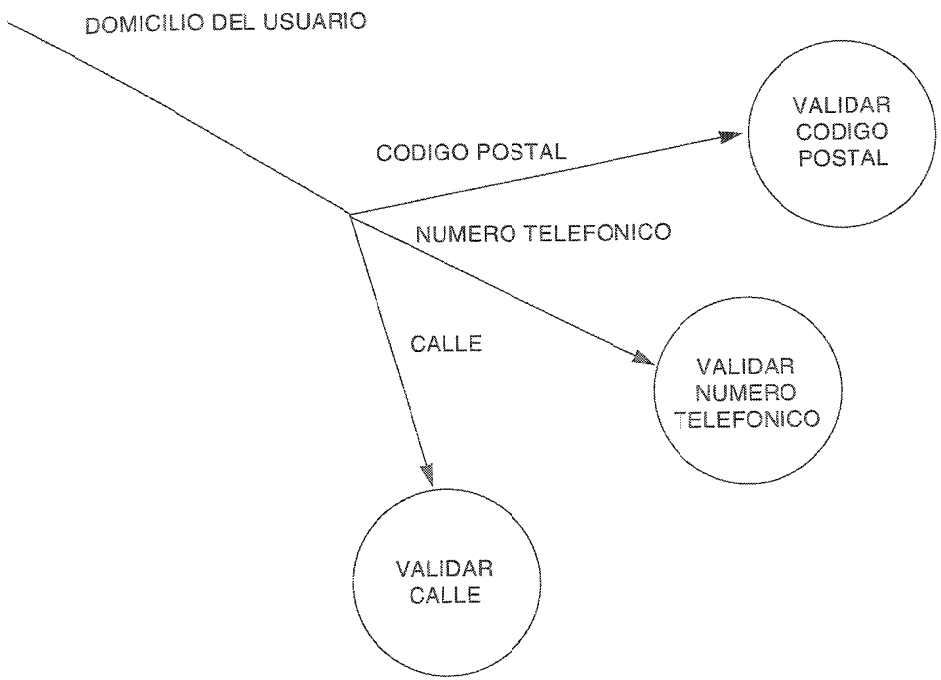


Figura 9.7(b): **Otro ejemplo de flujo divergente**

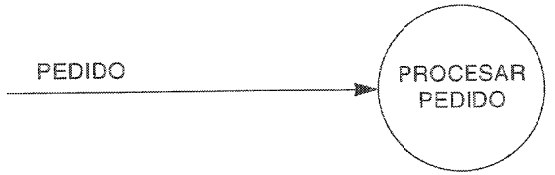


Figura 9.8(a): **Flujo de entrada**

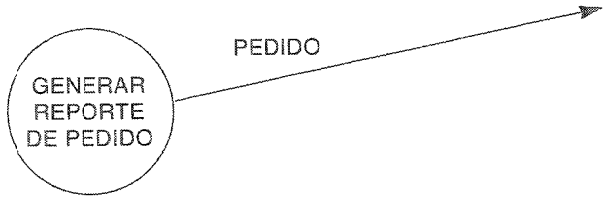


figura 9.8(b): **Flujo de salida**

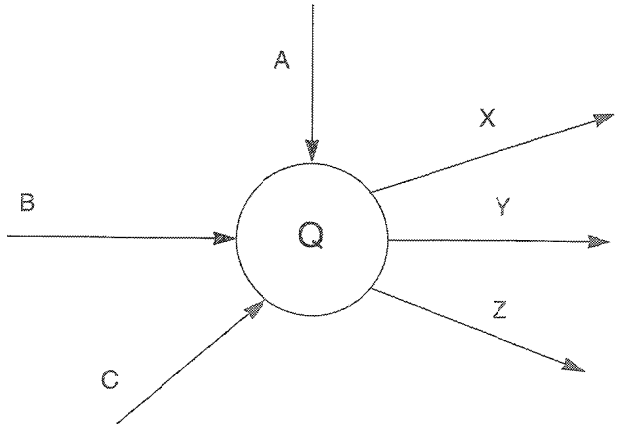


Figura 9.8(c): **Combinación de flujos de salida y entrada**

La respuesta a todas estas preguntas es muy sencilla: no sabemos. Todas estas interrogantes acarrean detalles de tipo *procedimiento*, que son el tipo de pregunta que se modelaría normalmente con un diagrama de flujo de datos o alguna otra herramienta de modelado de tipo procedimiento. *El DFD simplemente no intenta abordar estas cuestiones.* Si estas preguntas se vuelven importantes, entonces tendrá que modelarse el procedimiento interno de los diversos procesos; las herramientas para hacer esto se discuten en el capítulo 11.

9.1.3 El almacén

El *almacén* se utiliza para modelar una colección de paquetes de datos en reposo. Se denota por dos líneas paralelas, como los muestra la figura 9.9(a); una alternativa de notación se muestra en la figura 9.9(b);[5] otra más, que se utiliza en el caso de estudio del apéndice F, se muestra en la figura 9.9(c). De modo característico el nombre que se utiliza para identificar al almacén es el *plural* del que se utiliza para los paquetes que entran y salen del almacén por medio de flujos.

PEDIDOS

Figura 9.9(a): **Representación gráfica de un almacén**

5 La notación D1 en la figura 9.9(b) es simplemente una numeración que sirve para distinguir este almacén de otros que hay en el diagrama. La convención que sigue este libro *no* pide etiquetar o numerar los almacenes (simplemente porque no ha parecido necesario, ni siquiera útil), pero (como veremos en la sección 9.2), sí involucra la numeración de las burbujas.

D1 | PEDIDOS

Figura 9.9(b): **Notación alternativa para un almacén**

PEDIDOS

Figura 9.9(c): **Notación usada en el apéndice F**

Para el analista con conocimientos de proceso de datos es tentador referirse a los almacenes como *archivos* o *bases de datos* (por ejemplo, un archivo en cinta magnética o un archivo de disco organizado con IMS, DB2, ADABAS, IDMS, o algún otro sistema de manejo de bases de datos. De hecho, es así como a menudo se implantan los almacenes en un sistema computarizado; pero un almacén también pudiera consistir en datos almacenados en tarjetas perforadas, microfilm, microfichas, disco óptico o alguna más de otras posibles formas electrónicas. Y un almacén también puede ser un conjunto de fichas de papel en una caja de cartón, nombres y domicilios en un directorio, diversos archivos en un archivero, o varias formas *no* computarizadas. La figura 9.9(d) muestra un ejemplo característico de "almacén" en un sistema *manual* existente. Es precisamente debido a la variedad de formas de *implantación* posibles de un almacén que deliberadamente escogimos una notación gráfica simple y abstracta así como el término almacén en lugar de, por ejemplo, base de datos.[6]

Aparte de la forma física que toma el almacén, también existe la cuestión de su propósito: ¿Existe el sistema por causa de un *requerimiento* fundamental del usuario o por algún aspecto conveniente de la *realización* del sistema? En el primer caso, la base de datos existe como un área de almacenamiento diferida en el tiempo, necesaria entre dos procesos que ocurren en momentos diferentes. Por ejemplo, la figura 9.10 muestra un fragmento de un sistema en el cual, como *política del usuario* (independientemente de la tecnología que se use para implantar el sistema), el proceso de entrada de órdenes puede operar en tiempos diferentes (o posiblemente en el mismo) que el proceso de investigación de órdenes. El almacén de **ORDENES** debe existir en alguna forma, ya sea en disco, cinta, tarjetas o inscrito en piedra.

6 También es común referirse a un paquete de información del almacén como registro, y referirse a sus componentes como campos. Nada tiene de malo esta terminología, pero se usa tan a menudo en el contexto de las bases de datos que es probable que ocasione el tipo de problemas discutido anteriormente. Por ahora, utilizaremos el término paquete para describir una sola instancia de una colección de objetos relacionados en un almacén.

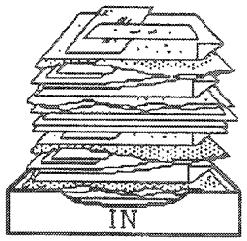


Figura 9.9(d): **Otro tipo de almacén**

La figura 9.11(a) muestra un tipo distinto de almacén: el almacén de implantación. Podemos imaginar al diseñador del sistema interponiendo un almacén de **ORDENES** entre **ENTRA ORDEN** y **PROCESA ORDEN** porque:

- Se espera que ambos procesos se ejecuten en la misma computadora, pero no hay suficiente memoria (o algún otro recurso de hardware) para cubrir ambos al mismo tiempo. Así, el almacén de **ORDENES** se crea como archivo intermedio, pues la tecnología de implantación disponible ha *forzado* a que los procesos se ejecuten en tiempos distintos.
- Se espera que cualquiera de los procesos, o ambos, se ejecuten en una configuración de hardware que es poco confiable. Así, el almacén de **ORDENES** se crea como respaldo en caso de que cualquiera de los procesos se aborte.

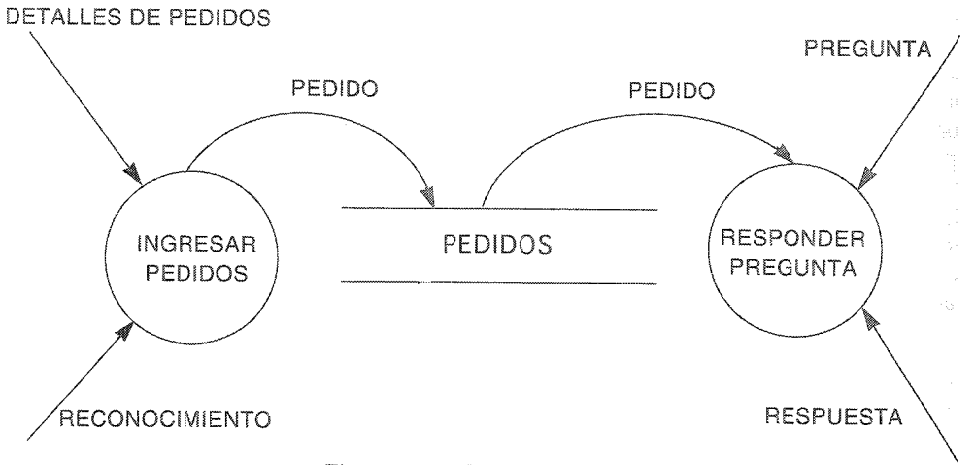


Figura 9.10: **Un almacén necesario**

- Se espera que diferentes programadores implanten los dos procesos (o, en un caso más extremo, que lo hagan diferentes *grupos* de programadores que trabajan en lugares geográficos distintos). Así, el almacén de **ORDENES** se crea para probar y corregir, de manera que si el sistema completo no trabaja ambos grupos puedan ver los contenidos del almacén y detectar el problema.
- El analista o el diseñador pensaron que el usuario pudiera algún día hacer accesos al almacén de **ORDENES** por alguna otra razón, aun cuando no haya expresado tal interés. En este caso, el almacén se crea anticipando necesidades futuras del usuario (y dado que costará algo implantar el sistema de esta manera, el usuario acabará pagando por algo que no se pidió).

Si fueran a excluirse los asuntos y modelar sólo los requerimientos *esenciales* del sistema, no existiría necesidad de un almacén de **ORDENES;** en lugar de eso se tendría un DFD como el que se muestra en la figura 9.11(b).

Como hemos visto hasta ahora, los almacenes se conectan por flujos a los procesos. Así, el contexto en el que se muestra un almacén en un DFD es uno de los siguientes (o ambos):

- Un flujo *desde* un almacén
- Un flujo *hacia* un almacén

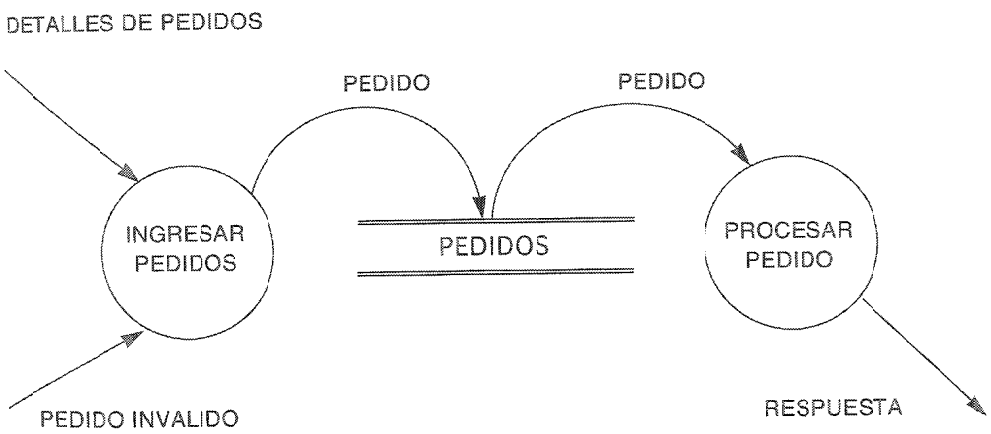


Figura 9.11(a): **Almacén "de implantación"**

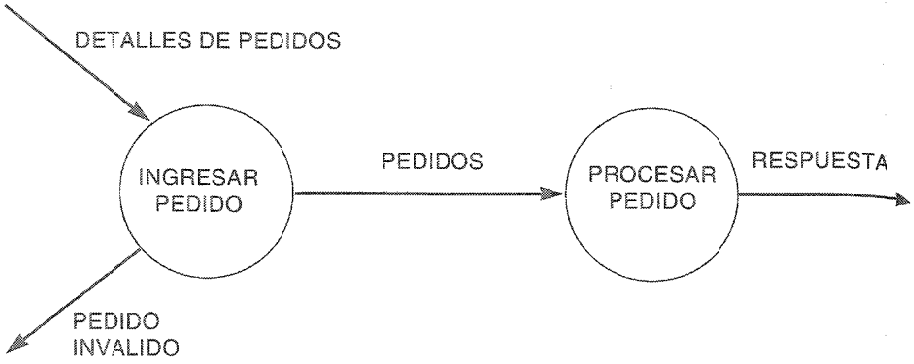


Figura 9.11(b): **Almacén de implantación eliminado**

En la mayoría de los casos, los flujos se etiquetarán como se discute en la sección 9.1.3. Sin embargo, muchos analistas no se molestan en etiquetar el flujo si una *instancia* completa del paquete fluye hacia o desde el almacén.[7] Por ejemplo, la figura 9.12 muestra un fragmento típico de un DFD.

Normalmente se interpreta un flujo que procede de un sistema como una lectura o un acceso a la información del almacén. Esto significa específicamente que:

- Se recupera del almacén un solo paquete de datos; esto es, de hecho el ejemplo más común de flujo desde un almacén. Imagínese, por ejemplo, un almacén llamado **CLIENTES,** donde cada paquete contiene nombre, domicilio y número telefónico de los clientes individuales. Así, un flujo típico del almacén podría implicar la recuperación de un paquete completo de información acerca de un cliente.
- Se ha recuperado más de un paquete del almacén. Por ejemplo, el flujo podría recuperar paquetes de información acerca de todos los clientes de la ciudad de Nueva York del almacén **CLIENTES.**
- Se tiene una porción de un paquete del almacén. En algunos casos, por ejemplo, sólo se podría recuperar la información del número telefónico del cliente del almacén **CLIENTES.**

7 Mencionaremos varias convenciones de este tipo en este capítulo, lo mismo que diversas más relacionadas con otras herramientas de modelado. El administrador del proyecto, el manual de estándares o la herramienta CASE que esté usando para su proyecto (vea el apéndice A) pudieran obligarlo a usar una convención u otra, pero debe ver que existe una cierta flexibilidad en cuanto a las herramientas y notación que se presentan aquí. Lo importante es la *consistencia:* todos los flujos portadores de paquetes que entran o salen de un almacén deben etiquetarse o no etiquetarse de manera consistente.

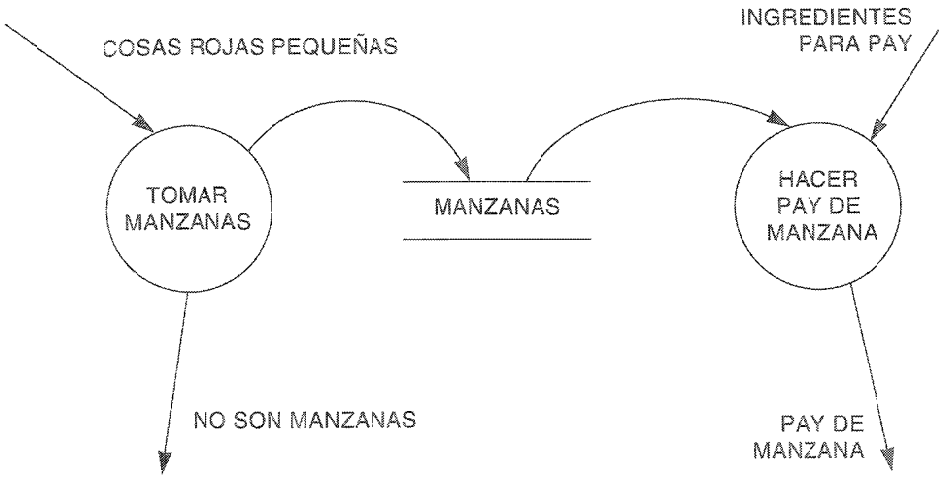


Figura 9.12: **Almacenes con flujos no etiquetados**

- Se tienen porciones de más de un paquete del almacén. Por ejemplo, un flujo podría recuperar del almacén **CLIENTES** la porción del código postal de todos los clientes que viven en el estado de Nueva York.

Como notamos antes cuando examinamos los flujos que entraban y salían de un proceso, tendremos muchas preguntas de tipo *procedimiento* cuando examinemos los flujos que entran y salen de un almacén: ¿representa el flujo un solo paquete, muchos, porciones de uno o porciones de diversos paquetes? En algunos casos, podemos darnos cuenta simplemente viendo la etiqueta del flujo: si el flujo no está etiquetado, significa que todo el paquete de información se está recuperando (como se dijo antes esto es simplemente una convención cómoda); si la etiqueta del flujo es la misma que la del almacén significa que se recupera todo un paquete (o múltiples instancias de uno completo); si la etiqueta del flujo es diferente del nombre del almacén, entonces se están recuperando uno o más componentes de uno o más paquetes.[8]

8 ¿Cómo podemos saber que las etiquetas del flujo tienen que ver con los componentes de un paquete de información del almacén? ¿Cómo saber, por ejemplo, que el flujo etiquetado como **NUMERO TELEFONICO** tiene algo que ver con los paquetes de información del almacén de **CLIENTES?** Es tentador, sobre todo en un proyecto real donde todo mundo está relativamente familiarizado con el tema, decir simplemente "Es que es obvio". Por supuesto que el número de teléfono forma parte del paquete del cliente. Pero, para estar seguro, se requiere una definición rigurosa de la composición del paquete **CLIENTES.** Esto se encuentra en el diccionario de datos, que se discutirá en el capítulo 10.

A pesar de que algunas de las preguntas de tipo procedimiento pueden responderse viendo con cuidado las etiquetas del flujo, no serán evidentes todos los detalles. De hecho, para conocer todo lo deseado acerca del flujo que emana del almacén, tendrán que examinarse los detalles: la *especificación del proceso* al cual se conecta el flujo. Las especificaciones de proceso se examinan en el capítulo 11.

Existe un detalle de tipo procedimiento del cual podemos estar seguros: el almacén es pasivo, y los datos no viajarán a lo largo del flujo a menos que el proceso lo solicite explícitamente. Existe otro detalle de tipo procedimiento que suponen, por convenio, los sistemas de proceso de datos: *el almacén no cambia cuando un paquete se mueve del almacén a lo largo del flujo.* Un programador pudiera referirse a esto como una lectura no destructiva o, en otras palabras, del almacén se recupera una copia del paquete y el almacén mantiene su condición original.[9]

Un flujo hacia un almacén habitualmente se describe como una escritura, una actualización o posiblemente una eliminación. Específicamente, sólo puede significar que se tiene una de las situaciones siguientes:

- Se están guardando uno o más paquetes nuevos en el almacén. Dependiendo de la naturaleza del sistema, los paquetes nuevos pudieran *anexarse* (es decir, de alguna manera acomodarse para que estén "después" de los paquetes existentes); o pudieran colocarse en algún lado entre los paquetes existentes. Esto es a menudo un asunto de la implantación (es decir, controlado por el sistema específico de administración de bases de datos), por lo que el analista no debiera preocuparse acerca de ello. Podría ser, sin embargo, cuestión de una política del usuario.
- Uno o más paquetes se están borrando o retirando del almacén.
- Uno o más paquetes se están modificando o cambiando. Esto pudiera traer consigo un cambio de *todo* un paquete, o (más comúnmente), de sólo una porción, o de una porción de múltiples paquetes. Por ejemplo, suponga que la policía tiene un almacén de sospechosos y que cada paquete contiene sus nombres y domicilios; puede ofrecérsele una nueva "identidad" a un sospechoso que coopera, en cuyo caso *toda* la información relacionada con su paquete cambiaría. Como alternativa, considere un almacén **CLIENTES** que contenga información acerca de los clientes que residen en Manhattan; si la oficina de correos decidiera cambiar el código postal para un área de Manhattan (como sucedió en 1983, cuando el área de Manhattan donde yo vivía cambió de código 10028 a 10128), se necesitaría un cambio a una porción de diversos paquetes.

9 Si está usando un DFD para mocelar algo que no sea puramente un sistema de proceso de información, esto pudiera no darse. Por ejemplc, un almacén puede contener cosas físicas, y el flujo puede ser un mecanismo para transferir materiales del almacén al proceso. En este caso, se sacaría físicamente un paquete del almacén y, como resultado, tendría menos contenidos. En un modelo de sistema que contenga almacenes de información y almacenes físicos, es importante hacer anotaciones en el DFD (o dar explicaciones en el diccionario de datos) para que el lector no se confunda.

En todos estos casos es evidente que el almacén cambió como resultado del flujo que ingresa. El proceso (o procesos) conectados con el otro extremo del flujo es el responsable de realizar el cambio al almacén.

Un punto que debiera ser evidente de todos los ejemplos que se han mostrado hasta ahora es el siguiente: los flujos conectados a un almacén sólo pueden transportar paquetes de información que el almacén sea capaz de guardar. Por ello, un flujo conectado a un almacén **CLIENTES** sólo puede transportar información relacionada con los clientes contenidos por el almacén; no puede transportar paquetes de inventarios o paquetes de manufactura o datos astronómicos.

9.1.4 El Terminador

El siguiente componente del DFD es un *terminador;* gráficamente se representa como un rectángulo, como se muestra en la figura 9.13. Los terminadores representan entidades externas con las cuales el sistema se comunica. Comúnmente, un terminador es una persona o un grupo, por ejemplo, una organización externa o una agencia gubernamental, o un grupo o departamento que esté dentro de la misma compañía u organización, pero fuera del control del sistema que se está modelando. En algunos casos, un terminador puede ser otro sistema, como algún otro sistema computacional con el cual se comunica éste.

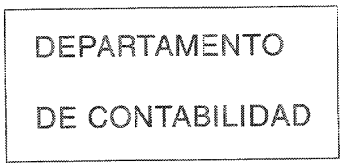


Figura 9.13: **Representación gráfica de un terminador**

Suele ser muy fácil identificar los terminadores en el sistema que se está modelando. A veces el terminador es el usuario; es decir, en sus discusiones con el usuario, éste dirá "Pretendo suministrar al sistema los datos X, Y y Z, y espero que me regrese los datos A, B y C". En otros casos, el usuario se considera parte del sistema y ayudará a identificar los terminadores relevantes; por ejemplo, le dirá "Tenemos que estar listos para recibir las formas tipo 321 del departamento de contaduría, y debemos enviar reportes financieros semanales al comité de finanzas". En este último caso, es evidente que el departamento de contaduría y el comité de finanzas son terminadores separados con los cuales se comunica el sistema.

Existen tres cosas importantes que debemos recordar acerca de los terminadores:

1. Son *externos* al sistema que se está modelando; los flujos que conectan los terminadores a diversos procesos (o almacenes) en el sistema representan la interfaz entre él y el mundo externo.

2. Como consecuencia, es evidente que ni el analista ni el diseñador del sistema están en posibilidades de cambiar los contenidos de un terminador o la manera en la que trabaja. En el lenguaje usado por diversos libros de texto clásicos sobre análisis estructurado, el terminador está fuera del dominio del cambio. Lo que esto significa es que el analista está modelando un sistema con la intención de permitir una considerable flexibilidad y libertad al diseñador para elegir la mejor implantación posible (la más eficiente o la más confiable, etc.). El diseñador puede implantar el sistema de manera bastante diferente de aquella en la que actualmente está implantado; el analista puede escoger modelar los requerimientos del sistema en forma que se vea considerablemente diferente de la manera en la que actualmente el usuario visualiza mentalmente el sistema (se verá más acerca de esto en la sección 9.4 y la parte III). Sin embargo, *el analista de sistemas no puede modificar los contenidos, la organización ni los procedimientos internos asociados con los terminadores.*

3. Las relaciones que existan *entre* los terminadores no se muestran en el modelo de DFD. Pudieran existir de hecho diversas relaciones, pero, por definición, no son parte del sistema que se está estudiando. De manera inversa, si existen relaciones entre los terminadores y si es esencial para el analista modelarlos para poder documentar los requerimientos del sistema, entonces, por definición, los terminadores son en realidad parte del sistema y debieran modelarse como procesos.

En los ejemplos sencillos que se han discutido hasta ahora hemos visto sólo uno o dos terminadores. En un sistema real típico pueden existir docenas de terminadores diferentes interactuando con él. Identificar los terminadores y su interacción con el sistema es parte del proceso de construir el modelo del ambiente, que se discutirá en el capítulo 17.

9.2 GUIA PARA LA CONSTRUCCION DE DFD

En la sección anterior vimos que los diagramas de flujo de datos constan de cuatro componentes sencillos: procesos (burbujas), flujos, almacenes y terminadores. Armado con estas herramientas, puede comenzar a entrevistar a los usuarios y a construir modelos de DFD de sistemas.

Sin embargo, existe un número de reglas adicionales que se requieren para poder utilizar DFD con éxito. Algunas de estas reglas ayudarán para no elaborar DFD erróneos (por ejemplo, incompletos o lógicamente inconsistentes). Algunas de

las reglas tienen la finalidad de ayudarle para que dibuje un DFD grato a la vista, y que, por tanto, tenga más probabilidades de que lo lea con cuidado el usuario.

Las reglas incluyen las siguientes:

1. Escoger nombres con significado para los procesos, flujos, almacenes y terminadores.

2. Numerar los procesos.

3. Redibujar el DFD tantas veces como sea necesario estéticamente.

4. Evitar los DFD excesivamente complejos.

5. Asegurarse de que el DFD sea internamente consistente y que también lo sea con cualesquiera DFD relacionados con él.

9.2.1 Escoger nombres con significado para los procesos, flujos, almacenes y terminadores.

Como se ha visto, un proceso en un DFD puede representar una *función* que se está llevando a cabo, o pudiera indicar *cómo* se está llevando a cabo, identificando a la persona, grupo o mecanismo involucrado. En el último caso, obviamente es importante etiquetar con precisión el proceso, de modo que quienes leen el DFD, especialmente los usuarios, puedan confirmar que se trata de un modelo preciso. Sin embargo, si el proceso lo hace una sola persona, recomiendo que identifique el papel que dicha persona está representando, más que su nombre o identidad. Así, en lugar de dibujar un proceso como el que se muestra en la figura 9.14(a), con el nombre de Paco inmortalizado a la vista de todos, sugerimos que represente el proceso como se muestra en la figura 9.14(b). La razón de esto tiene tres aspectos:

1. Paco pudiera ser reemplazado la próxima semana por María o por Juan. ¿Para qué introducir en el modelo algo susceptible de volverse obsoleto?

2. Paco pudiera estar realizando diversas labores diferentes en el sistema. En lugar de dibujar tres burbujas distintas, cada una etiquetada como Paco pero con distinto significado, es preferible indicar la labor misma que se está logrando, o por lo menos el papel que Paco está jugando en el momento (como se modela en cada una de sus burbujas).

3. Identificar a Paco es probable que atraiga atención hacia la manera particular en la que realiza la labor dada. Como veremos en la parte III, generalmente desearemos concentrarnos en la *política de negocios* que debe cumplirse, sin referirnos a los procedimientos (los cuales puecen basarse en costumbres e historia que ya no sean relevantes) que se utilizan para seguir dicha política.

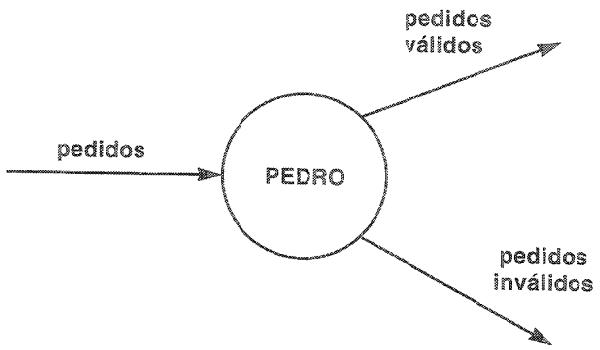


Figura 9.14(a): **Nombre de proceso no apropiado**

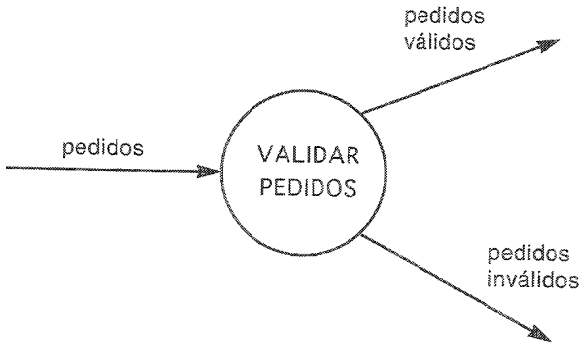


Figura 9.14(b): **Nombre de proceso más apropiado**

Si tenemos suerte de evitar nombres de personas (o de grupos) y papeles políticos, entonces podemos etiquetar los procesos de tal manera que se puedan identificar las *funciones* que el sistema está llevando a cabo. Un buen sistema que se puede utilizar para nombrar procesos es usar un *verbo* y un *objeto*. Es decir, escoja un verbo activo (un verbo transitivo, uno que tenga objeto) y un objeto apropiado para formar una frase descriptiva para el proceso. Los siguientes son ejemplos de nombres de procesos:

- CALCULAR TRAYECTORIA DEL PROYECTIL
- PRODUCIR INFORME DE INVENTARIO
- VALIDAR NUMERO TELEFONICO
- ASIGNAR ESTUDIANTES A LA CLASE

Encontrará, al seguir estas reglas, que es considerablemente más fácil utilizar verbos y objetos específicos si el proceso mismo es relativamente simple y está bien definido. Sin embargo, aun en los casos sencillos hay la tentación de utilizar nombres ambiguos como HACER, MANEJAR y PROCESAR. Cuando se utilizan verbos tan "elásticos" (verbos con significados para cubrir casi cualquier situación), a menudo significa que el analista no está seguro de cuál función se está llevando a cabo o que se han agrupado diversas funciones que en realidad no debieran agruparse. Estos son algunos nombres de procesos poco adecuados:

- HACER ALGO
- FUNCIONES MISCELANEAS
- MANEJAR ENTRADAS
- ENCARGARSE DE CLIENTES
- DATOS DE PROCESOS
- EDICION GENERAL

Los nombres de los procesos (al igual que los nombres de flujos y de terminadores) deben provenir de un vocabulario que tenga algún significado para el usuario. Esto sucederá de manera muy natural si el DFD se dibuja como resultado de una serie de entrevistas con los usuarios y si el analista tiene algún entendimiento mínimo de la materia de aplicación subyacente. Pero se deben tener en cuenta dos precauciones:

1. Hay una tendencia natural de los usuarios de utilizar abreviaturas y acrónimos específicos con los que están familiarizados; esto es cierto tanto para los procesos como para los flujos que describen. Por desgracia, esto suele resultar en un DFD fuertemente orientado a la manera en la que se hacen las cosas ahora. Por ello, el usuario pudiera decir: "Bueno, se consigue una copia de la forma 107, la forma rosada, usted sabe, y se la mandamos a José una vez llena". Una buena manera de evitar tales términos idiosincrásicos es escoger verbos (como "llenar") y objetos (como "forma 107") que tendrían significado para alguien de la misma industria o aplicación, pero que trabaje en una compañía u organización diferente. Si se está creando un sistema para bancos, los nombres de procesos y de flujos debieran, idealmente, ser comprensibles para alguien de un banco distinto.

2. Si el DFD lo dibuja alguien que tenga bases de programación, habrá la tendencia a utilizar terminología orientada a la programación, tal como: "RUTINA", "PROCEDIMIENTO", "SUBSISTEMA" y "FUNCION", aunque dichos términos pudieran no tener significado alguno en el mundo del usuario. A menos que oiga a los usuarios utilizar estas palabras en su propia conversación, evite utilizarlas en su DFD.

9.2.2 Numerar el proceso

Como una forma conveniente de referirse a los procesos en un DFD, muchos analistas numeran cada burbuja. No importa mucho cómo se haga esto, de izquierda a derecha, de arriba abajo o de cualquier otra manera servirá, *mientras haya constancia en la forma de aplicar los números.*

La única cosa que se debe tener en mente es que el sistema de numeración implicará, para algunos lectores casuales de su DFD, una cierta *secuencia* de ejecución. Esto es, cuando se muestre el DFD a un usuario, éste pudiera preguntar: "¿Acaso la burbuja número 1 sucede primero, luego la 2 y luego la 3?". De hecho, otros analistas y programadores pudieran hacer la misma pregunta; cualquiera que esté familiarizado con un diagrama de flujo puede cometer el error de suponer que los números asociados con las burbujas implican alguna secuencia.

Esto no es así en absoluto. El modelo de DFD es una red de procesos asincrónicos que se intercomunican, lo cual es, de hecho, una representación precisa de la manera en la que en realidad muchos sistemas operan. Alguna secuencia pudiera implicarse por la presencia o ausencia de datos (por ejemplo, pudiera resultar que la burbuja 2 no pueda realizar su función sino hasta que reciba datos de la burbuja 1), pero el esquema de numeración no tiene nada que ver con eso.

¿Para qué numerar entonces las burbujas? En parte, como se indicó anteriormente, es una manera muy conveniente de referirse a los procesos. Es más fácil en una discusión animada sobre un DFD decir "burbuja 1" en lugar de "EDITAR TRANSACCION Y REPORTAR ERRORES". Pero de mayor importancia aún es el hecho de que los números se convierten en base para la numeración jerárquica cuando se introduzcan los diagramas de flujo por niveles en la sección 9.3.

9.2.3 Evitar los DFD demasiado complejos

El propósito de un DFD es modelar de manera precisa las funciones que debe llevar a cabo un sistema y las interacciones entre ellas. Pero otro propósito del DFD es ser leído y comprendido, no sólo por el analista que construyó el modelo, sino por los usuarios que sean los expertos en la materia de aplicación. Esto significa que el diagrama debe ser fácilmente entendido, fácilmente asimilado y placentero a la vista.

Trataremos sobre varias reglas estéticas en la siguiente subsección, pero hay una regla principal que se debe tener en mente: *no cree un DFD con demasiados procesos, flujos, almacenes y terminadores.* En la mayoría de los casos, esto significa que no debiera haber más de media docena de procesos y almacenes, flujos y terminadores relacionados en un solo diagrama.[10] Otra manera de decir esto es que el DFD debe caber cómodamente en una hoja normal.

10 Esta regla proviene de "The Magical Number Seven, Plus or Minus Two", de George Miller, *Psychology Review*, 1956.

Existe una excepción importante a esto, que discutiremos en el capítulo 18: un DFD especial conocido como *diagrama de contexto,* que representa el sistema entero como un solo proceso y destaca las interfaces entre el sistema y los terminadores externos. La figura 9.15 muestra un diagrama de contexto típico y, como puede verse, con él basta para asustar a muchos analistas, por no mencionar al usuario desprevenido. Comúnmente, los diagramas de contexto como el que se muestra en la figura 9.15 no se pueden simplificar, pues describen, con el más alto detalle, una realidad que es intrínsecamente compleja.[11]

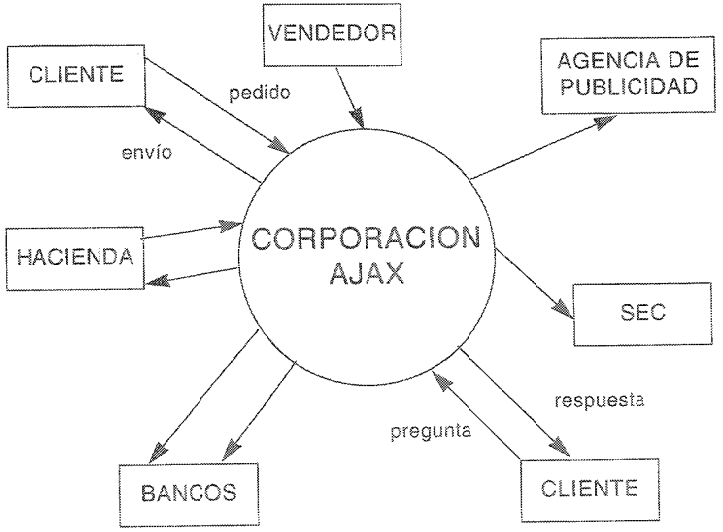


Figura 9.15: **diagrama de contexto típico**

9.2.4 Redibujar el DFD tantas veces como sea necesario

En un proyecto real de análisis de sistemas, el DFD que se analiza en este capítulo tendrá que dibujarse y volverse a dibujar, a menudo hasta 10 veces o más, antes de 1) ser técnicamente correcto, 2) ser aceptable para el usuario y 3) estar lo

11 En la realidad, hay algo que podemos hacer: si hay diversos flujos distintos entre un terminador y una sola burbuja del sistema, pueden consolidarse en un solo flujo de datos. El diccionario de datos, que se discute en el capítulo 10, se usará para explicar la composición y significado de un flujo agregado. Y, si el diagrama de contexto se está mostrando a un público muy diverso (por ejemplo, distintos grupos de usuarios con diferentes intereses), se pueden dibujar varios diagramas de contexto que enfaticen los flujos particulares que puedan interesarle a cada grupo de usuarios. Pero, en la mayoría de los casos, no hay escapatoria: si el sistema global es intrínsecamente complejo, lo será también el diagrama de contexto. Se discutirá más a fondo esto en el capítulo 18.

suficientemente bien dibujado como para que no sea embarazoso mostrarlo a la dirección de la organización. Esto pudiera parecer mucho trabajo, pero bien vale el esfuerzo de desarrollar un modelo preciso, congruente y agradable, de los requerimientos de su sistema. Lo mismo se cumple para cualquier otra disciplina de ingeniería: ¿querría usted volar en un avión diseñado por ingenieros que se aburrieron de sus dibujos de ingeniería tras la segunda iteración?[12]

¿Qué hace estéticamente agradable a un DFD? Esto es obviamente una cuestión de gustos y puede determinarse por normas dispuestas por su organización o por las características particulares de cualquier paquete que utilice de diseño de diagramas basado en una estación de trabajo automatizada. Y la opinión del usuario pudiera ser un tanto diferente de la suya; lógicamente, cualquier cosa que el usuario encuentre agradable debe determinar la manera en la que se dibuje el diagrama. Algunos de los asuntos que surgen para ser tratados en esta área son los siguientes:

- *Tamaño y forma de las burbujas.* Algunas organizaciones dibujan diagramas de flujo de datos con rectángulos u óvalos en lugar de círculos; esto es obviamente una cuestión de estética. Además, algunos usuarios pudieran molestarse si las burbujas del DFD no son del mismo tamaño: creen que si una burbuja es más grande que otra eso significa que esa parte del sistema es más importante o que difiere del resto de una manera significativa. (De hecho, por lo común sucede sólo debido a que el nombre de la burbuja era tan largo que el analista tuvo que dibujarla más grande para poderlo abarcar.)

- *Flujos curvos vs. rectos.* Para ilustrar esto, considere los DFD de la Figuras 9.16(a) y 9.16(b). ¿Cuál es más agradable? Muchos observadores se encogerán de hombros y dirán "en realidad da igual". Pero otros, y éste es el meollo del asunto, escogerán uno y rechazarán violentamente el otro. Obviamente, sería una buena idea conocer de antemano qué opción será aceptada y cuál será rechazada. El asunto de las flechas cruzadas es similar. ¿Se permiten o no se permiten?

- *Diagramas hechos a mano vs. los diagramas generados por máquina.* Dentro de algunos años, casi todos los DFD y modelos de sistemas relacionados se dibujarán con sistemas gráficos por computadora; sin embargo, muchos de los diagramas todavía hoy se dibujan a mano porque los analistas no tienen acceso a tales herramientas. No obstante, el asunto aquí es la reacción del usuario a estos diagramas: algunos prefieren marcadamente los diagramas generados por la máquina pues son más ordenados, mientras que otros prefieren los dibujados a mano porque los hace sentir que el diagrama no se ha "congelado" aún, y que todavía pueden introducir cambios.

12 En caso de que piense que los aviones son diferentes de los sistemas automatizados, o que son más críticos, tenga en mente que en la actualidad la mayoría de los aviones están controlados por computadora; un avión grande de pasajeros puede tener una docena o más de sistemas computacionales complejos a bordo, que a su vez tienen interfase con sistemas de tiempo real complejos tales como el que usa la administración federal de aviación para revisar el espacio aéreo en la vecindad de los aeropuertos.

9.2.5 Asegúrese de que su DFD sea lógicamente consistente

Como se verá en el capítulo 14, existen reglas que ayudan a asegurar la consistencia del DFD con los *otros* modelos de sistemas: el diagrama de entidad-relación, el diagrama de transición de estados, el diccionario de datos, y la especificación de procesos. Sin embargo, existen algunas reglas respecto a cómo asegurar que el DFD mismo sea consistente. Las principales reglas de consistencia son:

- *Evite sumideros infinitos,* burbujas que tienen entradas pero no salidas. También son conocidos por los analistas como "agujeros negros", en analogía con las estrellas cuyo campo gravitacional es tan intenso que ni la luz se les escapa. Un ejemplo de vórtice infinito se da en la figura 9.17.

- *Evite las burbujas de generación espontánea,* que tienen salidas sin tener entradas, porque son sumamente sospechosas y generalmente incorrectas. Un ejemplo factible de una burbuja que sólo tiene salidas es un generador de números aleatorios, pero es difícil imaginar algún otro ejemplo razonable. Una burbuja típica que sólo tiene salidas se muestra en la figura 9.18.

- *Tenga cuidado con los flujos y procesos no etiquetados.* Esto suele ser un indicio de falta de esmero, pero puede esconder un error aún más grave: a veces el analista no etiqueta un flujo o un proceso porque simplemente no se le ocurre algún nombre razonable. En el caso de un flujo no etiquetado, pudiera significar que diversos datos elementales no relacionados se agruparon arbitrariamente; en el caso de un proceso no etiquetado, pudiera significar que el analista estaba tan confundido que dibujó un diagrama de flujo disfrazado en lugar de un diagrama de flujo de datos.[13]

13 Hay un convenio idiomático que viola esto, que se discutirá en la sección 9.1.3: un flujo no etiquetado que sale o entra de un almacén es, por acuerdo, un indicio de que una instancia (o registro) completo se está metiendo o sacando del almacén.

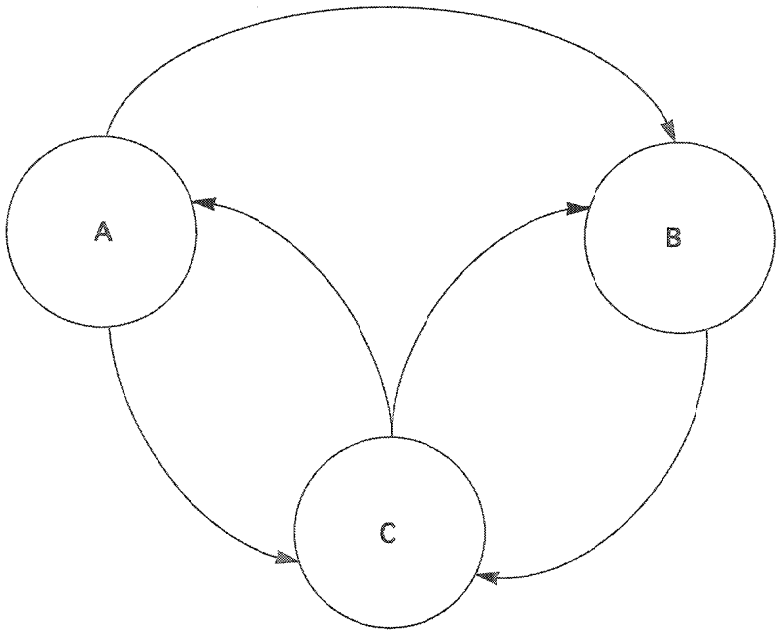


Figura 9.16(a): **Versión de un DFD**

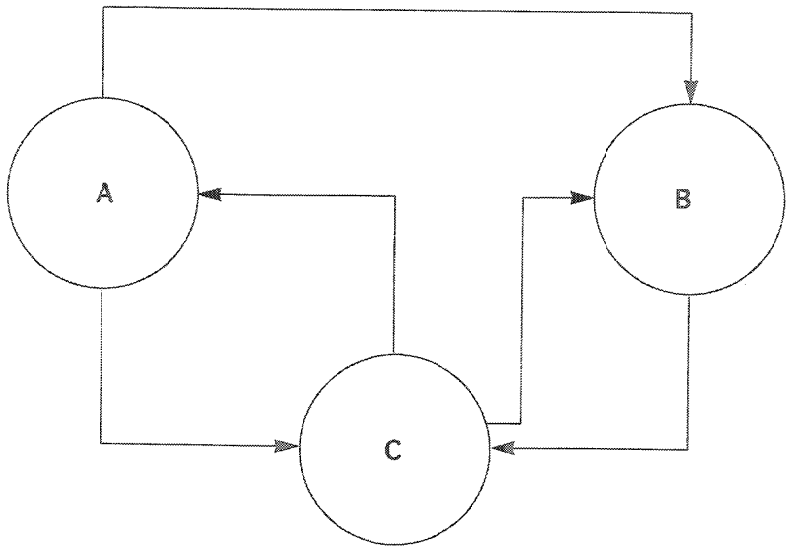


Figura 9.16(b): **Versión diferente de un DFD**

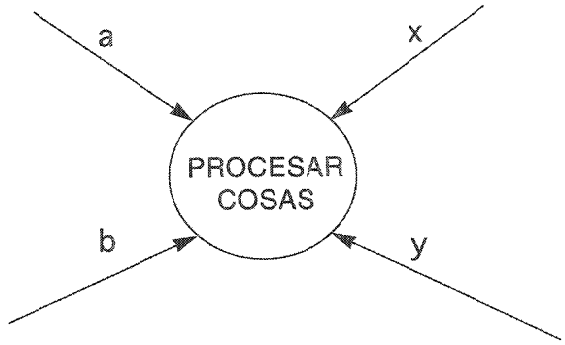


Figura 9.17: **Ejemplo de sumidero infinito**

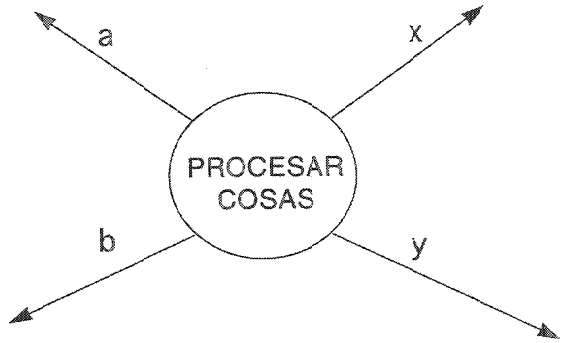


Figura 9.18: **Ejemplo de burbuja únicamente de salida**

- *Tenga cuidado con los almacenes de "sólo lectura" o "sólo escritura".* Esta regla es análoga a la de los procesos de "únicamente entradas" o "únicamente salidas". Un almacén típico debiera tener tanto entradas como salidas.[14] La única excepción a esta regla es el almacén externo, que sirve de interfaz entre el sistema y algún terminador externo. La figura 9.19 muestra un ejemplo de un sistema de bolsa con un almacén legítimo que sólo es para escritura.

14 A veces pudiera no ser evidente inmediatamente si el almacén tiene tanto entradas como salidas. Como veremos en la siguiente sección, a menudo los DFD se dividen en partes, por lo que pudiéramos encontrar que una parte del sistema parece sólo tener almacenes de únicamente lectura (o únicamente escritura). Alguna otra parte del sistema, documentada en otro DFD, pudiera tener la actividad de únicamente escritura (o únicamente lectura) correspondiente. Para verificar que alguna parte del sistema lea y alguna otra escriba en el almacén se requiere de una labor muy tediosa de revisión; es aquí donde los paquetes de análisis automatizado de sistemas que se discuten en el apéndice A se vuelven extremadamente valiosos.

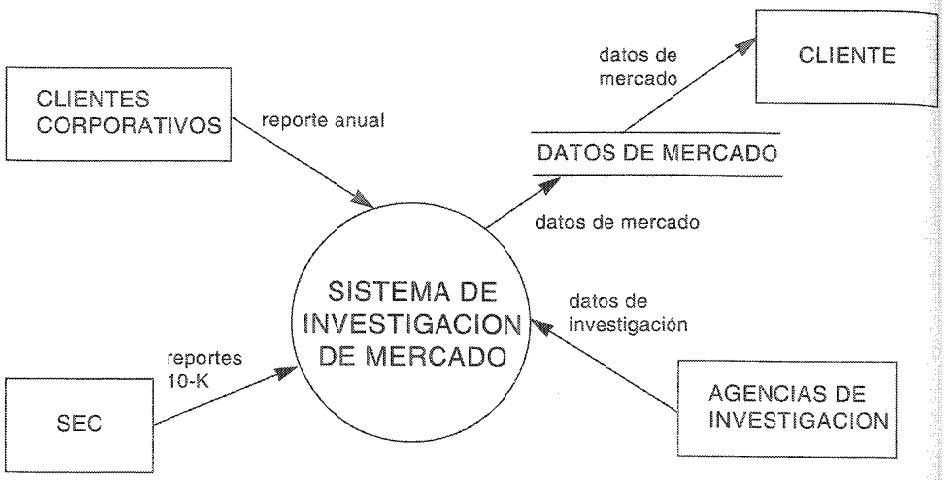


Figura 9.19: **Caso legítimo de almacén de únicamente escritura**

9.3 DFD por niveles

Hasta donde hemos llegado en este capítulo, los únicos diagramas de flujo de datos completos que hemos visto son el sistema sencillo de tres burbujas de la figura 9.1 y el sistema de una burbuja de la figura 9.19, pero los DFD que veremos en proyectos reales son considerablemente mayores y más complejos. Considere por ejemplo el DFD que se muestra en la figura 9.20. ¿Puede imaginarse mostrándole esto al usuario típico?

La sección 9.2.3 sugiere que deben evitarse diagramas como el de la figura 9.20. ¿Pero cómo? Si el sistema es intrínsecamente complejo y tiene docenas o incluso cientos de funciones que modelar, ¿cómo puede evitarse el tipo de DFD que muestra la figura 9.20?

La respuesta es organizar el DFD global en una serie de niveles de modo que cada uno proporcione sucesivamente más detalles sobre una porción del nivel anterior. Esto es análogo a la organización de mapas en un atlas, como se discutió en el capítulo 8: esperaríamos ver un mapa global que mostrara un país completo, o tal vez incluso el mundo completo; los mapas subsecuentes mostrarían los detalles de los países individuales, los estados individuales dentro de los países, etc. En el caso de los DFD, la organización de los niveles se muestra conceptualmente en la figura 9.21.

El DFD del primer nivel consta sólo de una burbuja, que representa el sistema completo; los flujos de datos muestran las interfaces entre el sistema y los terminadores externos (junto con los almacenes externos que pudiera haber, como lo ilustra

la figura 9.19). Este DFD especial se conoce como *diagrama de contexto* y se explica en el capítulo 18.

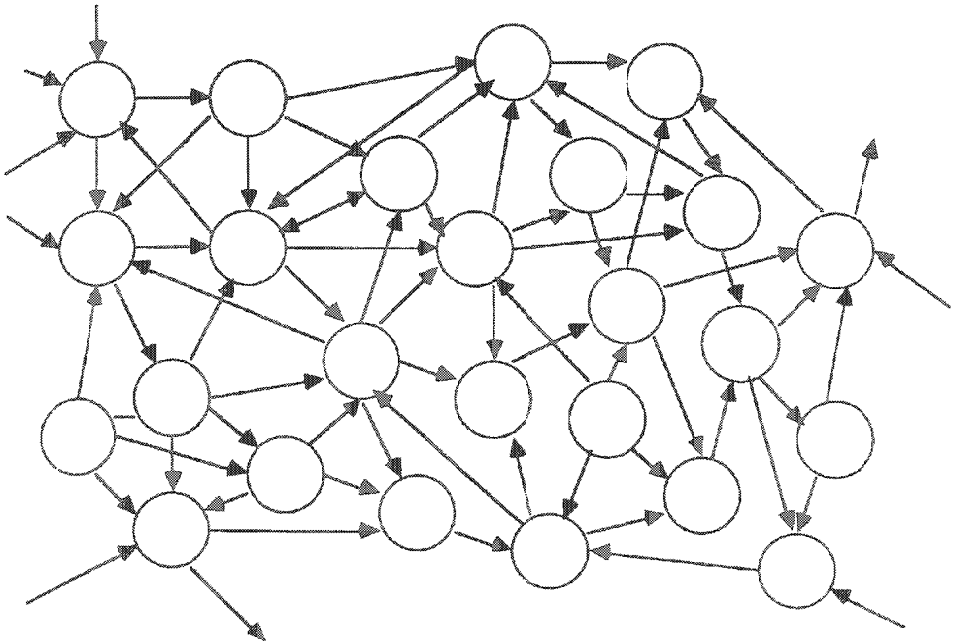


Figura 9.20: **DFD complejo**

El DFD que sigue del diagrama de contexto se conoce como la figura 0. Representa la vista de más alto nivel de las principales funciones del sistema, al igual que sus principales interfaces. Como se discutió en la sección 9.2.2, cada una de estas burbujas debiera numerarse para una referencia conveniente.

Los números también sirven como una manera adecuada de relacionar una burbuja con el siguiente nivel del DFD que la describe más a fondo. Por ejemplo:

- La burbuja 2 en la figura 0 se asocia con un DFD inferior conocido como figura 2. Las burbujas de la figura 2 se numeran 2.1, 2.2, 2.3, etc.
- La burbuja 3 de la figura 0 se asocia con un DFD inferior conocido como figura 3, las burbujas de la figura 3 se numeran 3.1, 3.2, 3.3, etc.
- La burbuja 2.2 de la figura 2 se asocia con un DFD de nivel inferior conocido como figura 2.2. Las burbujas de ésta se numeran 2.2.1, 2.2.2, 2.2.3, etc.

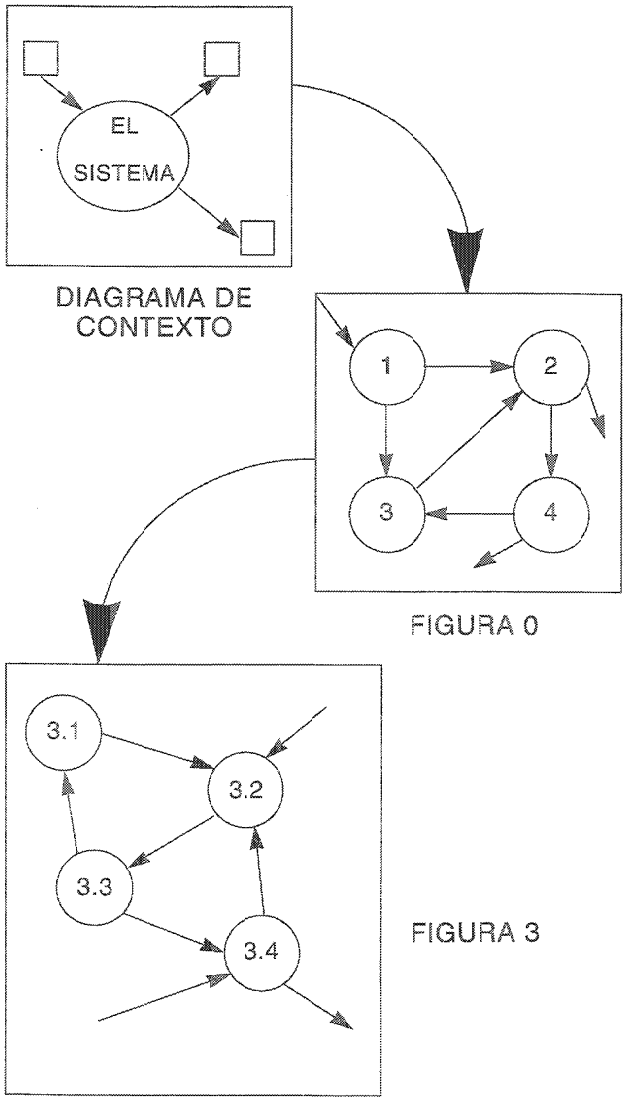


Figura 9.21: **DFD por niveles**

- Si una burbuja tiene nombre (que en realidad debiera tenerlo) entonces dicho nombre se reutiliza en la figura de nivel inmediato inferior. Así, si la burbuja 2.2 se llama **CALCULAR IMPUESTO DE VENTA,** entonces la figura 2.2, que parte la burbuja 2.2 en más detalles, debe etiquetarse **"figura 2.2: CALCULAR IMPUESTO DE VENTA".**

Como podrá verse, ésta es una manera bastante directa de organizar un DFD potencialmente enorme en un grupo de piezas manejables. Pero existen diversas cosas que debemos añadir a esta descripción de niveles:

1. *¿Cómo saber cuántos niveles debe haber en un DFD?* La respuesta se sugirió en la sección 9.2.3: cada DFD debe tener no más de media docena de burbujas y almacenes relacionados. Así, si se ha partido un sistema grande en tres niveles, pero sus DFD de nivel más bajo aún contienen 50 burbujas cada uno, entonces falta por lo menos un nivel más. En el capítulo 11 se ofrece una alternativa de verificación, al discutir las especificaciones de proceso para las burbujas en los DFD del nivel más bajo: Si no podemos escribir una especificación de proceso razonable para una burbuja en alrededor de una página, entonces es probable que sea demasiado compleja y debiera partirse en DFD de menor nivel antes de tratar de escribir la especificación.

2. *¿Existen reglas acerca del número de niveles que debieran esperarse en un sistema típico?* En un sistema simple, probablemente se encontrarán dos o tres niveles; uno mediano tendrá generalmente de tres a seis niveles; uno grande tendrá de cinco a ocho niveles. Debe ser bastante precavido con quien le diga que todos los sistemas pueden modelarse en exactamente tres niveles; tal persona también tratará de venderle la Torre Eiffel. Por otro lado, recuerde que el número total de burbujas se incrementa exponencialmente a medida que se baja de nivel al inmediato inferior. Si, por ejemplo, cada figura tiene 7 burbujas, entonces habrá 343 en el tercer nivel, 16,807 en el quinto, y 40,353,607 en el noveno.

3. *¿Deben partirse todas las partes del sistema con el mismo nivel de detalle?* Por ejemplo, si la burbuja 2 de la figura 0 requiere tres niveles más de detalle, ¿es necesario que también la burbuja 3 tenga tres niveles más de detalle? Es decir, la figura 3; un conjunto de figuras etiquetadas como figura 3.1, 3.2, ...; y un conjunto de figuras etiquetadas 3.1.1, 3.1.2, ..., 3.2.1, 3.2.2, etc. La respuesta es "no". Algunas partes del sistema pueden ser más complejas que otras y pueden requerir uno o más niveles de partición. Por otro lado, si la burbuja 2, que existe en la figura 0, resulta primitiva, mientras que la burbuja 3 requiere de siete niveles más de partición, entonces el modelo global del sistema está ladeado y probablemente esté mal organizado. En este caso, algunas porciones de la burbuja 3 debieran separarse en una burbuja aparte o tal vez ser reasignadas a la 2.

4. *¿Cómo se muestran estos niveles al usuario?* Muchos usuarios sólo querrán ver un diagrama: un usuario ejecutivo de alto nivel pudiera querer ver tan sólo el diagrama de contexto o tal vez la figura 0; un usuario de nivel operacional pudiera querer ver sólo la figura que describe el área en la

cual está interesado. Pero si alguien quiere ver una parte más extensa del sistema, o tal vez todo, entonces tiene sentido presentar los diagramas de una manera descendente: comenzar con el diagrama de contexto y continuar hasta niveles más bajos de detalle. A menudo resulta útil tener dos diagramas juntos en el escritorio (o mostrarlos por un proyector) cuando esté haciendo esto: el diagrama en el cual está particularmente interesado y el diagrama progenitor que provee un contexto de alto nivel.

5. *¿Cómo asegurar que los niveles del DFD sean consistentes entre sí?* El asunto de la consistencia resulta ser de importancia crítica, pues los diversos niveles del DFD los desarrollan en general *distintas personas* en un proyecto real; un analista en jefe puede concentrarse en el diagrama de contexto y la figura 0, mientras que diversos analistas ayudantes trabajan en las figuras 1, 2, etc. Para asegurar que cada figura sea consistente con su figura de más alto nivel se sigue una regla sencilla: *los flujos de datos que salen y entran de una burbuja en un nivel dado deben corresponder con los que entran y salen de toda la figura en el nivel inmediato inferior que la describe.* La figura 9.22(b) muestra dos niveles de un DFD que no están balanceados.

6. *¿Cómo se muestran los almacenes en los diversos niveles?* Esta es un área donde la redundancia se introduce deliberadamente en el modelo. La regla es la siguiente: *mostrar un almacén en el nivel más alto donde primeramente sirve de interfaz entre dos o más burbujas; luego, mostrarlo de nuevo en CADA diagrama de nivel inferior que describa más a fondo (o parta más) dichas burbujas de interfase.* Así, la figura 9.23 muestra un almacén compartido por dos procesos de alto nivel: **A** y **B**; el almacén se mostraría nuevamente en las figuras del nivel inmediato inferior que describen más a fondo **A** y **B.** El corolario de esto es que los almacenes *locales,* que utilizan sólo las burbujas en una figura de menor nivel, *no* se mostrarán en niveles superiores, dado que se incluyen de manera implícita en un proceso del nivel inmediato superior.

7. *¿Cómo se realiza de hecho la partición de los DFD en niveles?* La exposición hasta aquí ha sido un tanto sutil: a pesar de que ciertamente los DFD deben *presentarse* al público usuario de una manera descendente, no es necesariamente cierto que el analista deba *desarrollarlos* así. El enfoque descendente intuitivamente es muy atractivo: puede imaginarse comenzando con el diagrama de contexto y luego desarrollando la figura 0 para ir trabajando metódicamente en forma progresiva hasta los niveles más bajos de detalle.[15] Sin embargo, como se verá en el capítulo 17,

15 También es muy atractivo para los administradores del proyecto. En una ocasión se escuchó a una administradora de un proyecto muy grande decirle a su equipo: "Quiero que todos ustedes bajen burbujeando hasta el siguiente nivel de detalle para este fin de semana".

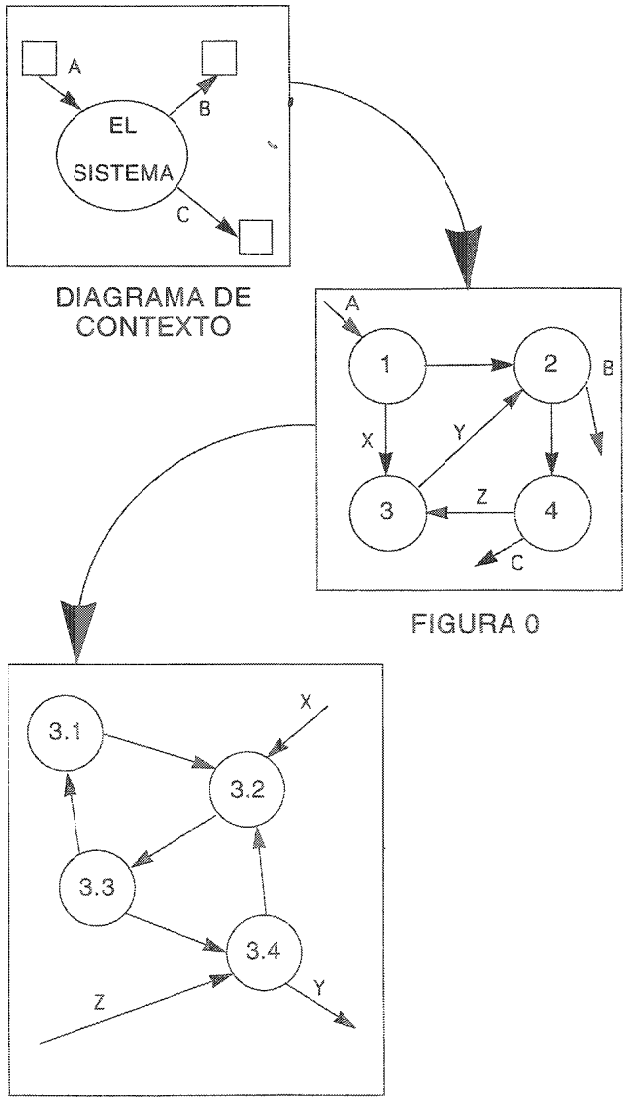


Figura 9.22(a): **Fragmento balanceado de un DFD**

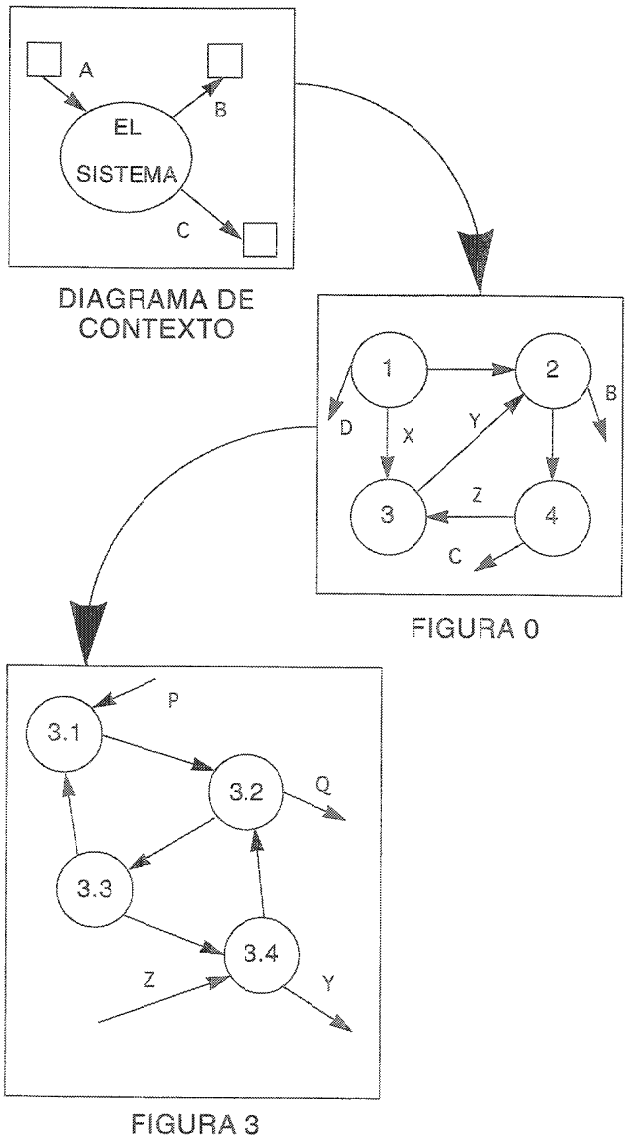


Figura 9.22(b): **Fragmento de un DFD no balanceado**

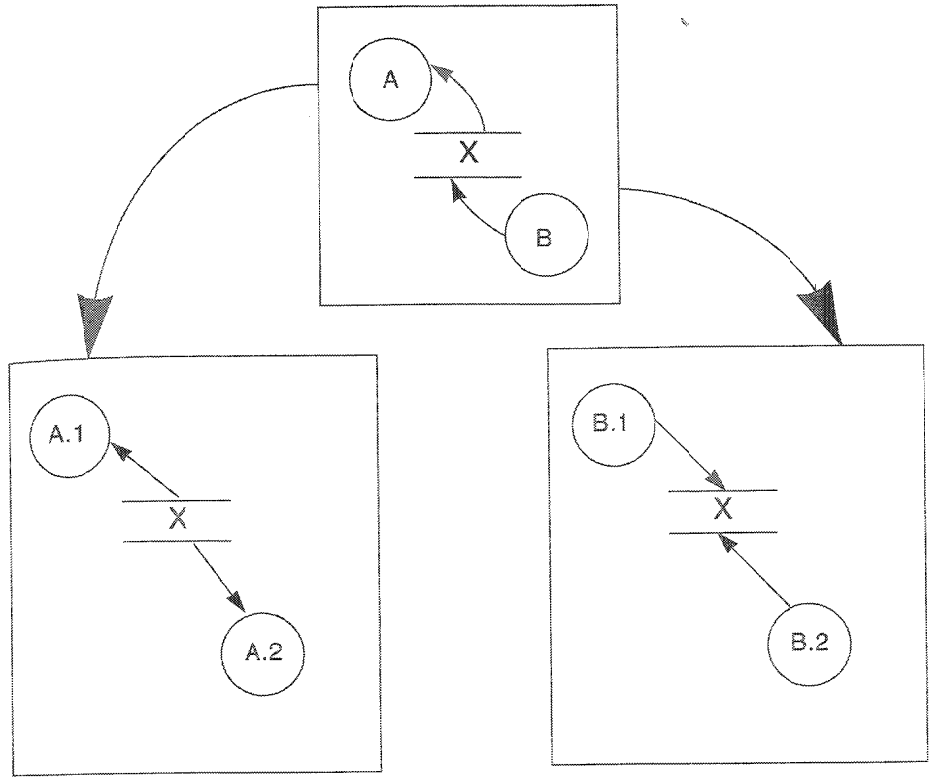


Figura 9.23: **Cómo se muestran los almacenes en niveles inferiores**

existen problemas con este enfoque; un enfoque que tiene más éxito es identificar primero los *acontecimientos* externos a los cuales debe responder el sistema y utilizarlos para crear un primer borrador del DFD. En el capítulo 20 veremos cómo es que esta primera aproximación del DFD pudiera requerir partirse *hacia arriba* (para crear DFD de mayor nivel), y *hacia abajo* (para crear DFD de menor nivel). Por ahora, es suficiente que simplemente se dé cuenta que la organización y presentación de un conjunto de DFD por niveles no necesariamente corresponde a la estrategia para desarrollar estos niveles en primer lugar.

9.4 EXTENSIONES DEL DFD PARA SISTEMAS DE TIEMPO REAL

Los flujos que se discuten a lo largo de este capítulo son flujos de *datos*, son simplemente los conductos a lo largo de los cuales viajan los paquetes de datos entre procesos y almacenes. Similarmente, las burbujas en los DFD que hemos visto hasta ahora pudieran considerarse como procesadores de datos. Para una amplia

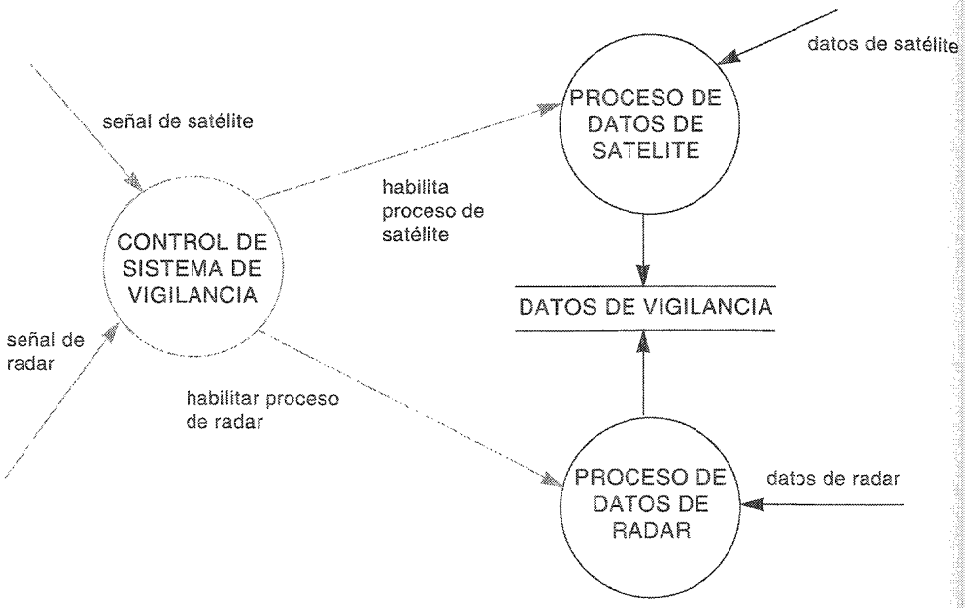


Figura 9.24: **DFD con flujos y procesos de control**

clase de sistemas, sobre todo de negocios, existen sólo dos tipos de flujos necesarios en el modelo del sistema. Pero para otra clase de sistemas, los de *tiempo real*, necesitamos alguna manera de modelar *flujos de control* (es decir señales o interrupciones). Y se requiere una manera de mostrar *procesos de control* (esto es, burbujas cuya única labor es coordinar y sincronizar las actividades de otras burbujas del DFD).[16] Se muestran gráficamente con líneas punteadas en el DFD, como lo ilustra la figura 9.24.

Un flujo de control puede imaginarse como un conducto que porta una señal binaria (esto es, está encendido o está apagado). A diferencia de otros flujos que se discuten en este capítulo, el flujo de control no porta datos con valores. El flujo de control se manda de un proceso a otro (o de algún terminador externo a un proceso) como una forma de decir: "Despierta, es hora de trabajar". Esto, desde luego, implica que el proceso ha estado "dormido" hasta la llegada del flujo de control.

16 En algunos casos, pudiera parecer apropiado incluir *almacenes de control* o *almacenes de eventos*. Son análogos al concepto de semáforo que introdujo Dijkstra en [Dijkstra, 1968]. Para mayores detalles, vea [Ward y Mellor, 1985].

Un *proceso* de control puede considerarse como una burbuja supervisora o ejecutiva, cuya labor es coordinar las actividades de otras burbujas en el diagrama; sus entradas y salidas consisten *sólo* de flujos de control. Los flujos de control salientes del proceso de control se utilizan para despertar a otras burbujas; los flujos de control entrantes generalmente indican que una de las burbujas ha terminado su labor o que se ha presentado alguna situación extraordinaria, de la cual necesita informarse a la burbuja de control. Por lo común sólo hay un proceso de control de estos en un DFD dado.

Como se indicó anteriormente, un flujo de control se utiliza para despertar un proceso normal; una vez despierto, el proceso normal procede a llevar a cabo su labor como lo describe la especificación del proceso (véase el capítulo 11). Sin embargo, el comportamiento interno de un proceso de control es diferente; aquí es donde el comportamiento dependiente del tiempo del sistema se modela con detalle. El *interior* del proceso de control se modela con un *diagrama de transición de estados*, que muestra los varios estados en los que se puede encontrar todo el sistema y las circunstancias que lo llevan a cambiar de estado. Los diagramas de transición de estados se discuten en el capítulo 13.

9.5 RESUMEN

Como vimos en este capítulo, el DFD es una herramienta sencilla pero poderosa para modelar las funciones de un sistema. El material de las secciones 9.1, 9.2 y 9.3 debiera bastar para modelar la mayoría de los sistemas de información clásicos dirigidos a los negocios. Si está trabajando con un sistema de tiempo real (por ejemplo, control de procesos, dirección de proyectiles o conmutación telefónica), serán importantes las extensiones de tiempo real que se discuten en la sección 9.4. Para más detalles sobre asuntos de tiempo real, consulte [Ward y Mellor, 1985].

Desafortunadamente muchos analistas piensan que los DFD son todo lo que necesitan conocer acerca del análisis estructurado. Si le pregunta a uno de sus colegas si está familiarizado con el análisis estructurado, es probable que diga: "Ah, sí, aprendí acerca de las burbujas y esas cosas". Por otro lado, esto es un tributo al poder de los DFD; a menudo es lo único que el analista recuerda después de leer un libro o tomar un curso de análisis estructurado. Por otro lado, es una situación peligrosa: sin las herramientas de modelado adicionales que se presentan en los capítulos siguientes, los DFD no tienen valor alguno. Aun cuando las relaciones entre datos y el comportamiento dependiente del tiempo del sistema sean triviales (lo cual es improbable), será necesario combinar los DFD con el diccionario de datos (que se trata en el capítulo 10) y las especificaciones de proceso (que se explican en el capítulo 11).

Así que no suelte este libro aún. Hay más que aprender.

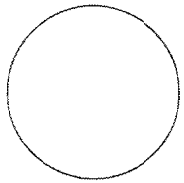
REFERENCIAS

1. Wayne Stevens, Glen Myers y Larry Constantine, "Structured Design", *IBM Systems Journal,* mayo de 1974.
2. Ed Yourdon y Larry Constantine, *Structured Design,* Nueva York: YOURDON Press, 1975.
3. Glen Myers, *Reliable Software through Composite Design,* Nueva York: Petrocelli/Charter, 1975.
4. Tom de Marco, *Structured Analysis and Systems Specification.* Englewood Cliffs, N.J.: Prentice-Hall, 1979.
5. Chris Gane y Trish Sarson, *Structured Systems Analysis: Tools and Techniques,* Englewood Cliffs, N.J.: Prentice-Hall, 1978.
6. Doug Ross y Ken Schoman, Jr., "Structured Analysis for Requirements Definition", *IEEE Transactions on Software Engineering,* enero de 1977, pp. 41-48. También reimpreso en Ed Yourdon, *Classics in Software Engineering.* Nueva York: YOURDON Press, 1979.
7. Paul Ward y Steve Mellor, *Structured Development of Real-Time Systems,* volúmenes 1-3. Nueva York: YOURDON Press, 1986.
8. Edsger W. Dijkstra, "Cooperating Sequential Processes", *Programming Languages,* F. Genuys (editor). Nueva York: Academic Press, 1968.
9. Paul Ward, "The Transformation Schema: An Extension of the Dataflow Diagram to Represent Control and Timing", *IEEE Transactions on Software Engineering,* febrero 1986, pp. 198-210.
10. Derek Hatley, "The use of Structured Methods in the Development of Large Software-Based Avionics Systems", *AIAA/IEEE 6th Digital Avionics Conference,* Baltimore, 1984.
11. M. Webb y Paul Ward, "Executable Dataflow Diagrams: An Experimental Implementatión". *Structured Development Forum,* Seattle, agosto de 1986.
12. E. Reilly y J. Brackett, "An Experimental System for Executing Real-Time Structured Analysis Models", *Proceedings of the 12th Structured Methods Conference,* Chicago, agosto de 1987.

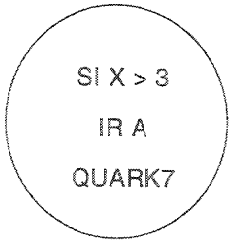
PREGUNTAS Y EJERCICIOS

1. Dé una breve descripción de un DFD. ¿Cuál es la diferencia entre un DFD y un diagrama de flujo?

2. Indique seis sinónimos de diagrama de flujo de datos.
3. ¿Para qué pueden usarse los DFD aparte de modelar sistemas de información?
4. ¿Cuáles son los cuatro principales componentes de un DFD?
5. ¿Cuáles son tres sinónimos comunes de "proceso" en un DFD?
6. ¿Existe algún significado en la elección de un círculo para un proceso? ¿Sería mejor utilizar un triángulo o un hexágono? ¿Por qué sí o no?
7. ¿Qué está mal en el siguiente proceso?



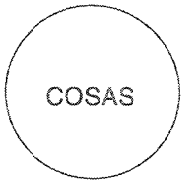
8. ¿Qué está mal en el siguiente proceso?



9. ¿Qué está mal en el siguiente proceso?

10. ¿Qué está mal en el siguiente proceso?

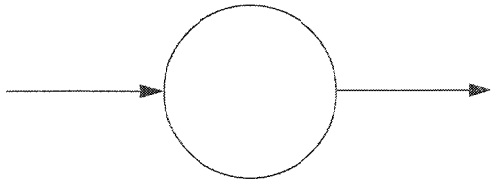
11. ¿Qué está mal en el siguiente proceso?



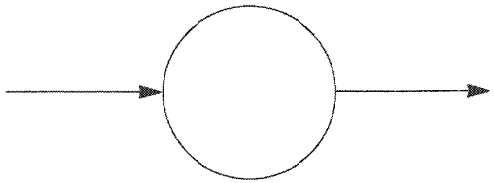
12. ¿Por qué querría un analista de sistemas dibujar un proceso que consistiera en el nombre de una persona o grupo organizacional?

13. ¿Se restringen los flujos en un DFD a mostrar el movimiento de la información? ¿Podrían mostrar el movimiento de alguna otra cosa?

14. ¿Qué tiene de mal este DFD?

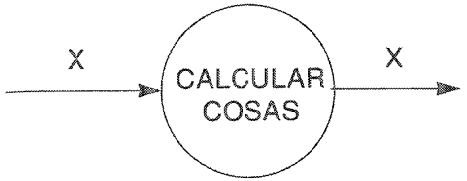


15. ¿Qué tiene de mal este DFD?



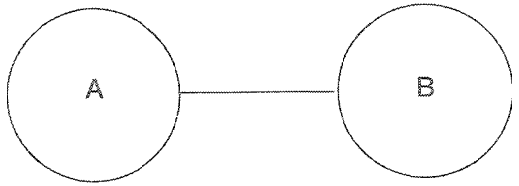
16. ¿Cómo puede tener diferentes significados el mismo fragmento de datos en un DFD? Dibuje un ejemplo de tal situación.

17. ¿Cuál es el significado del siguiente DFD?

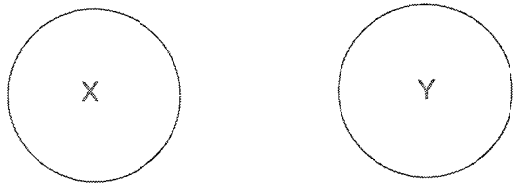


18. ¿Existe un límite para el número de entradas y salidas que puede tener un proceso en un DFD? ¿Cuál sería su reacción si viera un proceso con 100 entradas y 100 salidas?

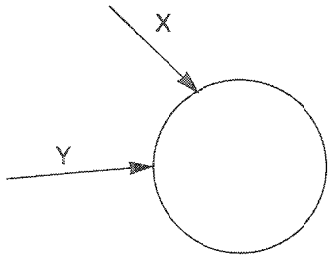
19. ¿Qué tiene de mal este DFD?

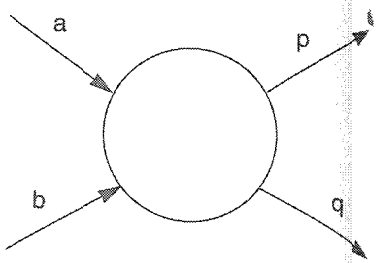


20. ¿Qué tiene de mal este DFD?

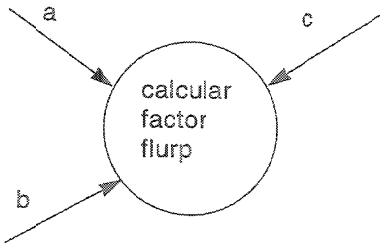


21. ¿Qué tiene de mal estos DFDs?

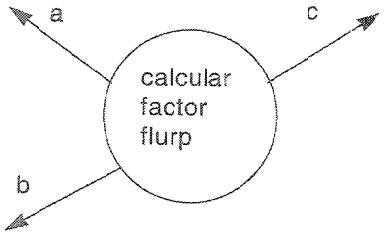




22. ¿Qué tiene de mal este DFD?

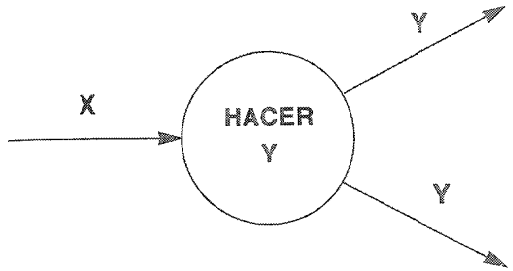


23. ¿Qué tiene de mal este DFD?

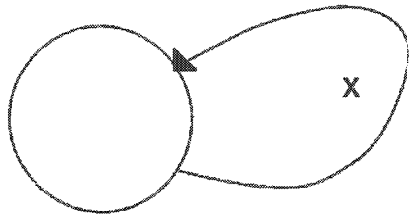


24. Dé una descripción de flujo de diálogo.

25. ¿Es válido el siguiente DFD? ¿Hay alguna manera alternativa de dibujarlo?



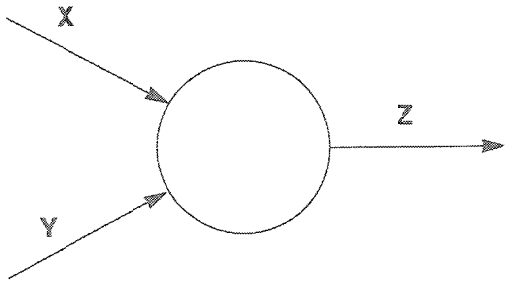
26. ¿Es válido el siguiente DFD? ¿Hay alguna manera alternativa de dibujarlo?



27. Bajo qué circunstancias esperaría ver copias del flujo de salida de un proceso?

28. ¿Por qué los DFD evitan mostrar detalles de procedimiento?

29. En el diagrama siguiente, ¿cuántos elementos X y cuántos Y se requieren para producir una salida Z?



30. ¿Qué representa un almacén en un DFD?

31. ¿Cuál es la convención para nombrar los almacenes en un DFD?

32. ¿Qué nombres alternativos puede tener un almacen? ¿Es aceptable el término archivo? ¿Por qué si o por qué no?

33. ¿Qué nombres se utilizan comúnmente para describir paquetes de información en un almacén?

34. ¿Cuáles son las cuatro razones comunes para mostrar almacenes de implantación en un DFD?

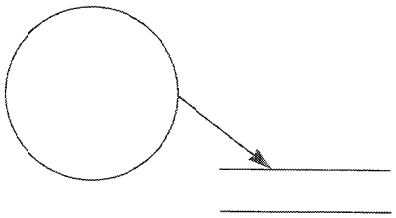
35. ¿Cree que los almacenes de implantación deban permitirse en un DFD? ¿Por qué sí o por qué no?

36. ¿Qué significa un flujo no etiquetado que entra o sale de un almacén?

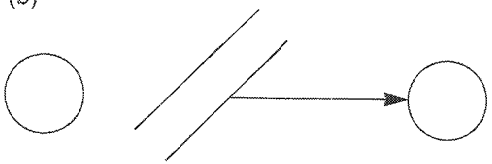
37. ¿Existe límite para el número de flujos que entran o salen de un almacén? De ser así, señale dicho límite.

38. ¿Qué tienen de mal los siguientes DFD?

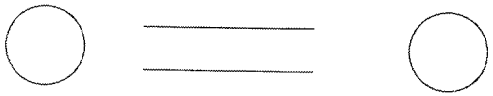
(a)



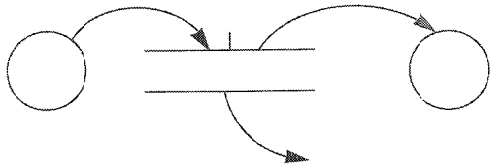
(b)



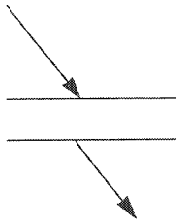
(c)



(d)

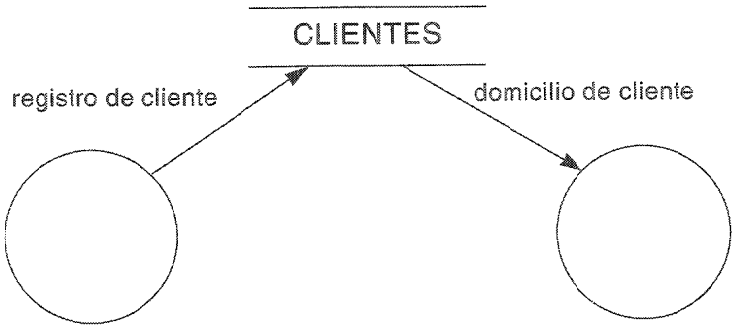


(e)



39. Cuáles son las cuatro posibles interpretaciones de un flujo de datos de un almacén a un proceso?

40. ¿Tiene sentido el siguiente DFD? ¿Por qué?

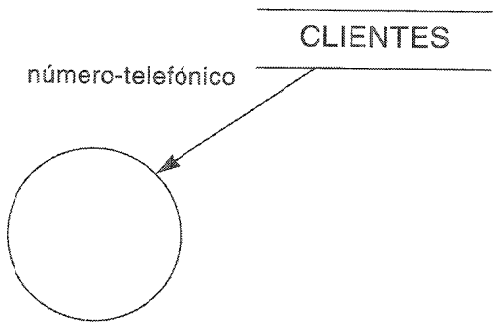


41. Dé un ejemplo de una situación donde un proceso pudiera extraer porciones de más de un registro de un almacén en un solo acceso lógico.

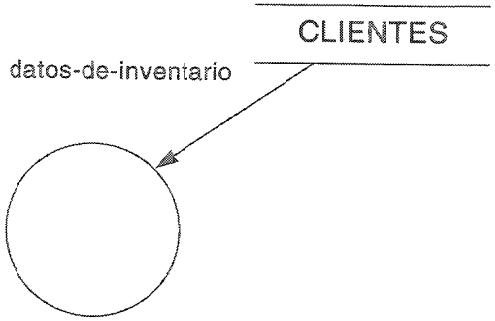
42. Dé un ejemplo de una situación donde un proceso pudiera extraer más de un paquete de información de un almacén en un solo acceso lógico.

43. ¿Puede distinguir viendo únicamente los diagramas si los siguientes DFD están correctos?

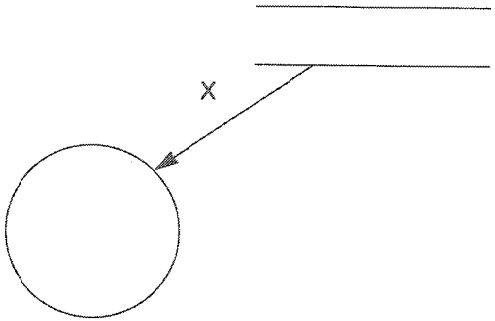
(a)



(b)



(c)

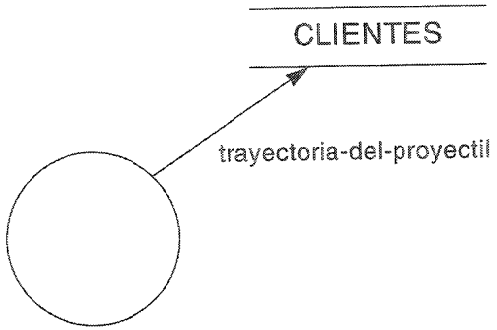


44. ¿Qué le sucede a un almacén tras habérsele retirado datos, a lo largo de un flujo, hacia un proceso? ¿Sucede esto en todos los sistemas o sólo en los de información?

45. ¿Cuáles son las principales interpretaciones de un flujo hacia un almacén?

46. ¿Cómo se muestra la *eliminación* de paquetes de información en un almacén?

47. ¿Qué tiene de mal el siguiente DFD?

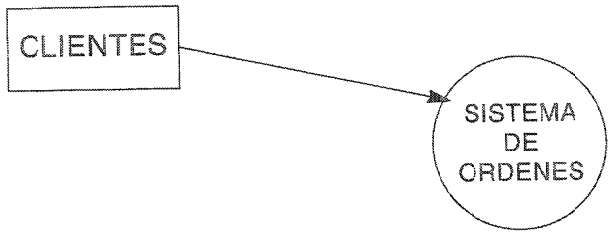


48. ¿Cuál es el propósito de mostrar un terminador en un DFD?

49. ¿Cómo debiera el analista identificar los terminadores?

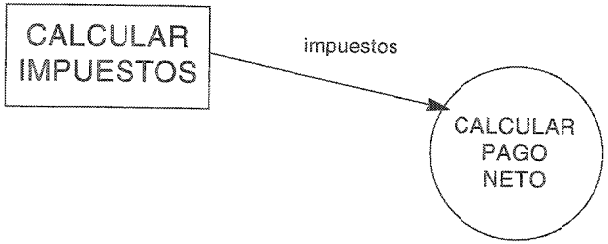
50. ¿Qué representan los flujos entre terminadores y procesos?

51. ¿Qué tienen de mal los siguientes DFDs?
(a)

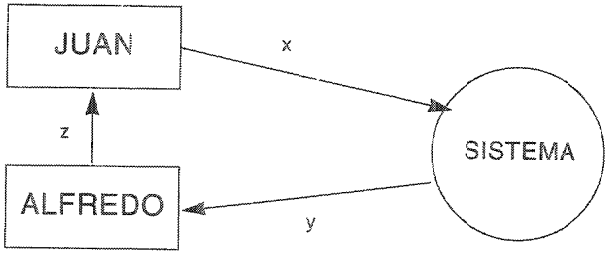


(b)

(c)



(d)



52. ¿Se le permite al analista cambiar los contenidos u organización de un terminador como parte de su proyecto? ¿Qué tal si opina rotundamente que debiera cambiarse?

53. ¿Por qué *no* deben los procesos mostrar el nombre de la persona que actualmente realiza dicha función?

54. ¿Cuál sería una buena regla para nombrar los procesos en un DFD?

55. Dé cinco ejemplos de nombres de procesos que no le gustaría ver en un DFD.

56. ¿Cómo se puede saber si es probable que el nombre escogido para un proceso tenga sentido?

57. ¿Cuál sería la mala interpretación que probablemente daría el usuario a los números en las burbujas de un DFD?

58. ¿Cómo puede saberse si es probable que un DFD dado es demasiado complejo para que lo comprenda el usuario?

59. ¿Cuánto se debe insistir en las reglas de complejidad? ¿Deben permitirse excepciones? ¿Por qué?

60. ¿Por qué resulta a menudo necesario redibujar varias veces un DFD?

61. ¿Cuáles son los aspectos principales para determinar si un DFD será estéticamente agradable? ¿Cree que pudieran expresarse como estándares?

62. ¿Prefieren sus colegas burbujas u óvalos para los procesos? ¿Qué prefiere usted?

63. ¿Cree que los flujos entre procesos deban mostrarse como curvas o como rectas? ¿Se le ocurren ventajas o desventajas en cualquiera de éstas? ¿Es importante esto?

64. ¿Qué es un sumidero infinito?

65. ¿Que es la generación espontánea de burbujas en un DFD? ¿Por qué debe evitarse en un DFD típico?

66. ¿Por qué son peligrosos los flujos y procesos no etiquetados?

67. ¿Por qué son generalmente erróneos los almacenes de únicamente lectura o únicamente escritura en un DFD?

68. ¿Por qué son importantes los DFD por niveles en el modelo de un sistema?

69. ¿Cuántos niveles de un DFD debiera el analista esperar ver en un sistema grande típico? ¿Puede sugerir un límite superior para el número de niveles en un DFD?

70. En un DFD por niveles:

    (a) ¿Cómo llamaría a las burbujas "hijas" debajo de la burbuja 2.3?

    (b) ¿Y a la *figura* en la que aparece la burbuja 2.3?

    (c) ¿Cuántas figuras de *mayor* nivel hay por encima de la figura en la que aparece la Burbuja 2.3? ¿Cómo se llaman?

71. ¿Es necesario que todas las partes de un sistema se dividan hasta el mismo nivel de detalle? ¿Por qué?

72. Suponga que alguien le mostrara un DFD en el cual la burbuja 1 estuviera dividida en dos niveles inferiores, y la 2 en 17 niveles inferiores. ¿Cuál sería su reacción? ¿Qué podría recomendar?

73. ¿Qué significa balancear, en el contexto de este capítulo? ¿Cómo puede darse cuenta si un DFD está balanceado?

74. ¿Por qué no está balanceada la figura 9.22(b) de este capítulo?

75. ¿Cuál es la regla a seguir para mostrar almacenes en los diferentes niveles de un DFD?

76. ¿Qué es un almacén local? ¿Cuáles son las reglas para mostrar un almacén local en un DFD por niveles?

77. Proyecto de investigación: ¿Cuál es la relación entre la regla para mostrar un almacén local y el concepto de diseño orientado a objetos? Para más información acerca de esto, vea los capítulos 19 y 20.

78. ¿Qué problemas pudieran asociarse con el *desarrollo* de un conjunto de DFD por niveles en forma descendente (en comparación con la *lectura* de un conjunto de DFD que ya exista de manera descendente)?

79. ¿Qué es un flujo de control? ¿En que difiere de un flujo de datos?

80. ¿Qué es un proceso de control? ¿En qué difiere de un proceso normal en un DFD?

81. ¿Qué es un almacén de control? ¿En qué difiere de un almacén normal en un DFD?

82. Dibuje un DFD para modelar la siguiente receta de *Fruits de Mer au Riz* (calamares surtidos con arroz), tomada de *The New York Times 60-Minute Gourmet,* de Pierre Franey (Nueva York: TIMES Books, 1979)

"1. Para empezar, prepare y mida las cantidades de todos los ingredientes para preparar el arroz. Para ahorrar tiempo, pique una taza extra de cebollas y dos dientes extra de ajo para la mezcla de mariscos y sepárelos.

2. Caliente dos cucharadas de aceite para el arroz en una sartén y añada 1/4 de taza de cebolla, pimiento verde y un diente de ajo, y déjelo al fuego hasta que se aje. Añada azafrán y cuézalo dos minutos más.

3. Añada el arroz, agua, sal y pimienta y tápelo bien. Déjelo cocinarse durante 17 minutos o hasta que se ablande el arroz. Mientras se cuece el arroz, prepare los mariscos. Recuerde retirar el arroz del fuego cuando esté listo. Puede dejarlo cubierto durante varios minutos sin perjuicio.

4. Caliente 1/4 de taza de aceite y añada la taza de cebolla y los dos dientes de ajo. Revuelva y cocine hasta que se aje. Añada pimiento rojo, jitomate, vino y orégano. Añada sal y pimienta. Tape y cocine durante 10 minutos.

5. Añada los calamares y vuelva a tapar. Cocine durante unos tres minutos y añada los camarones, los ostiones, y sal y pimienta al gusto. Tape y cocine cinco minutos más.

83. Dibuje un DFD para la siguiente receta de *Coquille St. Jacques Meuniere* (ostiones fritos en mantequilla), tomada de *The New York Times 60-Minute Gourmet,* de Pierre Franey (Nueva York: TIMES Books, 1979):

"Algo que se debe recalcar cien veces es la organización. Antes de cocinar, pique lo que se tenga que picar y mida lo que se tenga que medir. Saque todas las ollas y sartenes que se vayan a ocupar, en este caso dos cazuelas (una para los ostiones y la otra para los jitomates) y una sartén (para las papas).

1. Vacíe los ostiones en un plato y añada la leche, revolviendo para cubrir. Deje reposar un rato.

2. Ponga la harina en otro plato y añada sal y pimienta al gusto. Revuelva bien. Deje escurrir los ostiones. Cúbralos de harina y póngalos en una coladera grande. Sacúdalos para quitarles el exceso de harina. Sepárelos sobre una hoja de papel aluminio o papel encerado para que no se adhieran unos a otros.

3. Los ostiones deben cocerse a fuego alto evitando que se junten. Caliente tres cucharadas de aceite y una de mantequilla en una cazuela grande. Cuando la mezcla esté muy caliente, pero no humeante, añada la mitad de los ostiones, sacudiéndolos y volteándolos para que se cuezan rápida y uniformemente hasta dorarse

4. Use una espátula con ranuras para transferir los ostiones a un plato caliente. Añada las dos cucharadas restantes de aceite a la cazuela y, cuamdo esté caliente, añada el resto de los ostiones, sacudiéndolos y volteándolos como se hizo anteriormente. Cuando estén dorados, transfiéralos al plato junto con los demás. Limpie la cazuela con una toalla desechable, añada el resto de la mantequilla y cocine hasta que adquiera un color castaño. Póngaselo a los ostiones. Luego póngales jugo de limón y perejil picado."

84. Dibuje un DFD para la siguiente receta de *Omelette Pavillon* (Omelette con pollo, jitomate y queso), tomado de *The New york Times 60-Minute Gourmet,* de Pierre Franey (Nueva York: TIMES Books, 1979):

"Antes de iniciar, tenga a mano un tazón por cada omelette que piense preparar, y en cada uno ponga tres huevos. Añada sal y pimienta al gusto y dos cucharaditas de crema espesa. También puede ir batiendo los huevos para hacer todo más rápido.

1. Caliente dos cucharadas de mantequilla en una sartén y añada la harina. Revuelva con un molinillo hasta que la mezcla esté uniforme. Añada el caldo de pollo y ponga al fuego sin dejar de revolver rápidamente. Añada la crema y deje hervir. Manténgalo así durante 10 minutos.

2. Mientras tanto, caliente otra cucharada de mantequilla en una sartén y añada la cebolla. Cueza, sin dejar de revolver, hasta que se aje, y añada los jitomates, el tomillo, hojas de laurel, sal y pimienta. Deje cocer a fuego lento, revolviendo de vez en cuando, durante 10 minutos.

3. Caliente otra cucharada de mantequilla y añada el pollo. Cueza, sin dejar de revolver, durante 30 segundos. Añada 3 cucharadas de la salsa de crema. Déjelo al fuego hasta que hierva, y luego retírelo. Póngalo a un lado.

4. A la salsa restante añádale la yema de huevo y revuelva hasta obtener una mezcla uniforme. Añada sal y pimienta al gusto y el queso suizo rayado. Caliente, revolviendo, hasta que se funda el queso. Póngalo aparte.

5. Bata los huevos con sal y pimienta. Añada seis cucharadas de salsa de tomate. Caliente las tres cucharadas restantes de mantequilla en una sartén para omelettes o en una cazuela de teflón y, cuando esté caliente, añada los huevos. Cueza, revolviendo hasta que cuaje abajo pero siga estando húmedo en el centro. Póngale el pollo en el centro y añada el resto de la salsa de tomate. Rápidamente saque el omelette y póngalo en un refractario.

6. Cubra el omelette con el resto de la salsa de crema y espolvoréelo con queso parmesano rayado. Hornee hasta que se dore.

10 EL DICCIONARIO DE DATOS

Los diccionarios son como los relojes: el peor es mejor que no tener ninguno y del mejor no puede esperarse que sea muy exacto.
Sra. Priozzi, *Anécdotas de Samuel Johnson*, 1786

En este capítulo se aprenderá:

1. Por qué se necesita un diccionario de datos en un proyecto de desarrollo de sistemas.
2. La notación de las definiciones de los diccionarios de datos.
3. Cómo debe presentarse el diccionario de datos al usuario.
4. Cómo realizar un diccionario de datos.

La segunda herramienta de modelado importante que discutiremos es el *diccionario de datos*; aunque no tiene la presencia y el atractivo gráfico de los DFD, los diagramas de entidad-relación y los diagramas de transición de estados, es crucial. Sin los diccionarios de datos, el modelo de los requerimientos del usuario no puede considerarse completo; todo lo que se tendría es un borrador rudimentario, una "visión del artista" del sistema.

La importancia del diccionario de datos a menudo les pasa de largo a muchos adultos, pues no han utilizado un diccionario durante 10 o 20 años. Trate de recordar sus días en la primaria, cuando constantemente se le asediaba con nuevas pala-

bras en sus tareas. Recuerde también sus cursos de lenguas extranjeras, sobre todo los que requerían que leyera libros y revistas. Sin un diccionario, se habría perdido. Lo mismo sucede con un diccionario de datos en el analisis de sistemas: sin él se extraviará y el usuario no podrá estar seguro de que entendió los detalles de la aplicación.

El diccionario de datos de frases casi se autodefine. El diccionario de datos es un listado organizado de todos los datos pertinentes al sistema, con definiciones precisas y rigurosas para que tanto el usuario como el analista tengan un entendimiento común de todas las entradas, salidas, componentes de almacenes y cálculos intermedios. El diccionario de datos define los datos haciendo lo siguiente:

- Describe el *significado* de los flujos y almacenes que se muestran en los DFD.
- Describe la *composición* de agregados de paquetes de datos que se mueven a lo largo de los flujos, es decir, paquetes complejos (por ejemplo el domicilio de un cliente), que pueden descomponerse en unidades más elementales (como ciudad, estado y código postal).
- Describen la *composición* de los paquetes de datos en los almacenes.
- Especifica los *valores* y *unidades* relevantes de piezas elementales de información en los flujos de datos y en los almacenes de datos.
- Describe los detalles de las *relaciones* entre almacenes que se enfatizan en un diagrama de entidad-relación. Este aspecto del diccionario de datos se discutirá con más detalle en el capítulo 12, después de introducir la notación de entidad-relación.

10.1 LA NECESIDAD DE LA NOTACION EN EL DICCIONARIO DE DATOS

En la mayoría de los sistemas reales con los que se trabaja, los paquetes, o elementos de datos, serán lo suficientemente complejos como para que se necesite describirlos en términos de otras cosas. Los elementos complejos de datos se definen en términos de elementos más sencillos, y los sencillos en términos de los valores y unidades legítimos que pueden asumir.

Imagine, por ejemplo, la forma en la que respondería a las siguientes preguntas de un marciano (que es el concepto que muchos usuarios tienen del analista) acerca del significado del **nombre** de una persona:

Marciano: "Bien, ¿qué es esto llamado nombre?"

Usted (encogiéndose impacientemente de hombros): "Pues, usted sabe, es sólo un nombre, quiero decir, este, bueno, es lo que nos llamamos unos a otros."

Marciano (confundido): "¿Significa eso que los llama de distinto modo cuando está contento y cuando está enojado?"

Usted (un tanto sorprendido de la ignorancia de este extraterrestre): "No, claro que no. Un nombre es el mismo siempre. Un nombre de persona lo distingue de otras personas."

Marciano (entendiendo de repente): "¡Ah! Ya entiendo. Hacemos lo mismo en mi planeta. Mi nombre es 3.14159265358979323846264 3."

Usted (incrédulo): "Pero eso es un número, no un nombre."

Marciano: "Y es un muy buen nombre, me enorgullezco de él. Nadie tiene algo parecido."

Usted: "¿Pero cuál es su nombre y cuál su apellido? O 3 es su nombre y el resto su apellido?

Marciano: "¿Qué es todo esto de nombre y apellido? No entiendo. Tengo un solo nombre y siempre es el mismo."

Usted: "Pues no funcionan así las cosas aquí. Tenemos un nombre, un apellido, y en ocasiones un segundo nombre también."

Marciano: "¿Significa eso que usted puede llamarse 23 45 99?"

Usted: "No. No permitimos números en nuestros nombres. Sólo puede usar los caracteres alfabéticos de la A a la Z."

Como podrá imaginar, la conversación podría continuar durante mucho tiempo. Puede pensar que el ejemplo es exagerado porque rara vez nos encontraremos con marcianos que no tengan el concepto del significado de un nombre. Pero no está muy alejado de las discusiones que se suscitan (o que debieran suscitarse) entre el analista y el usuario, en las cuales pudieran surgir las siguientes preguntas:

- ¿Debe tener todo mundo un nombre? Qué tal el personaje "Sr. T" de la popular serie de televisión "Los cuatro fantásticos"?
- ¿Qué pasa con los signos de puntuación en los apellidos de las personas, por ejemplo "D'Arcy"?
- ¿Se permiten los segundos nombres abreviados, por ejemplo, "Juan X Jasso?"
- ¿Existe una longitud *mínima* para el nombre de una persona? Por ejemplo, ¿es legal el nombre "X Y"? (Es fácil imaginarse que pudiera confundir a muchos sistemas de cómputo, pero ¿existe alguna razón legal o de negocios por la cual una persona no pudiera llamarse X y apellidarse Y?)

- ¿Cómo debemos tratar los sufijos que a veces siguen al apellido? Por ejemplo, se supone que el nombre "Juan Jasso Jr." es legítimo, pero ¿se considera el Jr. como parte del apellido, o en una categoría aparte? Y si está en una nueva categoría, ¿no debiéramos permitir también digitos, como por ejemplo, Samuel Sosa 3º?

Nótese, por cierto, que ninguna de estas cuestiones tiene algo que ver con la forma en la que se almacenará la información en la computadora; simplemente estamos tratando de determinar, como cuestión de política de negocios, lo que constituye un nombre válido.[1]

Como se podrá imaginar, se vuelve algo tedioso describir la composición de los elementos de datos en una forma narrativa. Necesitamos una notación concisa y compacta, así como un diccionario normal tiene notación compacta y concisa para definir el significado de las palabras ordinarias.

10.2 NOTACION DEL DICCIONARIO DE DATOS

Existen muchos esquemas de notación comunes utilizados por el analista de sistemas. El que se muestra a continuación es de los más comunes y utiliza varios símbolos sencillos:

=	está compuesto de
+	y
()	optativo (puede estar presente o ausente)
{}	iteración
[]	seleccionar una de varias alternativas
**	comentario
@	identificador (campo clave) para un almacén
|	separa opciones alternativas en la construcción

Por ejemplo, se puede definir el **nombre** para nuestro amigo marciano así:

nombre = **título de cortesía + nombre + (segundo nombre) + apellido**

título de cortesía = [Sr. | Srita. | Sra. |Dr. | Profesor]

nombre = **{carácter legal}**

1 Por otro lado, es probable que la política de negocios actual haya tenido una fuerte influencia de los sistemas de cómputo que la organización ha estado usando durante los últimos 30 años. Hace 50 años se hubiera considerado excéntrico a alguien que se hiciera llamar "Jua5n So7to", pero probablemente hubiera sido aceptado por la mayoría de las organizaciones, porque los nombres se transcribían en pedazos de papel por manos humanas. Los sistemas puramente computacionales (como la mayoría de los de uso actual) tienen muchos más problemas con nombres no estándar como éste.

segundo nombre = **{carácter legal}**

apellido = **{carácter legal}**

carácter legal = [A-Zla-zl0-9l'l-l l]

Como puede apreciarse, los símbolos parecen algo matemáticos y pudiera preocuparse porque sea demasiado complicado de entender. Sin embargo, como veremos pronto, la notación es bastante fácil de leer. La experiencia de miles de proyectos de procesamiento de datos y varias decenas de miles de usuarios nos ha mostrado que la notación, además, es bastante entendible para casi todos los usuarios *si se presenta de manera correcta*; discutiremos esto en la Sección 10.3

10.2.1 Definiciones

La definición de un dato se introduce con el símbolo "=". En este contexto, el "=" se lee: "se define como", o "se compone de", o simplemente "significa". Por ello, la notación

A = B + C

puede leerse de las siguientes maneras:

- Cuando digamos **A**, queremos decir una **B** y una **C**
- **A** se compone de **B** y **C**
- **A** se define como **B** y **C**

Para definir por completo un dato, nuestra definición debe incluir lo siguiente:

- El *significado* del dato dentro del contexto de la aplicación de este usuario. Por lo común se ofrece como comentario utilizando la notación "**"
- La *composición* del dato, si se compone de partes elementales con significado.
- Los *valores* que puede tomar el dato, si es un dato elemental que no puede descomponerse más.

Así, si estamos construyendo un sistema médico que siga la evolución de los pacientes, podrían definirse los términos **peso** y **estatura** de la siguiente manera:

peso = *peso del paciente al ser admitido al hospital*
unidades: kilogramos; gama 1-200

estatura = *estatura del paciente al ser admitido al hospital*
unidades: centímetros; escala: 20-200

Nótese que hemos descrito las *unidades* relevantes y la *escala* relevante entre un par de caracteres "*". Repetimos que esto es un convenio de notación que muchas organizaciones encuentran adecuado, pero que puede cambiarse de ser necesario.

Además de las unidades y la escala, podría requerirse la especificación de la precisión de la medición del dato. Para datos tipo precio, por ejemplo, es importante indicar si los valores se expresarán en moneda entera o redondeados al último centavo, etc. En muchas aplicaciones científicas y de ingeniería es importante indicar el número de dígitos significativos en el valor de los datos.

10.2.2 Elementos de datos básicos

Las partes elementales de los datos son aquellas para las cuales ya no existe una descomposición con significado dentro del contexto del ambiente del usuario. Esto usualmente es una cuestión de aplicación y es algo que se debe explorar cuidadosamente con el usuario. Por ejemplo, hemos visto en la exposición anterior que el término nombre puede descomponerse en **nombre**, **segundo nombre, apellido** y **título de cortesía**. Pero tal vez en algunos ambientes de usuario no se requiere tal descomposición, ni sea relevante, ni tenga significado (esto es, en ambientes donde los términos **apellido**, segundo nombre, *etc., no tengan significado para el usuario*).

Cuando se han identificado los datos elementales, deben introducirse al diccionario de datos. Como se indicó anteriormente, el diccionario de datos debe proporcionar una breve narrativa, encerrada entre caracteres "*", que describa el *significado* del término en el contexto del usuario. Desde luego, habrá terminos que se definan solos, es decir, cuyo significado es universal para todos los sistemas de información, o donde el analista pudiera estar de acuerdo en que no se necesita aclarar más. Por ejemplo, los siguientes pudieran considerarse términos que se autodefinen en un sistema que maneja información sobre personas:

estatura actual
peso actual
fecha de nacimiento
sexo
teléfono particular

En estos casos no se necesita un comentario narrativo; muchos analistas usan la notación "**" para indicar "sin comentarios" cuando el dato se defina solo. Sin embargo, es importante especificar los valores y unidades de medida que los datos elementales pueden tomar. Por ejemplo:

peso actual =
**
unidades: libras; escala: 1-400

estatura actual =
**
unidades: pulgadas; escala: 1-96

fecha de nacimiento =
**
Unidades: días a partir del 1º de enero de 1900; escala: 0-36500

sexo =
valores: [M | F]

10.2.3 Datos opcionales

Un dato opcional, como la frase implica, es aquel que puede estar o no presente en un dato compuesto. Existen muchos ejemplos de datos opcionales en sistemas de información:

- El nombre de un cliente pudiera no incluir un segundo nombre
- El domicilio de un cliente pudiera incluir o no información secundaria, como el número de departamento.
- El pedido de un cliente pudiera contener el domicilio al que se tiene que mandar la cuenta, el domicilio al que hay que hacer el envío, o ambos.

Las situaciones de este tipo deben verificarse con cuidado con el usuario y deben documentarse precisamente en el diccionario de datos. Por ejemplo, la notación

domicilio de cliente = (domicilio de envío) + (domicilio para cuenta)

significa, literalmente, que el domicilio del cliente pudiera consistir en:

- sólo un domicilio de envío

o bien

- sólo un domicilio para enviar cuentas

o bien

- un domicilio de envío y uno para cuentas

o bien

- *ninguno* de los dos

Esta última posibilidad es dudosa. Es mucho más probable que el usuario realmente quiere decir que el domicilio debe consistir en uno u otro o ambos. Esto pudiera expresarse de la siguiente manera:

domicilio del cliente = [domicilio de envío | domicilio para cuentas | domicilio de envío + domicilio para cuentas]

Podría también argumentarse que, en un negocio por correspondencia, *siempre* se requiere un domicilio de envío a donde se deberá mandar la mercancía solicitada por el cliente; un segundo domicilio para el envío de la cuenta es opcional (por ejemplo, el departamento de contabilidad del cliente). Así, es posible que la verdadera política del usuario esté expresada por

domicilio del cliente = domicilio de envío + (domicilio para cuentas)

Desde luego, la única manera de saber esto es pedirle al usuario que explique con cuidado las implicaciones de las diferentes notaciones que se mostraron.[2]

10.2.4 Iteración

La notación de iteración se usa para indicar la ocurrencia *repetida* de un componente de un dato. Se lee como "cero o más ocurrencias de". Así, la notación

solicitud = nombre del cliente + domicilio de envío + {artículo}

significa que la solicitud *siempre* debe contener un nombre de cliente, un domicilio de envío, y también *cero o más ocurrencias* de un **artículo.** Así, pudiéramos estar tratando con un cliente que pide un artículo, o dos, o algún comprador compulsivo que decide ordenar 397 artículos diferentes.[3]

En muchas situaciones reales, el usuario querrá especificar los límites inferior y superior de la iteración. Tal vez, en el ejemplo anterior, el usuario señale que no tiene sentido que un cliente haga un pedido de cero artículos; debe haber por lo menos uno. Podría también especificarse un límite superior; quizá, se permitirán cuando más 10 artículos. Puede indicarse esto de la siguiente forma:

2 Existe una posibilidad que pudiera explicar la ausencia tanto del domicilio de envío como del de cobro en un pedido de un cliente: el cliente que llega personalmente para comprar un artículo y llevárselo en el acto. Es posible que se le quisiera identificar explícitamente (definiendo un nuevo **dato en persona**, que tendría valor de verdadero o falso) ya que 1) los clientes que llegan en persona pudieran requerir un trato distinto (por ejemplo, sus pedidos estarían exentos de cargos de envío) y 2) es una buena forma de asegurarse de que la ausencia del dato de domicilio no fue por error.

3 Tenga en mente nuevamente que estamos definiendo el significado intrínseco de *negocios* de un dato dado sin referirnos a la tecnología usada para implantarlo. A la larga, por ejemplo, es probable que los diseñadores pregunten sobre un límite superior razonable o el número de artículos diferentes que puede contener un mismo pedido. "Para poder lograr una labor eficiente de nuestro sistema SUPERMARAVILLA de administración de bases de datos, tendremos que restringir el número de artículos a 64. Es poco probable, de todos modos, que alguien pida más de 64 y, si lo hacen, pueden sencillamente colocar varios pedidos". Además, el usuario pudiera tener sus propias limitaciones, basadas en las formas escritas o los reportes impresos con los que trabaja; esto es parte del modelo de implantación del usuario, que se tratará en el capítulo 21.

solicitud = nombre del cliente + domicilio de envío + 1{artículo}10

Es correcto especificar *sólo* el límite inferior, *sólo* el límite superior, *ambos*, o *ninguno*. Así que se permite cualquiera de los siguientes:

a = 1{**b**}

a = {**b**}10

a = 1{**b**}10

a = {**b**}

10.2.5 Selección

La notación de *selección* indica que un dato consiste en exactamente *un* elemento de entre un conjunto de opciones alternativas. Las opciones se encierran en corchetes "[" y "]", y se separan por una barra vertical "|". Como ejemplos típicos tenemos:

sexo = [Femenino | Masculino]

tipo de cliente = [Gobierno | Industria | Universidad | Otro]

Es importante revisar las opciones de selección con el usuario para asegurarse de cubrir todas las posibilidades. En el último ejemplo, el usuario pudiera tender a concentrar su atención en los clientes "gobierno", "industria" y "universidad", y podría requerir un recordatorio de que existen clientes de la categoría de "ninguno de los anteriores".

10.2.6 Alias

Un *alias*, como el término implica, es una alternativa de nombre para un dato. Esto es una ocurrencia común cuando se trata con diversos grupos de usuarios en diferentes departamentos o ubicaciones geográficas (y a veces con diferentes nacionalidades e idiomas), que insisten en utilizar distintos nombres para decir lo mismo. El alias se incluye en el diccionario de datos para que esté completo, y se relaciona con el nombre primario u oficial del dato. Por ejemplo:

comprador = *alias de cliente*

Nótese que la definición de comprador *no* muestra su composición (es decir, no muestra que consiste en nombre, domicilio, número telefónico, etc.). Todos estos detalles deben darse *sólo* para el nombre primario del dato, para minimizar la redundancia en el modelo.[4]

4 Tal vez pueda ignorar este consejo si está utilizando un paquete computarizado de generación de diccionarios de datos que pueda manejar y controlar la redundancia; sin embargo, esto es poco común. Lo crucial es recordar que si se cambia la definición de un dato primario (por ejemplo, si se decide que la definición de cliente ya no debe incluir número telefónico), entonces el cambio se debe aplicar a todos los alias también.

Aun cuando el diccionario de datos relaciona correctamente los alias con el nombre primario de los datos, debe evitarse el uso de alias hasta donde sea posible. Esto se debe a que los nombres de datos se suelen ver primero, y son más visibles para todos los usuarios en los DFDs, en donde *pudiera no ser tan obvio que* **comprador** y **cliente** *sean alias.* Es mejor, de ser posible, lograr que todos los usuarios se pongan de acuerdo en un solo nombre.[5]

10.3 COMO MOSTRAR EL DICCIONARIO DE DATOS AL USUARIO

El diccionario de datos lo crea el analista durante el desarrollo del modelo del sistema, pero el usuario debe ser capaz de leerlo y entenderlo para poder verificar el modelo. Esto plantea unas preguntas obvias:

- ¿Podrán los usuarios entender la notación del diccionario de datos?
- ¿Cómo podrían los usuarios verificar que el diccionario está completo y correcto?
- ¿Cómo se crea el diccionario?

La cuestión de la aceptación por el usuario de la notación del diccionario puede despistar en la mayoría de los casos. Es cierto, la notación del diccionario se ve algo matemática; pero, como se ha visto, el número de símbolos que el usuario debe aprender es muy pequeño. Los usuarios están acostumbrados a la variedad de notaciones formales en su trabajo y vida personal; considere, por ejemplo, la notación musical, que es mucho más compleja:

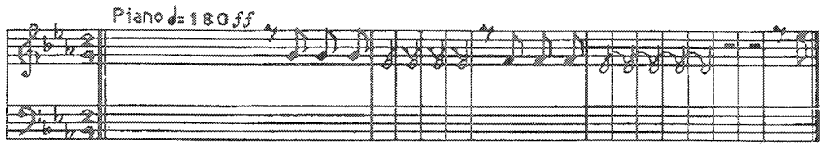


Figura 10.1: **Notas musicales**

Similarmente, la notación de los juegos de canasta, ajedrez, y varias actividades más es cuando menos igual de compleja que la del diccionario de datos que se muestra en este capítulo.

5 Una alternativa sería anotarle algo al flujo *en el diagrama de flujo de datos* para indicar que es alias de otra cosa; por ejemplo, se podría agregar un asterisco al final a los nombres que son alias. De esta forma, la notación **comprador*** podría usarse para indicar que comprador es alias de otra cosa. Pero incluso esto es molesto.

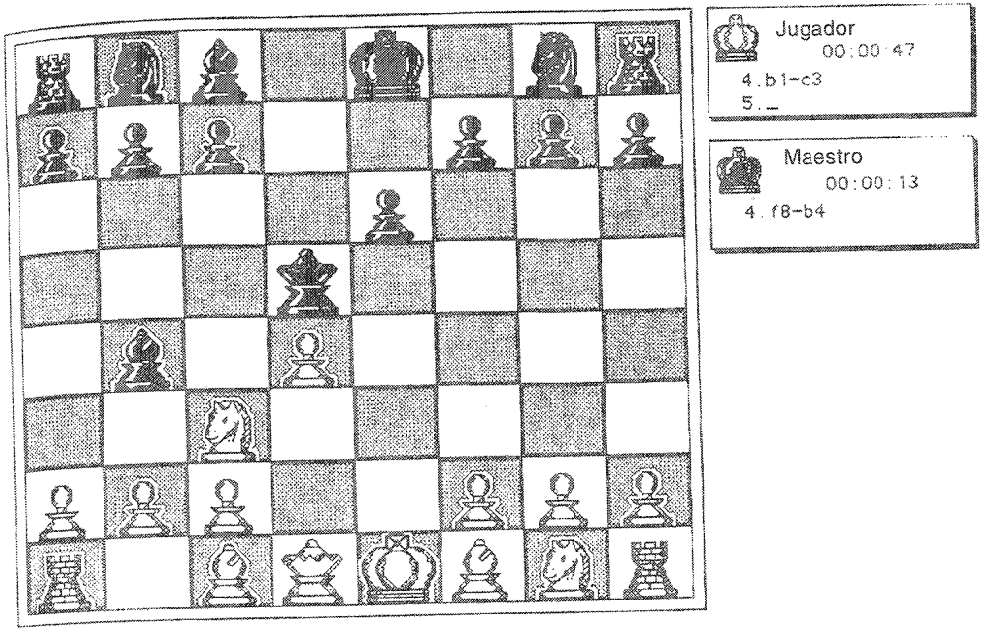


Figura 10.2: **Notación de ajedrez**

La cuestión de la verificación del diccionario de datos por el usuario lleva generalmente a esta pregunta: "¿Deben los usuarios leer en detalle todo el diccionario para asegurarse de que está correcto?" Es difícil imaginar que algún usuario estuviera dispuesto a hacer esto. Es más probable que el usuario verifique que el diccionario es correcto *en conjunto con el DFD, el diagrama de entidad-relación y el diagrama de transición de estados o la especificación del proceso que esté leyendo.*

Hay varios detalles acerca de la corrección del sistema que el analista puede hacer por su cuenta, sin ayuda del usuario: puede asegurarse de que el diccionario esté completo y sea consistente y no contradictorio. Así que puede examinar el diccionario por sí solo y preguntar lo siguiente:

- ¿Se ha definido en el diccionario cada flujo del DFD?
- ¿Se han definido todos los componentes de los datos en el diccionario?
- ¿Se ha definido más de una vez algún dato?
- ¿Se ha utilizado la notación correcta para todas las definiciones del diccionario de datos?

- ¿Hay elementos de datos en el diccionario que no estén relacionados con los DFD, los diagramas de entidad-relación o los de transición de estado?

10.4 IMPLANTACION DEL DICCIONARIO DE DATOS

En un sistema mediano o grande, el diccionario de datos puede representar una cantidad formidable de trabajo. No es fuera de lo común ver un diccionario de datos con varios miles de entradas, e incluso un sistema relativamente sencillo tendrá varios cientos de entradas. Así que se debe pensar en cómo desarrollar el diccionario de datos, porque es probable que la tarea sea demasiado para el analista.

El enfoque más fácil es hacer uso de una computadora para introducir definiciones al diccionario, verificar que estén completas y consistentes, y producir reportes apropiados. Si su organización está utilizando cualquier sistema moderno de administración de bases de datos (por ejemplo, IMS, ADABAS, TOTAL, IDMS), ya dispone de una ayuda para el diccionario. En este caso, debiera aprovecharla y utilizarla para construir el diccionario de datos. Sin embargo, debe tener cuidado de las siguientes limitaciones posibles:

- Pudiera verse forzado a limitar los nombres de datos a cierta longitud (por ejemplo, 15 o 32 caracteres). Esto probablemente no sea un gran problema, pero podría ser que el usuario insista en un nombre como **destino-del-envío-del-cliente** y que su paquete de elaboración de diccionarios lo obliga a abreviar esto a: **dest-env-clien.**

- Podría haber otras limitaciones artificiales para el nombre. Por ejemplo, el carácter "-" pudiera no permitirse, y podría verse forzado a utilizar en su lugar el carácter "_". O podría verse obligado a utilizar prefijos (o sufijos) en todos los nombres, para indicar el nombre del proyecto de desarrollo del sistema, lo cual lleva a nombres como:

  ctas.pag.GHZ345P14. número_teléfono_vendedor

- Podría verse obligado a asignarle atributos *físicos* (por ejemplo, número de bytes, o bloques de almacenamiento en disco, o representaciones como decimales redondeados) a un dato, aun cuando no sea cuestión del usuario. El diccionario de datos que se discute en este capítulo debe ser un diccionario de análisis y no debiera requerir decisiones de implantación innecesarias o irrelevantes.

Algunos analistas también están empezando a utilizar paquetes automatizados que incluyen gráficos para DFD y otros, *además* de capacidad para elaborar diccionarios de datos. Nuevamente, si tal ayuda existe, debe aprovecharla. Estos paquetes se discuten con mayor detalle en el apéndice A.

Si no dispone de ayudas automáticas para construir el diccionario de datos, debiera por lo menos hacer uso de un procesador de palabras convencional para

crear un archivo de texto de definiciones del diccionario de datos. O, si tiene acceso a una computadora personal, pudiera usar cualquiera de los programas comunes de administración de archivos o de bases de datos (por ejemplo, dBASE, Rbase-5000, PFS-File, Microsoft File en la Apple Macintosh) para construir y administrar el diccionario de datos.

Sólo en los casos más extremos debiera recurrir a un diccionario manual, es decir, tarjetas individuales de 3 x 5 para cada entrada del diccionario. Esto a menudo era necesario en los años 70 e incluso en los 80; a pesar de la popularidad de las computadoras personales y los procesadores de palabras, es decepcionante ver cuántas organizaciones han mantenido a sus programadores y analistas en la época del oscurantismo. Los hijos del zapatero, como dice el refrán, son los últimos en recibir zapatos. Pero hoy en día esto es imperdonable. Si está trabajando en un proyecto donde no tiene acceso a un paquete para elaboración de diccionarios de datos o a un paquete automatizado de herramientas para analista, o a una computadora personal o un sistema procesador de palabras, entonces debería 1) renunciar y encontrar un mejor empleo, o 2) conseguir su propia computadora personal, o 3) hacer ambas cosas.

10.5 RESUMEN

Construir un diccionario de datos es una de las labores más tediosas, y largas, del análisis de sistemas. Pero también es una de las más importantes: sin un diccionario formal que defina el significado de los términos, no se puede esperar precisión.

En el siguiente capítulo veremos cómo se usa el diccionario de datos y el DFD para construir *especificaciones de proceso* para cada uno de los procesos de más bajo nivel.

REFERENCIAS

1. J.D. Lomax, *Data Dictionary Systems.* Rochelle Park, N.J.:NCC Publications, 1977.

2. Tom DeMarco, *Structured Analysis and Systems Specification.* Nueva York: YOURDON Press, 1979.

3. D. Kroenke, *Database Processing.* Chicago: Science Research Associates, 1977.

4. Shaku Atre, *Data Base: Structured Techniques for Design, Performance, and Management.* Nueva York, Wiley, 1980.

PREGUNTAS Y EJERCICIOS

1. Dé una definición de diccionario de datos.

2. ¿Por qué es importante un diccionario de datos para el análisis de sistemas?

3. ¿Qué información da un diccionario de datos acerca de un dato?
4. ¿Qué significa la notación "=" en un diccionario de datos?
5. ¿Qué significa la notación "+" en un diccionario de datos?
6. ¿Qué significa la notación "()" en un diccionario de datos?
7. ¿Qué significa la notación "{}" en un diccionario de datos?
8. ¿Qué significa la notación "[ll]" en un diccionario de datos?
9. ¿Cree Ud. que los usuarios con los que trabaja pueden entender la notación de diccionario estándar que se da en este capítulo? Si no es así, ¿puede sugerir alguna alternativa?
10. Dé un ejemplo de dato elemental.
11. Dé tres ejemplos de datos opcionales.
12. Cuáles son los posibles significados de lo siguiente:

    (a) **domicilio = (ciudad) + (estado)**

    (b) **domicilio = calle + ciudad + (estado) + (código postal)**
13. Dé un ejemplo del uso de la notación de iteración.
14. ¿Cuál es el significado de cada una de las siguientes notaciones?

    (a) a = 1{b}

    (b) a = {b}10

    (c) a = 1{b}10

    (d) a = 10{b}10
15. ¿Tiene sentido definir un **pedido** de la siguiente manera?

    pedido = nombre-de-cliente + domicilio-de-envío + 6{artículo}

    ¿Por qué? o ¿por qué no?
16. Dé un ejemplo de la construcción de selección.
17. ¿Qué significado tiene un alias en el diccionario de datos?
18. ¿Por qué debieran usarse lo menos posible los alias?

19. ¿Qué tipo de anotación debe hacerse en el DFD para indicar que un dato es un alias?
20. ¿Cuáles son los tres asuntos de importancia que surgen cuando el usuario ve el diccionario de datos?
21. ¿Cree Ud. que los usuarios en su organización podrán entender la notación del diccionario de datos?
22. ¿Cree Ud. que la notación que se muestra en este capítulo sea más compleja, o menos, que la musical?
23. ¿Cuáles son las tres actividades de verificación de posibles errores que puede llevar a cabo el analista en el diccionario de datos *sin* ayuda del usuario?
24. ¿Cuáles son las limitaciones probables de un paquete automatizado de generación de diccionarios de datos?
25. Dé una definición de diccionario de datos de **nombre-del-cliente** basada en la siguiente especificación verbal del usuario: "Cuando registramos el nombre de un cliente, tenemos cuidado de incluir un título de cortesía, que puede ser 'Sr.', 'Srita.', 'Sra.' o 'Dr.'. (Existen otros muchos títulos como 'Profesor', 'Sir', etc., pero no nos ocupamos de ellos.) Cada uno de nuestros clientes tiene un nombre de pila, pero permitimos sólo una inicial si ellos lo prefieren. Los segundos nombres son opcionales. Y, desde luego, se requiere el apellido; permitimos una buena gama de apellidos, incluyendo los que conllevan guión ('Smith-Frisby', por ejemplo) y apóstrofo ("D'Arcy"), etc. Incluso permitimos un sufijo optativo, para dar cabida a cosas como 'Tom Smith, Jr.'o 'Harvey Shmrdlu 3º'.
26. ¿Qué está mal en las siguientes definiciones de diccionario de datos?

    (a) a = b c d

    (b) a = b + + c

    (c) a = {b

    (d) a = 4{b}3

    (e) a = {x)

    (f) x = ((y))

    (g) p = 4{6{y}8}6
27. En el ejemplo del hospital de la sección 9.2, ¿qué implican las definiciones de **peso** y **estatura**? Comentario: Implicaría que sólo estamos midiendo en unidades enteras y no estamos considerando las fracciones adicionales, etc.

28. Escriba una definición de diccionario de datos de la información que contiene su licencia de manejo. Si no tiene, encuentre algún amigo que tenga.

29. Dé una definición de diccionario de datos de la información que contiene una tarjeta común de crédito bancario típica (por ejemplo, Visa o MasterCard).

30. Dé una definición de diccionario de datos de la información que contiene un pasaporte.

31. Dé una definición de diccionario de datos de la información que contiene un billete de lotería.

11 ESPECIFICACIONES DE PROCESO

Nuestros pequeños sistemas tienen su día.
Alfred, Lord Tennyson
In Memoriam, 1850

En este capítulo se aprenderá:

1. Cómo escribir especificaciones estructuradas de procesos.
2. Cómo escribir especificaciones de proceso con pre/post condiciones.
3. Cómo utilizar tablas de decisiones para escribir especificaciones de proceso.
4. Cuándo utilizar herramientas alternativas de especificación.

En este capítulo se explora la *especificación del proceso*, la descripción de qué es lo que sucede en cada burbuja primitiva de nivel más bajo en un DFD. Varios textos, incluyendo [DeMarco, 1978], [Gane y Sarson, 1977] y [Weinberg, 1978] también utilizan el término *minispec* (como abreviatura de especificación en miniatura) como alternativa de especificación de proceso. Sin importar el nombre, el propósito de una especificación de proceso es bastante claro: define lo que debe hacerse para transformar entradas en salidas. Es una descripción detallada de la política de negocios del usuario que cada burbuja lleva a cabo.

Como veremos en este capítulo, existe una variedad de herramientas que podemos utilizar para producir una especificación de proceso: tablas de decisiones, lenguaje estructurado (español, inglés, etc.), pre/post condiciones, diagramas de flujo, diagramas de Nassi/Shneiderman, etc. A pesar de que la mayoría de los analistas están a favor del lenguaje estructurado, debe recordar que se puede usar cualquier método mientras satisfaga dos requerimientos cruciales:

- *La especificación del proceso debe expresarse de una manera que puedan verificar tanto el usuario como el analista.* Precisamente por esta razón se evita el lenguaje narrativo como herramienta de especificación: es notoriamente ambiguo, sobre todo si describe acciones alternativas (decisiones) y acciones repetitivas (ciclos). Por naturaleza, también tiende a causar gran confusión cuando expresa condiciones booleanas compuestas (esto es, combinaciones de los operadores booleanos AND, OR y NOT) (y, o, no, respectivamente).

- *El proceso debe especificarse en una forma que pueda ser comunicada efectivamente al público amplio que esté involucrado.* A pesar de que el analista es típicamente quien *escribe* la especificación del proceso, habitualmente será un público bastante diverso de usuarios, administradores, auditores, personal de control de calidad y otros, el que leerá la especificación del proceso. Una especificación pudiera expresarse tal vez con cálculo de predicados, o en Pascal, o en un enfoque de diagramación formal como USE-IT de Higher Order Software;[1] pero de nada sirven esas especificaciones si la comunidad usuaria se rehúsa a verlas. Lo mismo pudiera suceder con las tablas de decisiones, con el lenguaje estructurado o con otras herramientas; en gran medida esto es función de la personalidad, antecedentes y humor de los usuarios con los que trate.

Como se mencionó anteriormente, la mayoría de los analistas usan el lenguaje estructurado como método favorito para escribir especificaciones de proceso. Tal vez sea más importante señalar que la mayoría de los analistas y de las organizaciones utilizan *una* herramienta para escribir *todas* sus especificaciones.[2] Esto es, en mi opinión, un gran error: usted debe sentirse libre para utilizar una *combinación* de herramientas de especificación, según a) las preferencias del usuario, b) sus propias preferencias y, c) la naturaleza propia de los diversos procesos.

---

1 Para más información acerca de USE-IT, vea *Structured Techniques for Computing*, de James Martin y Carma McClure. (Englewood Cliffs, N.J.: Prentice-Hall, 1986).

2 Esto a menudo lo causa la introducción de un conjunto completo de *estándares* para el análisis estructurado en la organización. A pesar de que los estándares son un buen esfuerzo para combatir la desidia, ignorancia y anarquía total, a menudo van demasiado lejos y prescriben una solución rígida para todo problema. Como lo indica el dicho: "Si su única herramienta es un martillo, todo el mundo parece un clavo".

Una buena herramienta de especificación de proceso debe también tener una tercera característica: no debe imponer (o implicar) decisiones de diseño e implantación arbitrarias. A menudo esto es muy difícil, pues el usuario, de quien depende la "política" que realizará cada burbuja en el DFD, suele escribirla *en los términos en los que la lleva a cabo en la actualidad.* Su trabajo como analista consiste en destilar de esto la esencia de *lo que* dicha política es y no cómo se lleva a cabo hoy en día.

Considere el siguiente ejemplo: el analista discute un pequeño fragmento del sistema, como lo ilustra la figura 11.1. Quiere desarrollar una especificación de proceso para la burbuja etiquetada CALCULAR FACTOR-W. Dado que el analista no está familiarizado con la aplicación, entrevistó al usuario para aprender que la política a seguir para calcular los factores-W para cualquier valor de entrada x es la siguiente:

1. El factor-W no se produce como resultado de un solo cálculo. De hecho, tenemos que empezar por hacer una adivinanza. El usuario dice que en lo particular le gusta usar el 14 como primer intento de adivinar.

2. Luego volvemos a adivinar. Esto se hace dividiendo el número que acabamos de adivinar entre el número **x** con el que comenzamos.

3. Luego tomamos el resultado de dicho cálculo y se lo restamos al número que acabamos de adivinar.

4. Tomamos el resultado del paso 3 y lo dividimos entre dos. Esto se convierte en nuestro nuevo número adivinado.

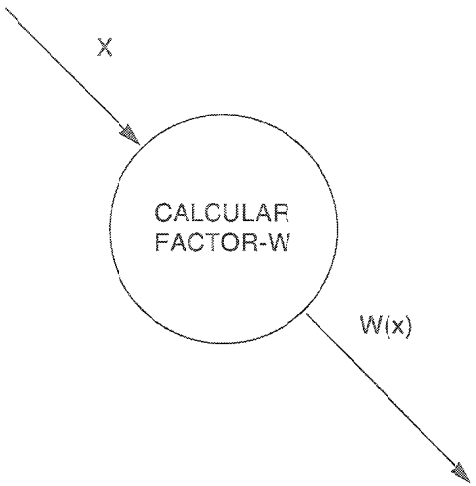


Figura 11.1: **Cálculo del factor-W**

5. Si el nuevo número adivinado y el anterior son muy cercanos, digamos con diferencia menor a 0.0001, entonces nos podemos detener; el nuevo número adivinado es el **factor-W**. De otro modo, regrese al paso 2 y vuelva a repetir todo.

Podría argumentarse que esta especificación de proceso es difícil de leer y entender pues está escrita en lenguaje narrativo. De hecho, la descripción siguiente es más compacta (note que las barras verticales "|" en el **HASTA** significan el "valor absoluto de" la expresión que encierran.

factor-W_0 = 14

REPITE para N = 0 en pasos de 1

factor-W_{N+1} = (factor-W_N - (X/factor-W_N)) / 2

HASTA |factor-W_{N+1} - factor-W_N| < 0.0001

Sin embargo, incluso esto tiene fallas: describe una política en términos de una implantación de procedimiento particular. La política, como pudiera ser evidente (pero que igualmente pudiera *no* serlo), es el algoritmo de Newton-Raphson de aproximación a la raíz cuadrada.[3] La siguiente especificación de proceso describe la misma política, pero da al diseñador/programador completa libertad para escoger su propio algoritmo:

PRECONDICION
Existe un número **X** no negativo

POSTCONDICION
Se produce un **factor-W** tal que
X = factor-W * factor-W

El programador puede en efecto optar por usar el algoritmo del usuario para calcular la raíz cuadrada, pero no debería sentirse restringido por el analista a hacerlo. De hecho, la atención extravagante al algoritmo del procedimiento, sobre todo en la primera versión de la especificación anterior, impedía por completo ver lo que el proceso era en realidad.

Antes de explorar las diversas herramientas de especificación de proceso, debería hacerse énfasis en un punto: las especificaciones de proceso *sólo* se desarrollan para los procesos de *más bajo nivel* en un conjunto de diagramas por niveles en un DFD. Como se ve en la figura 11.2, los procesos de mayor nivel se definen por medio de la red de procesos del nivel inmediato inferior. En otras palabras, la especificación de proceso para una burbuja de nivel superior es el DFD de nivel inferior.

3 Intente el algoritmo en un par de casos de prueba. Encontrará que converge bastante rápidamente.

Escribir una especificación de proceso adicional en lenguaje estructurado sería superfluo y redundante; esto es, crearía una especificación más difícil de actualizar.[4]

En este capítulo nos concentraremos en tres herramientas principales de especificación de proceso:

- Lenguaje estructurado (español, inglés, etc.)
- Pre/post condiciones
- Tablas de decisión

También comentaremos brevemente sobre varias herramientas menos utilizadas: lenguaje narrativo, diagramas de flujo y diagramas de Nassi-Shneiderman.

11.1 LENGUAJE ESTRUCTURADO

El *lenguaje estructurado*, como el nombre indica, es "lenguaje español (o inglés u otro) con estructura". Es decir, es un subconjunto de todo el idioma con importantes restricciones sobre el tipo de frases que pueden utilizarse y la manera en que pueden juntarse dichas frases. También se conoce con nombres como PDL (siglas en inglés de lenguaje de diseño de programas) y PSL (lenguaje de planteamiento o especificación de problemas). Su propósito es hacer un balance razonable entre la precisión del lenguaje formal de programación y la informalidad y legibilidad del lenguaje cotidiano.

Una *frase* en lenguaje estructurado puede consistir en una ecuación algebraica, por ejemplo,

X = (Y*Z)/(Q+14)

o en una sencilla frase imperativa que consista en un verbo y un objeto. Nótese que esta frase no tiene el punto y coma que termina una instrucción en muchos lenguajes de programación; puede o no terminar con un punto ("."), dependiendo de sus gustos en esta materia. Además, note que las frases que describen los cálculos pueden usarse con prefijos de los verbos CALCULAR, AÑADIR, FIJAR, etc., por lo que se pudo haber escrito el ejemplo anterior así:

CALCULAR X = (Y*Z)/(Q+14)

y también se tienen cálculos expresados en lenguaje, como los siguientes:

4 Sin importar esta advertencia, debemos señalar que, como analista, a veces se le pedirá que produzca una especificación de proceso escrita de los procesos de mayor nivel. Esto sucederá si el usuario decide que quiere mostrar la especificación a su jefe, y le preocupa que el jefe no tolere la idea de DFD por niveles. Así que el usuario le dirá, "Mire, sé que no necesita una especificación de proceso para estas burbujas de alto nivel, pero apreciaría que las escribiera de todos modos para que el jefe pueda entender de qué se trata el sistema". Tendrá que lidiar con este problema con la misma diplomacia que utiliza para resolver todos los demás problemas políticos de su proyecto.

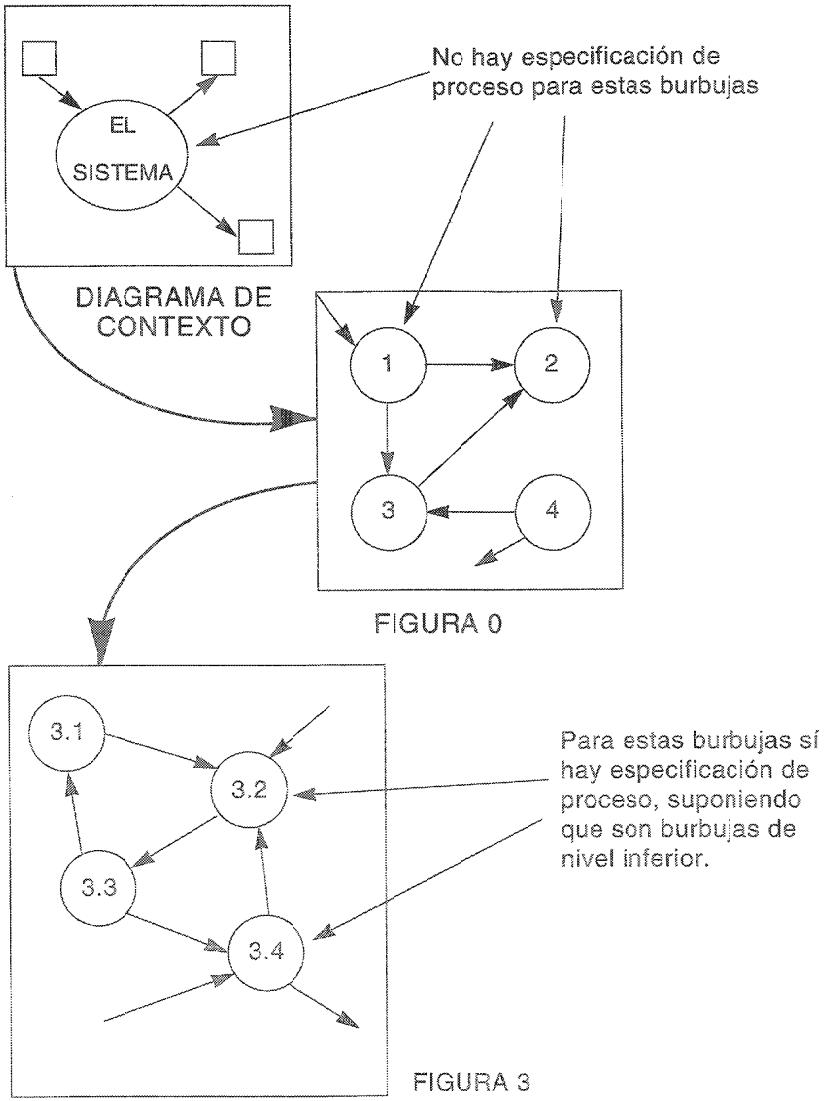


Figura 11.2: **Especificaciones de proceso para burbujas de bajo nivel**

FIJAR IMPUESTO A 13
SUMAR 3 A X
MULTIPLICAR PRECIO UNITARIO POR CANTIDAD
DIVIDIR GANANCIAS ACTUALES ENTRE PERDIDAS ACTUALES

Los verbos deben escogerse de entre un pequeño grupo de verbos orientados a la acción tales como:

CONSEGUIR (o ACEPTAR o LEER)
PONER (o MOSTRAR o ESCRIBIR)
ENCONTRAR (o BUSCAR o LOCALIZAR)
SUMAR
RESTAR
MULTIPLICAR
DIVIDIR
CALCULAR
BORRAR
ENCONTRAR
VALIDAR
MOVER
REEMPLAZAR
FIJAR
ORDENAR

En muchas organizaciones se llega a la conclusión de que entre 40 y 50 verbos son suficientes para describir cualquier política dentro de una especificación de proceso.

Los objetos (el tema de las frases imperativas sencillas) deben consistir sólo en datos que se han definido en el diccionario o ser términos locales. Los términos locales son aquellos que se definen explícitamente en una especificación de proceso individual; sólo son conocidos, relevantes y con significado dentro de dicha especificación de proceso. Un ejemplo típico de término local es un cálculo intermedio, que se utiliza para producir una salida final del proceso.[5] Por ejemplo, la especificación de proceso en lenguaje estructurado que se muestra a continuación examina una serie de registros de pedidos en el almacén PEDIDOS, para calcular un total diario:

total-diario = 0
HACER MIENTRAS haya más **pedidos** en **PEDIDOS** con **fecha-de-pedido** = fecha de hoy
    LEER el siguiente **PEDIDO** en **PEDIDOS** con **fecha-de-pedido** = fecha de hoy
    MOSTRAR a Contabilidad **número-de-pedido, nombre-del-cliente** y **cantidad-total**. total-diario = total-diario + **cantidad-total**

5 Los términos locales se definen dentro de la especificación de proceso donde ocurren, y no se definen en el diccionario de datos. A menudo se derivan (o calculan directamente) de términos que ya están en el diccionario de datos, de modo que sería redundante añadir dichos términos locales. Además, por definición, los términos locales sólo se conocen en un contexto local (es decir, dentro de una burbuja en un DFD). No deben aparecer como flujo en el DFD, y usualmente no forman parte del vocabulario normal de las palabras orientadas a la aplicación del usuario.

FIN HACER

MOSTRAR a Contabilidad total-diario

Finalmente, el lenguaje estructurado permite que se combinen frases en unas cuantas formas limitadas que se toman de las construcciones acostumbradas de la programación estructurada.[6]

- La construcción **SI-ENTONCES-OTRO** se utiliza para describir frases alternativas que se deben realizar según el resultado de la decisión binaria. La construcción **SI-ENTONCES-OTRO** puede tomar cualquiera de las formas siguientes:

  SI condición-1
  frase-1
  FIN SI

  o bien

  SI condición-1
  frase-1
  OTRO
  frase-2
  FIN SI

  De esta forma, el analista puede escribir:

  SI el **cliente** vive en Nueva York
  añadir **cliente** a **PROSPECTOS-DE-MERCADO**
  FIN SI

  o bien

  SI edad-del-cliente es mayor que 65
  fijar cuota a **cuota-ancianos**
  OTRO
  fijar cuota a **cuota-normal**
  FIN SI

- La construcción **CASO** se utiliza para describir frases alternativas que se efectuarán basándose en los resultados de una decisión multivaluada (en contraste con la decisión *binaria* que tiene lugar con la construcción **SI-ENTONCES**). La construcción CASO toma la forma general:

6 Si no está familiarizado con la programación estructurada, consulte cualquiera de los textos ordinarios sobre el tema, o vea algunos de los primeros artículos sobre el tema que se recopilaron en [Yourdon, 1979].

HACER CASO
CASO variable = valor-1
frase-1
.
.
.
CASO variable = valor-n
frase-n
OTRO
frase-n+1
FIN CASO

Por lo que el analista puede escribir:

HACER CASO
CASO edad-del-cliente < 13
fijar **cuota** a **cuota-niños**
CASO edad-del-cliente > 12 y **edad-del-cliente** < 20
fijar **cuota** a **cuota-adolescente**
CASO edad-del-cliente > 19 y **edad-del-cliente** < 65
fijar **cuota** a **cuota-adultos**
OTRO
fijar **cuota** a **cuota-ancianos**
FIN CASO

O, como otro ejemplo, considere la siguiente porción de una especificación de proceso en lenguaje estructurado:

HACER CASO
CASO estado = "NY"
fijar **impuesto-de-venta** a 0.0825
CASO estado = "NJ"
fijar **impuesto-de-venta** a 0.07
CASO estado = "CA"
fijar **impuesto-de-venta** a 0.05
OTRO
fijar impuesto-de-venta a 0
FIN CASO

Nótese que la cláusula **OTRO** suele usarse para abarcar situaciones que el usuario se olvida de especificar y por las que el analista se olvida de preguntar; a menudo llevará a discusiones entre el usuario y el analista que de otra manera no sucederían sino hasta después de puesto en operación el sistema. Considere el siguiente ejemplo:

```
HACER CASO
CASO forma-de-pago = "efectivo"
  fijar descuento a 0.05
CASO forma-de-pago = "tarjeta-de-crédito"
  fijar descuento a 0.01
OTRO
  fijar descuento a 0
FIN CASO
```

El usuario podría cuestionar esta especificación del proceso y preguntar por qué el analista incluyó el **OTRO**; el analista pudiera responder preguntando acerca de pagos con cheque, cheques de viajero, monedas de oro e intercambios.

La construcción **HACER-MIENTRAS** se usa para describir una frase que deberá llevarse a cabo repetitivamente hasta que alguna condición booleana se haga verdadera. Toma la forma general:

```
HACER-MIENTRAS condición-1
  frase-1
FIN HACER
```

La prueba (en el ejemplo anterior, la "condición-1") se hace *antes* de que se ejecute la frase-1, por lo cual, si la condición no se satisface, es posible que la frase-1 se ejecute *cero* veces.

Por ejemplo, el analista puede escribir:

```
HACER-MIENTRAS haya más artículos en el pedido-del-cliente
  precio-extendido = precio-unitario*cantidad-de-unidades
FIN HACER
```

Muchas organizaciones incluyen otra estructura que ejecuta una frase especificada por lo menos una vez antes de hacer una prueba para ver si debe repetirse. Esta variante, usualmente conocida como la construcción **REPITE-HASTA**, tiene la siguiente forma:

```
REPITE
frase-1
HASTA condición-1
```

Se pueden construir *frases compuestas* a partir de combinaciones de frases sencillas y las estructuras sencillas que se presentaron anteriormente, de acuerdo con las siguientes reglas:

1. Una secuencia lineal de frases sencillas equivale (estructuralmente) a una frase sencilla. Así que la secuencia


```
frase-1
frase-2
.
.
.
frase-n
```

es estructuralmente equivalente a una frase sencilla única *y puede ser sustituida dondequiera que se espere una frase sencilla.* Esto permite construir estructuras como la siguiente:

```
SI condición-1
        frase-1
        frase-2
OTRO
        frase-3
        frase-4
        frase-5
FIN SI
```

o bien

```
HACER MIENTRAS condición-1
        frase-1
        frase-2
        frase-3
FIN HACER
```

2. Una construcción **SI-ENTONCES-OTRO** sencilla se considera estructuralmente equivalente a una frase única sencilla. Esto permite que las estructuras **SI-ENTONCES-OTRO** se aniden dentro de otras estructuras iguales, o dentro de estructuras **HACER-MIENTRAS**, o dentro de estructuras **CASO**. Por ejemplo:

```
SI condición-1
        frase-1

        SI condición-2
                frase-2
                frase-3
        OTRO
                frase-4
                frase-5
        FIN SI
        frase-6
```

```
OTRO
        frase-7
        SI condición-3
                frase-8
        FIN SI
        frase-9
FIN SI
```

3. Una estructura **HACER-MIENTRAS** sencilla se considera estructuralmente equivalente a una frase única sencilla. Esto permite que las estructuras **HACER-MIENTRAS** se aniden dentro de otras estructuras iguales, o dentro de estructuras **SI-ENTONCES-OTRO,** o dentro de estructuras **CASO.** Así, se puede tener una especificación en lenguaje estructurado de la siguiente naturaleza:

```
gran-total = 0
HACER-MIENTRAS haya más pedidos que procesar
        total-de-pedidos = 0
        LEER el siguiente pedido de PEDIDOS
        HACER-MIENTRAS haya más artículos en el pedido
                total-de-pedidos = total-de-pedidos + número-
                de-artículos
        FIN HACER
        MOSTRAR número-de-pedido, total-de-pedidos
        gran-total = gran-total + total-de-pedidos
FIN HACER
MOSTRAR gran-total
```

4. Una estructura sencilla **CASO** se considera estructuralmente equivalente a una frase única sencilla. Esto permite que las estructuras **CASO** se aniden dentro de otras iguales, dentro de estructuras **SI-ENTONCES-OTRO** o dentro de estructuras **HACER-MIENTRAS.**

Como puede verse, esto permite construir arbitrariamente descripciones complejas de políticas de negocios, que a la vez mantienen un control estricto sobre el vocabulario, organización y estructura de la descripción. Sin embargo, esta complejidad es también la principal desventaja del lenguaje estructurado: si el analista compone una especificación de proceso demasiado compleja para ser entendida y verificada por el usuario, entonces falló. Esto usualmente se puede evitar mediante las siguientes tres reglas:

1. Restrinja la especificación de proceso en lenguaje estructurado a una sola página de texto (por ejemplo, una hoja de 8 x 11, que son 66 líneas de texto en un sistema procesador de palabras). Si la especificación ocupa más de una página, entonces el analista (con la ayuda del usuario) debe pensar en una forma totalmente distinta de formular la política (por ejemplo, escoger un algoritmo diferente, más sencillo). Si no se puede, entonces es posible que el proceso mismo (esto es, la burbuja dentro del DFD) sea demasiado complejo, y debe partirse en una red de procesos más simples de nivel inferior.

2. No permita más de tres niveles de anidamiento (es decir, tres niveles de estructuras anidadas **SI-ENTONCES-OTRO** o tres niveles de estructuras **CASO,** etc.). En particular, en el caso de estructuras **SI-ENTONCES-OTRO,** incluso más de dos niveles de anidamiento es un indicio de que sería preferible una especificación mediante una *tabla de decisiones*; esto se discute en la sección 11.3.

3. Evite confusiones acerca de los niveles de anidamiento utilizando sangrías, como se muestra en los ejemplos anteriores. Esto se puede lograr y controlar muy fácilmente si está utilizando algún tipo de auxilio automatizado para desarrollar las especificaciones de proceso (incluso algo tan sencillo como un sistema estándar de procesamiento de textos). Si las especificaciones de proceso las está tecleando manualmente alguna persona no familiarizada con la programación o el análisis estructurados, tendrá que explicarle con mucho cuidado qué tipo de sangrías se desea; también debiera revisar con cuidado el texto resultante para ver que esté correcto.

Muchos analistas preguntan si se puede esperar que el usuario lea y entienda una especificación de proceso escrita en lenguaje estructurado. En esta área, mi experiencia ha resultado casi uniformemente positiva: los usuarios sí pueden leer el lenguaje estructurado, con las siguientes advertencias:

1. Tendrá que repasar una o dos veces el documento para asegurarse de que entiendan el formato y las diversas construcciones. En la primera lectura, bien pudiera parecer un documento legal, sobre todo si ha enfatizado la construcción **SI-ENTONCES-OTRO** y las demás.

2. No se refiera a la especificación del proceso como "lenguaje estructurado". De ser necesario, refiérase a ella como "una descripción formal de la política de negocios para realizar esta actividad".

3. Ponga mucha atención al formato global y la distribución del documento; la sangría de los bloques anidados de lógica es de especial importancia. Algunos usuarios prefieren un estilo de sangría de "silueta", es decir, donde los niveles de sangría se numeran 1.1, 1.1.1, 1.1.1.1, etc.

En el caso de estudio del Apéndice F se muestran diversos ejemplos de especificaciones de proceso en lenguaje estructurado.

11.2 PRE/POST CONDICIONES

Las pre/post condiciones son una manera conveniente de describir la *función* que debe realizar el proceso, sin decir mucho acerca del *algoritmo* o *procedimiento* que se utilizará. Resulta ser un enfoque particularmente útil cuando:

1) El usuario tiene tendencia a expresar la política llevada a cabo por la burbuja en términos de un algoritmo particular que ha estado utilizando durante décadas.

2) El analista está razonablemente seguro de que existen *muchos* algoritmos distintos que podrían usarse.

3) El analista desea que el programador explore varios de estos algoritmos, pero no quiere involucrarse personalmente con tales detalles y, *sobre todo*, no quiere enredarse en discusiones con el usuario acerca del mérito relativo de cada uno.

Un ejemplo de una especificación de proceso escrita con el enfoque de la pre/post condición se muestra en la figura 11.3:

ESPECIFICACION DE PROCESO 3.5: CALCULAR EL IMPUESTO SOBRE VENTAS

Precondición 1
Ocurre **DATOS-VENTA** con **TIPO-ITEM** que corresponde con **CATEGORIA-ITEM** en **CATEGORIAS-IMPUESTO**
Postcondición 1
IMPUESTO-SOBRE-VENTA se hace igual a **MONTO-VENTA** * **IMPUESTO**

Precondición 2
Ocurre **DATOS-VENTA** con **TIPO-ITEM** que no concuerda con **CATEGORIA-ITEM** en **CATEGORIAS-IMPUESTO**
Postcondición 2
Se genera **MENSAJE-ERROR**

Figura 11.3: **Especificación de una pre/post condición**

Como puede verse, existen dos partes principales del proceso: *precondiciones* y *postcondiciones*. Además, tales especificaciones pueden contener *términos locales*, como se define en la sección 11.1 (ver también la Nota 5).

Las *precondiciones* describen todas las cosas (si hay) que deben darse antes de que el proceso pueda comenzar a ejecutarse. A veces es conveniente pensar que el proceso es "la bella durmiente", y que las precondiciones representan el "beso mágico" que despertará al proceso y lo echará a andar. De manera alternativa, se

puede imaginar a las precondiciones como una garantía del usuario: "Garantizo que cuando se active este proceso se cumplirán las siguientes cosas". Típicamente, las precondiciones describirán lo siguiente:

- *Qué entradas se encuentran disponibles.* Estas entradas llegan mediante un flujo conectado con un proceso, como se muestra en el DFD. Nótese que puede haber casos en los que diversos flujos entran a un proceso, pero sólo uno de ellos es precondición necesaria para que se active el proceso. Por ejemplo, si hubiera una especificación que empieza con:

  Precondición
  ocurre el dato **X**

  asociada con el DFD que se muestra en la figura 11.4, se interpretaría de la siguiente forma: la llegada del dato **X** es el estímulo activador que hace que el proceso empiece a trabajar. Como parte de su trabajo, busca en-

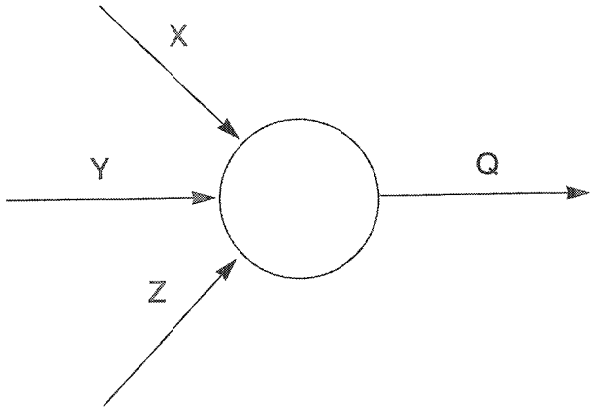


Figura 11.4: **DFD con entradas X, Y y Z**

  tradas de los flujos **Y** o **Z**, o ambos, *pero **Y** y **Z** no son necesarios para que el proceso comience su trabajo.*

- *Qué relación debe existir entre las entradas.* Muy a menudo, una precondición especificará que deben llegar dos entradas con campos que corresponden (por ejemplo, detalles de pedidos y detalles de envío con el mismo número de cuenta). O bien, la precondición puede especificar que un componente de un dato de entrada debe estar dentro de cierto intervalo (por ejemplo, "pedido con fecha de entrega a más de 60 días").

- *Qué relaciones deben existir entre entradas y almacenes de datos.* Una precondición pudiera estipular que exista un registro dentro de un almacén que corresponda con algún aspecto de un dato de entrada (por ejem-

plo, la precondición puede establecer que "hay un pedido-de-cliente con número-de-cuenta-de-cliente que corresponde con un número-de-cuenta-de-cliente del almacén de clientes").

- *Qué relaciones deben existir entre diferentes almacenes o dentro de un almacén dado.* Es decir, la precondición podría establecer que "hay un pedido en el almacén de pedidos cuyo número-de-cuenta-de-cliente corresponde con el número-de-cuenta-del-cliente en el almacén de clientes". O bien, la precondición pudiera establecer que "existe un pedido en el almacén de pedidos con fecha-de-envío igual a la fecha actual".

De manera similar, las postcondiciones describen lo que debe darse cuando el proceso ha concluido. Nuevamente, esto puede imaginarse como una garantía: "Garantizo que cuando el proceso haya concluido se debe cumplir lo siguiente". Las postcondiciones típicamente describen lo siguiente:

- *Las salidas que generará o producirá el proceso.* Esta es la forma más común de postcondición (por ejemplo, "se producirá una factura").
- *Las relaciones que existirán entre los valores de salida y los valores originales de entrada.* Esto es común para la situación donde una salida es una función matemática directa de un valor de entrada. De esta forma, una postcondición pudiera afirmar que "la **factura-total** se calcula como la suma de **precios-unitarios-de-artículos** más **costos-de-envío**".
- *Las relaciones que existirán entre valores de salida y los valores en uno o varios de los almacenes.* Esto es común cuando la información debe recuperarse de un almacén y utilizarse como parte de la salida de un proceso. Por ejemplo, una especificación de proceso pudiera tener como postcondición la siguiente afirmación: "el **balance-actual** en el almacén **INVENTARIO** se incrementará con **cantidad-recibida**, y el nuevo **balance-actual** se producirá como salida de este proceso."
- *Los cambios que se hayan dado en los almacenes:* nuevos artículos añadidos, artículos existentes que se hayan modificado, o artículos existentes que se hayan eliminado. Así, pudieran verse afirmaciones tales como "el **pedido** se anexará al almacén de **PEDIDOS**", o "el registro de **clientes** se eliminará del almacén de **CLIENTES**".

Cuando se esté construyendo una especificación de pre/post condiciones se debe comenzar por describir las situaciones normales de proceso. Pudieran existir diversas situaciones normales diferentes (por ejemplo, combinaciones únicas de relaciones de entrada/almacenaje válidas), cada una de las cuales se expresa como precondición distinguible e individual. Para cada una de estas precondiciones se debe describir la condición de la burbuja del proceso cuando se han producido las salidas y se han modificado los almacenes. Después de haber descrito las situaciones

normales de proceso, deben incluirse precondiciones y postcondiciones apropiadas para los casos de error y casos anormales. Considere la especificación de pre/post condiciones que se muestra en la figura 11.5(b), que se desarrollaría para un nuevo sistema de la especificación narrativa de la figura 11.5(a).

> Si un cliente dice que está clasificado como cliente que puede llevar mercancía "a su cuenta" cuando llega a la caja a pagar, entonces busco su cuenta en mi archivo. Si la encuentro, y no está señalada como "suspendida" o "cancelada", entonces cargo la mercancía a su cuenta con el número de ésta y el monto de la venta. De otra manera, le digo que tendrá que pagar en efectivo o hablar con el gerente.

Figura 11.5(a): **Un ejemplo de especificación narrativa**

Precondición 1
El comprador llega con un número-de-cuenta que corresponde con un número de cuenta en CUENTAS, cuyo código-de-status se pone en "válido".
Postcondición 1
Se produce una factura con número-de-cuenta y monto-de- venta.

Precondición 2
La precondición 1 falla por algún motivo (el número-de- cuenta no se encuentra en CUENTAS, o el código-de-status no es "válido").

Postcondición 2
Se produce un mensaje de error.

Figura 11.5(b): **Ejemplo de pre/post condiciones**

Aunque el enfoque de pre/post condiciones sea bastante útil y tenga un gran número de ventajas, hay ocasiones en las cuales puede no ser apropiado. La falta de pasos intermedios entre entradas (precondiciones) y salidas (postcondiciones) es deliberada y consciente, pero puede volverse difícil de entender si el lector no visualiza algún tipo de procedimiento que lleve de las entradas a las salidas. Además, si existen relaciones complejas entre entradas y salidas, podría ser más fácil escribir una especificación utilizando lenguaje estructurado. Un ejemplo de especificación de precondición/postcondición que probablemente sea demasiado complicado se muestra en la figura 11.6

DETERMINAR TASA DE PRESTAMO SEGUN FACTORES DE COMPRADORES

Precondición 1

Ocurre **una solicitud-de-préstamo**

y **antigüedad** > 5 o **valor-neto** > **monto-del-préstamo**
y **gastos-mensuales** < 0.25 * **monto-del-préstamo** o **garantía-colateral** > 2 * **monto-del-préstamo** y **edad** > 25
o **garantía-colateral** > **monto-del-préstamo** y **edad** > 30
o **antigüedad** > 2 y **valor-neto** > 2 * **monto-del-préstamo** y **edad** > 21 y **gastos-mensuales** < 0.5 * **monto-del-préstamo**

Postcondición 1

monto-aprobado = monto-del-préstamo

Figura 11.6: **Especificación de pre/post condición demasiado complicada**

Como con todas las formas de especificación de proceso, permita que su propio juicio y las reacciones del usuario lo guíen; si el usuario encuentra la especificación de precondición/postcondición demasiado difícil de leer, escoja otro formato. En el caso de estudio del apéndice G se muestra el enfoque de precondición/postcondición; el enfoque alternativo de lenguaje estructurado se utiliza en el caso de estudio del apéndice F. Analice cuidadosamente ambos casos de estudio para determinar lo adecuado de estas dos herramientas de especificación de proceso.

11.3 TABLAS DE DECISION

Existen situaciones donde ni el lenguaje estructurado ni las pre/post condiciones son adecuadas para escribir especificaciones de proceso. Esto se da sobre todo si el proceso debe producir alguna salida o tomar alguna acción basada en *decisiones complejas.* Si las decisiones se basan en diversas variables distintas (por ejemplo, datos de entrada), y si dichas variables pueden tomar diversos valores, entonces la lógica expresada por el lenguaje estructurado o por las pre/post condiciones probablemente sea tan compleja que el usuario no la comprenderá. Probablemente sea preferible una tabla de decisiones.

Como se muestra en la figura 11.7, una tabla de decisiones se crea listando todas las variables relevantes (a veces conocidas como *condiciones* o *entradas)* y todas las acciones relevantes en su lado izquierdo; nótese que las variables y acciones están separadas por medio de una línea horizontal gruesa. En este ejemplo, las variables son *lógicas,* lo cual significa que pueden tomar el valor de verdadero o falso

En muchas aplicaciones es fácil (y preferible) expresar las variables como binarias (verdadero-falso), pero también se pueden construir las tablas de decisión a partir de variables multivaluadas; por ejemplo, se puede construir una tabla de decisiones con una variable llamada "edad-del-cliente", cuyos valores relevantes sean "menos de 10", "entre 10 y 30", y "más de 30".

A continuación, se lista en columnas separadas cada combinación posible de valores de las variables; cada columna usualmente se conoce como *regla.* Una regla describe una acción (o acciones) que deben llevarse a cabo para una combinación específica de valores de las variables. Por lo menos debe especificarse una acción para cada regla (esto es, para cada columna de la tabla de decisiones), o el comportamiento del sistema para tal situación no quedará especificado.

	1	2	3	4	5	6	7	8
Edad > 21	Y	Y	Y	Y	N	N	N	N
Sexo	M	M	F	F	M	M	F	F
Peso > 100	Y	N	Y	N	Y	N	Y	N
Medicamento 1	X				X			X
Medicamento 2		X			X			
Medicamento 3			X			X		X
Ningún medicamento				X			X	

Figura 11.7: **Tabla de decisiones típica**

Si existen N variables con valores binarios (verdadero-falso), entonces existirán 2^N reglas distintas; así que si existen tres condiciones, habrá 8 reglas, y si hay 7 condiciones habrá 128 reglas. Enumerar todas las reglas es un proceso sencillo. Al tratar el Sí (o V) como un cero binario, y el No (o F) como un uno binario, es fácil generar una secuencia de 000, 001, 010, 011, 100, 101, etc., hasta que se hayan generado todas las 2^N combinaciones.[7]

Debe discutirse cada regla con el usuario para asegurarse de que se ha identificado la acción o acciones correctas para cada combinación de variables. Es bastante común, al hacer esto, encontrar que el usuario jamás ha pensado en ciertas combinaciones de variables, o que nunca hayan ocurrido en su experiencia.[8] La ventaja del enfoque de la tabla de decisiones es que el analista se puede concentrar en una regla a la vez.

---

7 Desde luego, habrá situaciones en las cuales las condiciones de la tabla de decisiones no sean binarias por naturaleza, sino que puedan tomar diversos valores (por ejemplo, una solicitud de seguro puede involucrar **edad-de-cliente,** y puede usar valores tales como "menor de 18 años", "de 18 a 64", y "de 65 o más"). Para determinar el número total de reglas en una tabla de éstas debemos multiplicar el número de valores que puede tomar la variable 1 por el número de variables que puede tomar la variable 2 por ... el número de valores que puede tomar la variable N. Así, si tenemos una aplicación donde la variable 1 puede tomar 3 valores, la 2 puede tomar 5, y la 3 puede tomar 4, entonces necesitaremos 3 x 5 x 4 = 60 reglas distintas.

8 Existen reglas para simplificar las tablas de decisiones y combinar diversas reglas en reglas compuestas, pero no se verán en este libro. Véase [Yourdon, 1976] para mayores detalles.

Otra ventaja del enfoque de la tabla de decisiones es que no implica ninguna forma particular de implantación. Es decir, cuando el analista entrega la tabla (junto con los DFD, y otras cosas) al diseñador/programador, hay una tremenda libertad de elección en términos de la estrategia de implantación: la tabla de decisiones puede programarse con afirmaciones anidadas tipo SI, con una construcción CASO o con una construcción GO TO DEPENDING ON de COBOL; en el caso extremo, un generador de código de tabla de decisiones puede generar código *automáticamente* desde la tabla. Por ello, a menudo se conocen las tablas de decisiones como una herramienta de modelado de sistemas que *no es de tipo procedimiento,* pues no especifican algún algoritmo de procedimiento específico para realizar las acciones requeridas.

En resumen, deben seguirse los siguientes pasos para crear una tabla de decisiones para una especificación de proceso:

1. Identificar todas las condiciones, o variables, de la especificación. Identificar todos los valores que cada variable pueda tomar.
2. Calcular el número de combinaciones de las condiciones. Si todas las condiciones son binarias, entonces existen 2^N combinaciones de N variables.
3. Identificar cada posible acción que se pide en la especificación.
4. Crear una tabla de decisiones "vacía", listando todas las condiciones y acciones en el lado izquierdo y numerando las combinaciones de las condiciones en la parte superior de la tabla.
5. Listar todas las combinaciones de condiciones, una para cada columna de la tabla.
6. Examinar cada columna (conocida como regla) e identificar las acciones apropiadas que se deben tomar.
7. Identificar con el usuario las omisiones, contradicciones o ambigüedades.

11.4 OTRAS HERRAMIENTAS DE ESPECIFICACION DE PROCESO

11.4.1 Gráficas y diagramas

En algunos casos puede ser apropiado expresar una especificación de proceso como una gráfica o diagrama. De hecho, el usuario pudiera tener *ya* una gráfica o diagrama que se esté utilizando para llevar a cabo aquella parte de la aplicación. De ser así, úsela. No hay necesidad de que el analista traduzca la gráfica a lenguaje estructurado; en lugar de ello, deje que el *programador* traduzca la gráfica directamente a COBOL, FORTRAN o algún otro lenguaje de programación, cuando sea el momento de implantar el sistema.

Considere, por ejemplo, una especificación de proceso que determina el monto del seguro del cliente como función de su edad. El usuario dijo que la política actual de negocio es determinar el monto a partir de la gráfica que se muestra en la figura 11.8.

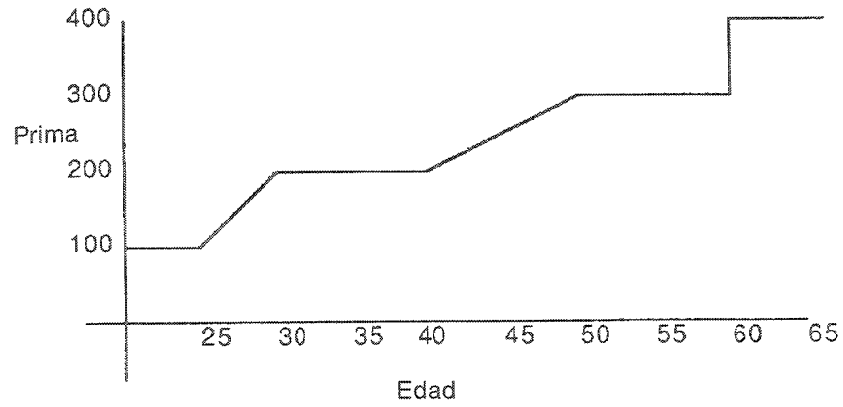


Figura 11.8: **Prima de seguros como función de la edad**

Suponiendo que la política no vaya a cambiar cuando se construya un nuevo sistema, y suponiendo que el monto del seguro sea *sólo* función de la edad, no hay necesidad de que el analista haga más trabajo. La figura 11.8 es la especificación del proceso.

11.4.2 Lenguaje narrativo

Como se ha indicado varias veces en este capítulo, el lenguaje narrativo no es una herramienta recomendable para escribir especificaciones de proceso. Esto se debe a:

- Un vocabulario no restringido (es decir, uso indiscriminado de sujetos, verbos y adjetivos) hace que sea probable que la descripción del proceso incluya términos que no estén en el diccionario de datos y cuyo significado no quede claro.
- Las acciones alternativas (es decir, decisiones) a menudo se expresan de una manera burda y ambigua. Esto se vuelve aún más peligroso cuando se expresan decisiones anidadas.
- Las acciones receptivas (es decir, ciclos) también se expresan de una manera burda y ambigua. Los ciclos anidados son extremadamente peligrosos cuando se expresan en lenguaje coloquial.

- El concepto de estructuras de bloque sólo se puede expresar con sangrías o con una presentación de "silueta". Si se está dispuesto a llegar hasta aquí, bien puede usarse de una vez la notación formal del lenguaje estructurado.

Si por alguna razón se ve obligado a usar lenguaje narrativo, debe por lo menos mantener algunas de las ventajas del enfoque altamente diferenciado del análisis estructurado que hemos discutido a lo largo de todo este libro. Es decir, bajo *ninguna* circunstancia debe permitirse tener que llegar a escribir una especificación monolítica de novela victoriana de 2 000 páginas. Por lo menos, divida la especificación en porciones pequeñas, de modo que puedan escribirse 2 000 "cuentos cortos" independientes.

11.4.3 Diagramas de flujo

Se ha evitado hasta ahora el uso de diagramas de flujo en el análisis, pero esto es reflejo de la falta de interés actual por ellos más que una denuncia.[9] Mucho de la crítica hacia ellos resultó de su mal uso en las dos siguientes áreas:

1. Como herramienta de alto nivel de modelado de *sistemas,* los diagramas de flujo son muy malos. Un diagrama de flujo muestra una lógica secuencial y de tipo procedimiento; como se vio en el capítulo 9, los DFD son una herramienta más apropiada para modelar una red de procesos no sincronizados y comunicados entre sí.

2. No hay nada que impida que el analista pueda crear un diagrama de flujo no estructurado y arbitrariamente complejo, del tipo que se muestra en la figura 11.9.

Sin embargo, *si* el diagrama de flujo se usa sólo para describir lógica detallada y *si* el analista de sistemas se limita a los símbolos de elaboración de diagramas de flujo equivalentes a las construcciones del español (u otro lenguaje) estructurado que se exponen en la sección 11.1, entonces no tiene nada de incorrecto su uso. Para crear un diagrama de flujo estructurado, el analista de sistemas tiene que organizar su lógica con las combinaciones anidadas de los símbolos de diagrama de flujo que se muestran en la figura 11.10.[10]

9 Sin embargo, es interesante notar que los diagramas de flujo pudieran estar a punto de experimentar un renacimiento. El trabajo reciente de David Scanlan de la Universidad Estatal de California en Sacramento muestra que los estudiantes de programación prefieren rotundamente los diagramas de flujo para aprender algoritmos. Si esto resulta cierto para los estudiantes de programación, pudiera serlo también para los usuarios. Para más detalles sobre esto, vea el documento de Scanlan titulado: "A Niche for Structured Flowcharts", *Proceedings of the 1987 ACM Computer Science Conference.*

10 Para más información sobre los diagramas de flujo estructurados, véase el texto clásico, [Böhm y Jacopini, 1966].

Una alternativa es el uso de los diagramas de Nassi-Shneiderman, que se discuten en la sección 11.4.4. Sin embargo, debe señalarse que *muy pocos* analistas utilizan diagramas de flujo para especificaciones de proceso (ni, para el caso, para

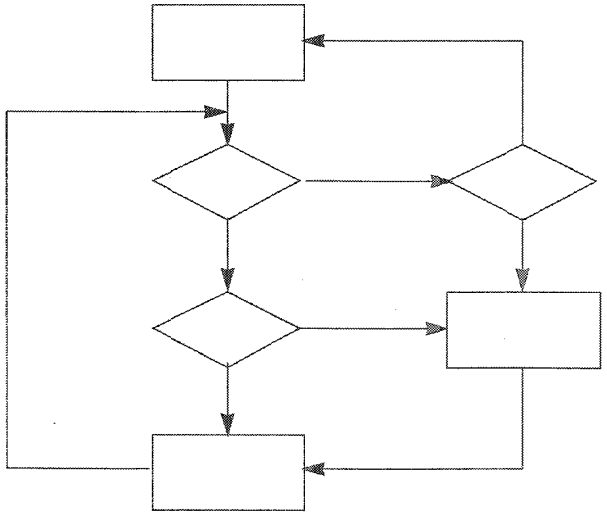


Figura 11.9: **Un diagrama de flujo no estructurado**

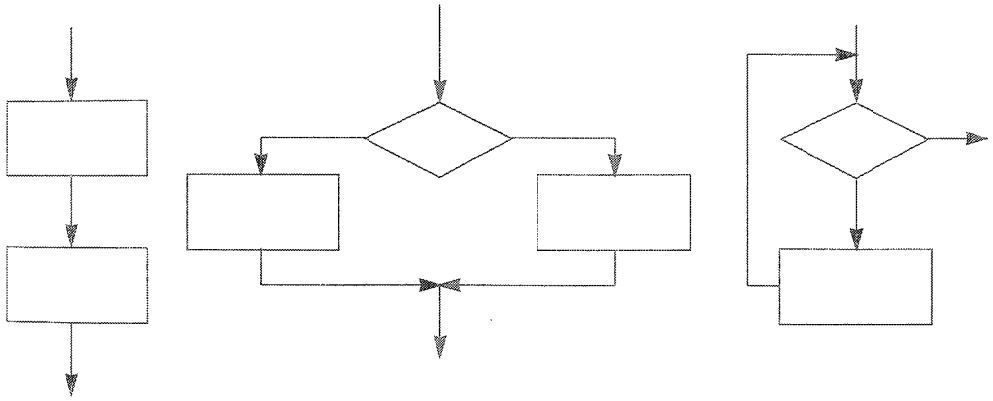


Figura 11.10: **Los símbolos de Böhm-Jacopini en un diagrama de flujo estructurado**

diseño tampoco). Aunque las herramientas automatizadas que se describen en el Apéndice A pueden usarse para crear y mantener diagramas de flujo, la verdad es que el lenguaje estructurado, las tablas de decisiones y las especificaciones de pre/post condiciones son más fáciles de crear y mantener.

11.4.4 Los diagramas de Nassi-Shneiderman

Cuando por primera vez se empezó a volver popular la programación estructurada a mediados de los años 70, los diagramas de Nassi-Shneiderman se introdujeron como una técnica estructurada de creación de diagramas de flujo; véase [Nassi y Shneiderman, 1973] y [Chapin, 1974]. Un diagrama típico Nassi-Shneiderman tiene la forma que se muestra en la figura 11.11.

Nótese que una afirmación sencilla imperativa se representa por medio de un rectángulo, como muestra la figura 11.12(a); el rectángulo también puede utilizarse para representar un *bloque* de afirmaciones secuenciales. La construcción binaria **SI-ENTONCES-OTRO** se representa por medio de la notación gráfica de la figura 11.12(b); y la construcción repetitiva **HACER-MIENTRAS** se representa por la notación gráfica de la figura 11.12(c).

Los diagramas de Nassi-Shneiderman usualmente son más organizados, más estructurados y más comprensibles que un diagrama de flujo típico; por ello, a veces se los prefiere como herramienta para crear especificaciones de proceso. Sin embargo, aún requieren una cantidad no trivial de gráficos, y no está claro que los gráficos añadan mucho valor. Como se ha oído murmurar a muchos analistas tras pasarse una hora creando un diagrama de Nassi-Shneiderman, "Esto es sólo lenguaje estructurado con cajas dibujadas alrededor de las instrucciones".

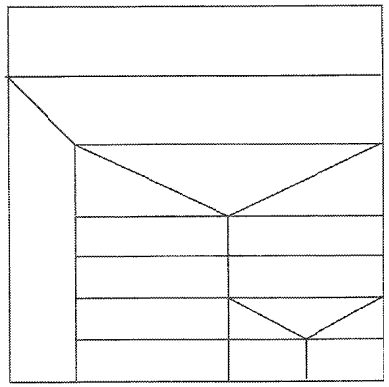


Figura 11.11: **Diagrama típico de Nassi-Schneiderman**

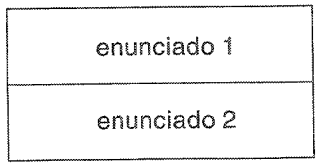


Figura 11.12(a): **Representación de un enunciado secuencial**

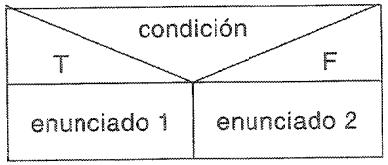


Figura 11.12(b): **Representación de la construcción SI-ENTONCES-OTRO**

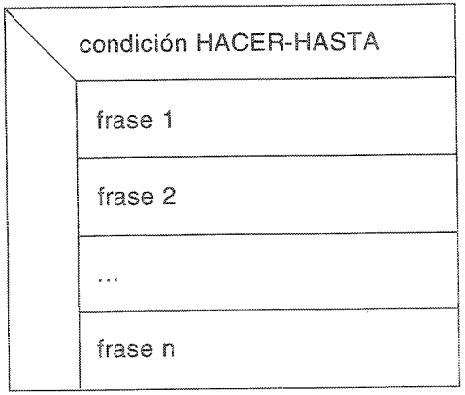


Figura 11.12 (c): **Representación de una construcción HACER-MIENTRAS**

Por otro lado, investigaciones recientes llevadas a cabo por David Scanlan en la Universidad Estatal de California [Scanlan, 1987] muestran que del 75 al 80 por ciento de los estudiantes de computación prefieren *con mucho* los diagramas de Nassi-Shneiderman al pseudocódigo al estudiar algoritmos complejos; aunque esto no coincide con la reacción negativa típica de los programadores con experiencia hacia los diagramas de flujo, las conclusiones de Scanlan se basaron en estudios cuidadosos y analíticos de factores en una muestra de varios cientos de estudiantes. Aunque los usuarios finales no necesariamente tienen las mismas preferencias que los estudiantes de computación, existe por lo menos la posibilidad de que prefieran la representación gráfica de una especificación de proceso a una narrativa textual.

11.5 RESUMEN

El propósito de este capítulo fue mostrar que hay muchas maneras diferentes de describir la política detallada del usuario dentro de cada burbuja primitiva en un DFD. Aunque el lenguaje estructurado sea la técnica más comúnmente usada en la actualidad, debe considerarse el uso de tablas de decisiones, diagramas de flujo, pre/post condiciones o cualquier otro enfoque que pueda verificarse y comunicarse fácilmente a sus usuarios.

Tenga en mente que las especificaciones del proceso representan la mayor parte del trabajo detallado que se tiene en la construcción de un modelo de sistemas; pueden existir cientos, o incluso miles, de especificaciones de proceso, cada una de las cuales mida una página. Debido a la cantidad de trabajo involucrado, podría considerar el enfoque de implantación descendente que se discutió en el capítulo 5: comenzar la fase de diseño e implantación de su proyecto antes de que hayan concluido todas las especificaciones de proceso.

Tenga en mente también que la actividad de escribir especificaciones de proceso sirve como "prueba de cordura" para los DFD que ya se hayan desarrollado. Podría descubrirse que la especificación del proceso requiere flujos de datos de entrada o de salida adicionales (es decir, flujos que no aparecieron en el DFD). Y al escribirla podría también encontrarse que se necesitan funciones adicionales; por ejemplo, al escribir la especificación para una función que añade un nuevo registro al almacén de **CLIENTES,** podría notarse que el DFD no tiene una burbuja que modifique o elimine un registro de dicho almacén. Así que se pueden esperar cambios, revisiones y correcciones del modelo de DFD, basadas en el trabajo detallado de la escritura de las especificaciones del proceso.

REFERENCIAS

1. Tom DeMarco, *Structured Analysis and Systems Specification.* Englewood Cliffs, N.J.: Prentice-Hall,1979.
2. Chris Gane y Trish Sarson, *Structured Systems Analysis: Tools and Techniques.* Englewood Cliffs, N.J.: Prentice-Hall, 1978.
3. Edward Yourdon, *Techniques of Program Structure and Design,* 2ª edición, Englewood Cliffs, N.J.: Prentice-Hall, 1988.
4. James Martin y Carma McClure, *Diagramming techniques for Software Engineering.* Englewood Cliffs, N.J.: Prentice-Hall, 1985.
5. Victor Weinberg, *Structured Analysis.* Englewood Cliffs, N.J.: Prentice-Hall, 1978.
6. Edward Yourdon, *Classics in Software Engineering.* New York: YOURDON Press, 1979.

7. Corrado Böhm y Guiseppe Jacopini, "Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules," *Communications of the ACM,* Vol.9, número 5 (mayo de 1966), pp. 366-371. También reimpreso en *Classics in Software Engineering* (op. cit.).
8. I. Nassi y B. Shneiderman, "Flowchart Techniques for Structured Programming", *ACM SIGPLAN Notices,* Vol. 8, número 8 (agosto de 1973), pp. 12-26.
9. Ned Chapin, "New Format for Flowcharts", *Software–Practice and Experience,* Vol. 4, número 4 (octubre a diciembre, 1974), pp. 341-357.
10. H. McDaniel, editor, *Application of Decision Tables.* Princeton, N.J.: Brandon Systems Press, 1970.
11. S. Pollack, H. Hicks, y W. Harrison, Decision Tables: Theory and Practice. Nueva York, Willey, 1971.
12. T.R. Gildersleeve, *Decision Tables and their Practical Applications in Data Processing.* Englewood Cliffs, N.J.: Prentice-Hall, 1970.
13. David Scanlan, "Cognitive Factors in Preference for Structured Flowcharts: A Factor Analytic Study," presentado en la primera conferencia de autores de YOURDON Press, Nueva York, 5 de diciembre de 1987.

PREGUNTAS Y EJERCICIOS:

1. Considere la siguiente especificación, que se da en forma *narrativa.* ¿Cuál de las herramientas de especificación que se presentaron en este capítulo cree que sería la más apropiada? ¿Por qué?

   > Cuando me llega una solicitud de compra, mi tarea consiste en escoger a un proveedor de nuestro archivo de proveedores disponibles. Desde luego, algunos proveedores se eliminan de inmediato porque sus precios son demasiado elevados, o porque se han puesto temporalmente en la "lista negra" por su baja calidad. Pero mi verdadera tarea es escoger al mejor proveedor de entre los que calificar: el que entregará nuestro pedido en el tiempo más corto. Mi jefe tenía un sistema para estimar el tiempo de entrega, y me lo enseñó, pero ahora sólo veo dónde está el proveedor, la cantidad de artículos pedidos y la fecha en la que necesitamos las cosas, y *sé cuál* proveedor es el mejor... ya ni siquiera estoy seguro de cómo lo hago.

2. ¿Qué es una especificación de proceso? ¿Cuáles son sus objetivos?
3. ¿Cuáles son las cinco herramientas comunes para modelado de especificaciones de proceso?
4. ¿Cuáles son los tres requerimientos que debe satisfacer una especificación de proceso?

5. ¿Debe un proyecto de desarrollo de sistemas utilizar *una sola* herramienta para las especificaciones de proceso? ¿Por qué?

6. Proyecto de investigación: ¿Qué herramientas de especificación se utilizan en su organización? ¿Existen restricciones sobre qué herramientas deban usarse? ¿Cree que se estén usando las herramientas *correctas?*

7. ¿Cuáles burbujas de un DFD requieren especificaciones de proceso?

8. ¿Cuáles son las consecuencias de escribir especificaciones de proceso para burbujas no atómicas (no primitivas)?

9. ¿Cómo debe el analista determinar las especificaciones de proceso para una burbuja?

10. ¿Cómo es que a veces las especificaciones de proceso imponen decisiones arbitrarias de diseño e implantación? ¿Cuáles son las consecuencias de esto?

11. Proyecto de investigación: Encuentre un ejemplo de especificación de proceso en su organización que muestre decisiones de diseño o implantación arbitrarias. ¿Cómo lo reescribiría para evitar ese problema?

12. Dé una definición del término Lenguaje Estructurado. ¿Qué sinónimos hay para este término?

13. ¿Cuántos verbos se necesitan para formar frases del lenguaje estructurado? Sugiera una lista de 20 verbos.

14. ¿Por qué se necesitan usualmente ecuaciones algebraicas en una especificación de proceso en lenguaje estructurado?

15. ¿Qué características debieran tener los objetos en una especificación de proceso en lenguaje estructurado?

16. ¿Qué son los términos locales?

17. ¿Deben incluirse términos locales en el diccionario de datos? ¿Por qué?

18. ¿Aparecen los términos locales como flujos en los DFD?

19. Dé un ejemplo específico de un término local.

20. ¿Qué construcciones de la programación estructurada se utilizan en el lenguaje estructurado?

21. ¿Cuál es el propósito del término **OTRO** en el lenguaje estructurado?

22. ¿Cuál es la diferencia entre la construcción **HACER-MIENTRAS** y la construcción **REPITE-HASTA** en el lenguaje estructurado?

23. ¿Qué es una frase compuesta?

24. ¿Se necesita la construcción **CASO** en el lenguaje estructurado? ¿Cuáles son sus ventajas y desventajas?

25. ¿Cuáles son los cuatro componentes de una frase compuesta?

26. ¿Cuál es la diferencia entre (a) y (b)?

(a)
```
SI A ENTONCES B
        SI B ENTONCES
                frase-1
        OTRO
                frase-2
OTRO
        frase-2
```

(b)
```
SI A Y B ENTONCES
        frase-1
OTRO
        frase-2
```

¿Cuál de estos dos ejemplos cree que sea más fácil de entender? ¿Por qué?

27. Proyecto de investigación: Construya varios ejemplos similares al de la pregunta 26, y haga una encuesta entre sus usuarios para ver cuál versión prefieren.

28. ¿Cuáles son las tres reglas que deben seguirse para asegurarse de que una especificación en lenguaje estructurado será legible?

29. ¿Cree Ud. que tres niveles de construcciones anidadas **SI** sean correctos en una especificación en lenguaje estructurado? ¿Por qué?

30. Proyecto ce investigación: Haga varios ejemplos de especificaciones de proceso en lenguaje estructurado que empleen dos, tres y cuatro niveles de construcciones SI anidadas. Haga una encuesta para determinar, en promedio, cuántos niveles están dispuestos a aceptar los usuarios en su organización.

31. ¿Cuál es el verdadero propósito de la sangría en una especificación de proceso en lenguaje estructurado?

32. ¿Por qué es importante hacer revisiones de las especificaciones de proceso en lenguaje estructurado con el usuario apropiado?

33. ¿Cuál es el propósito de utilizar una numeración de tipo "silueta" en una especificación de proceso en lenguaje estructurado?

34. Dé una definición de una técnica de especificación de precondición/postcondición.

35. ¿Cuál es la diferencia entre la técnica de especificación del lenguaje estructurado y la de precondición/postcondición?

36. Bajo qué condiciones es buena la técnica de especificación de precondición/postcondición para que la use el analista?

37. Dé una definición de precondición.

38. ¿Cuáles son las cuatro cosas que usualmente describe una precondición?

39. ¿Cuál es la relación entre precondiciones en una especificación de proceso y flujo de entrada en un DFD?

40. ¿Qué pasa si *no* se satisfacen las precondiciones en una especificación de proceso?

41. Dé una definición de postcondición.

42. ¿Cuáles son las cuatro cosas que típicamente describe una postcondición?

43. ¿Cuál es la relación entre postcondiciones y flujos de salida en un DFD?

44. Si una especificación de proceso tiene cuatro conjuntos de precondiciones, ¿cuántos conjuntos de postcondiciones debe tener?

45. ¿Cuál es el número mínimo de precondiciones que pudiera tener una especificación de proceso?

46. ¿Cuáles son las desventajas en potencia del enfoque precondición/postcordición?

47. Proyecto de investigación: Encuentre un ejemplo de una especificación real a la que no le quede bien el enfoque de precondición/postcondición. ¿Por qué no le quedaría bien?

48. Vea la figura 11.6. ¿Cuál sería una herramienta de modelado mejor para crear una especificación de proceso para esa situación?

49. Proyecto de investigación: Encuentre un ejemplo de una especificación de proceso real a la que sí le quede bien el enfoque de precondición/postcondición. ¿Por qué le quedaría bien?

50. ¿Qué es una tabla de decisiones? ¿Cuáles son sus componentes?

51. ¿Bajo qué condiciones la tabla de decisiones es una buena herramienta para modelar especificaciones?

52. ¿Qué tiene mal la siguiente tabla de decisiones?

Edad mayor de 21 años	V	F	F
Sexo Masculino	V	V	F
Medicamento 1	X		
Medicamento 2		X	

53. ¿Qué tiene mal la siguiente tabla de decisiones?

Estatura mayor de 1.8O m	V	V	F	F
Peso mayor de 100	V	F	V	F
Prima de seguro = $100	X		X	
Prima de seguro = $200		X		X
Prima de seguro = $300				

54. ¿Qué tiene mal la siguiente tabla de decisiones?

Estatura mayor de 1.8O m	V	V	F	F
Peso mayor de 100	V	F	V	F
Prima de seguro = $100	X	X		X
Prima de seguro = $200			X	
Prima de seguro = $300		X		

55. ¿Qué diferencia hay entre una tabla de decisiones con variables binarias y una con variables multivaluadas?

56. Si una tabla de decisiones tiene N variables binarias, ¿cuántas reglas debe tener?

57. ¿Bajo qué condiciones debiera el analista usar una gráfica para una especificación de proceso?

58. ¿Cuáles son las ventajas que tiene una gráfica sobre el lenguaje estructurado como herramienta para el modelado de especificaciones de proceso?

59. ¿Cuáles son las cuatro principales desventajas del lenguaje narrativo como herramienta para el modelado de especificaciones de proceso?

60. ¿Qué reglas se deben seguir en caso de que se tenga por fuerza que usar lenguaje narrativo como herramienta de modelado para especificaciones de proceso?

61. ¿Cuáles son las dos críticas que comúnmente se hacen a los diagramas de flujo como herramienta de modelado?

62. ¿Cuáles son las tres componentes de un diagrama de flujo estructurado?

63. Proyecto de investigación: Muestre la misma especificación a un usuario en la forma de lenguaje estructurado y en forma de diagrama de flujo. ¿Qué enfoque se prefiere? Para más información, vea [Scanlan, 1987].

64. ¿Qué es un diagrama de Nassi-Shneiderman? ¿Cuál es la diferencia entre un diagrama de flujo y un diagrama de Nassi-Shneiderman?

65. Tomado de *Structured Analysis,* de Victor Weinberg (Nueva York: YOURDON Press, 1978): Escriba una tabla de decisiones para la siguiente especificación narrativa, e indique las omisiones, ambigüedades o contradicciones que encuentre:

> "La tienda Swell emplea a un cierto número de vendedores para vender una variedad de artículos. La mayoría de estos vendedores obtienen sus ingresos de comisiones sobre los artículos que venden, pero existen algunos empleados que obtienen salario más comisión; es decir, obtienen un salario fijo, sin importar el tipo o cantidad de artículos que venden, más una comisión sobre ciertos artículos. La tienda Swell tiene diversas líneas de mercancía, algunas de las cuales se clasifican como artículos estándar (por ejemplo, una lata de sopa de tomate) porque son de uso común y no requieren de técnicas creativas de venta; además, hay artículos extra que son altamente remunerativos pero difíciles de vender (como un Cadillac de chapa de oro, incrustado con diamantes). Los artículos estándar y los extra generalmente representan los límites inferior y superior del espectro de precios, con un mayor número de artículos enmedio.

> Los compradores también se clasifican. Algunos se conocen como regulares, pues hacen transacciones tan a menudo que no se requiere hacerles venta creativa. Sin embargo, la mayoría de los clientes hace pocas transacciones en la tienda Swell, y es probable que entren, compren algo y no vuelvan a ser vistos.
>
> La política de comisiones de la gerencia es la siguiente: si un empleado no asalariado vende un artículo que no es estándar ni extra a una persona que no sea compradora regular, recibe un 10% de comisión, a menos que el artículo cueste más de $10,000, en cuyo caso la comisión es del 5 por ciento. Para todos los vendedores, si se vende un artículo estándar, o si se le vende algo a un cliente regular, no se da comisión alguna. Si un vendedor asalariado vende un artículo extra, recibe una comisión del 5%, a menos de que el artículo se venda a más de $1,000, en cuyo caso recibe una comisión neta de $25. Si un vendedor no asalariado vende un artículo extra a alguien que no sea un comprador regular, recibe un 10% de comisión, a menos de que el artículo cueste más de $1,000, en cuyo caso recibe una comisión neta de $75".

12 DIAGRAMAS DE ENTIDAD-RELACION

Obviamente, el juicio de un hombre no puede superar la información en la que él lo basó. Désele la verdad y aun así pudiera errar teniendo la oportunidad de acertar; pero no le den noticias, o dénsele sólo datos incompletos y distorsionados, mostrados de manera ignorante, descuidada o prejuiciosa, con propaganda y falsedades, y se destruirán todos sus procesos de razonamiento, y él se convertirá en algo menos que un hombre.

Arthur Hays Sulzberger
Discurso, Asociación de Editores del Estado de Nueva York, 1948

En este capítulo se aprenderá:

1. Por qué son útiles los modelos de datos en el análisis de sistemas.
2. Los componentes de un diagrama de entidad-relación.
3. Cómo escribir un diagrama de entidad-relación.
4. Cómo refinar un diagrama inicial de entidad-relación.

En este capítulo se explora una notación gráfica para modelar datos. El *diagrama de entidad-relación* (también conocido como DER, o diagrama E-R) es un modelo de red que describe con un alto nivel de abstracción la distribución de datos almacenados en un sistema. Es muy diferente del DFD, que modela las *funciones* que lleva a cabo un sistema; y es diferente del *diagrama de transición de estados*, que modela el comportamiento dependiente del tiempo de un sistema.

¿Por qué podríamos estar interesados en modelar los datos de un sistema? Primariamente, porque las estructuras de datos y las relaciones pueden ser tan complejas que se deseara enfatizarlas y examinarlas *independientemente* del proceso que se llevará a cabo. De hecho, esto se da sobre todo si mostramos el modelo del sistema a los usuarios ejecutivos de mayor nivel de una organización (por ejemplo el vicepresidente o los gerentes de departamento, quienes podrían no estar interesados en los detalles funcionales cotidianos del sistema). Tales usuarios a menudo se preocupan más por los datos: ¿qué datos requerimos para manejar nuestro negocio? ¿Quién los tiene? ¿Quién tiene acceso a ellos?

Algunas de estas preguntas, por ejemplo, la tenencia o acceso a los datos, pudieran ser responsabilidad de un grupo dedicado dentro de la organización. A menudo, el grupo de *administración de datos* (grupo AD) es responsable de administrar y controlar la informacion esencial de un negocio; cuando se comience a construir un nuevo sistema de información se requerirá hablar con estas personas para poder coordinar la información del sistema con el modelo global, corporativo, de información. *El diagrama de entidad-relación es una herramienta útil para llevar a cabo esta conversación.*

A menudo existe otro grupo dentro de la administración con un nombre similar: el grupo de *administración de bases de datos* (a veces conocido como grupo de ABD). Se suele localizar dentro del departamento de proceso de datos (mientras que el de administración de datos no necesariamente está ahí), y su labor es asegurar que las bases de datos computarizadas se organicen, administren y controlen de manera eficiente. Así que suele ser el equipo de implantación responsable de tomar el modelo esencial (independiente de alguna tecnología específica) y traducirlo a un diseño de base de datos eficaz para IMS, ADABAS, IDMS, TOTAL o algún otro sistema de base de datos. *El diagrama de entidad-relación es una herramienta efectiva de modelado para comunicarse con el grupo de administración de bases de datos.* Basándose en la información presentada por el DER, el grupo de administración de base de datos puede ver el tipo de claves o índices o apuntadores que se necesitarán para llegar de manera eficiente a los registros de las bases de datos.

Para el analista, el DER representa un gran beneficio también: enfatiza las relaciones entre almacenes de datos en el DFD que de otra forma se hubieran visto sólo en la especificación de proceso. Por ejemplo, un DER típico se muestra en la figura 12.1. Cada una de las cajas rectangulares corresponde a un almacén de datos en un DFD (correspondencia que se explorará más a fondo en el Capítulo 14), y puede verse que hay relaciones (conexiones) que normalmente no se aprecian en un DFD. Esto se debe a que el DFD enfoca la atención del lector a las *funciones* que el sistema efectúa, no a los datos que ocupa.

Considere un caso extremo: ¿Qué tal si no se están realizando funciones? ¿Qué tal si el propósito del sistema que está construyendo no es *hacer* algo, sino meramente ser el recipiente de una gran cantidad de información interesante? Tal

sistema pudiera llamarse sistema de consultas ad hoc, o sistema de apoyo a decisiones. En tal tipo de sistema, pudiéramos concentrarnos por completo en el modelo de datos, y ni siquiera preocuparnos por construir el modelo de DFD, que está orientado a las funciones. Desde luego, esto es claramente una situación rara: la mayoría de los sistemas sí efectúan funciones; a menudo, encontramos que construir primeramente el modelo de datos hace más fácil descubrir cuáles son las funciones requeridas.

Desde luego, la notación del DER de la figura 12.1 es bastante misteriosa hasta el momento. En las siguientes secciones se examinará la estructura y los componentes de un DER para luego discutir las reglas para dibujar un DER bien estructurado. La notación que se presenta en este capítulo se deriva de [Flavin, 1981] y es similar a la que desarrollaron [Chen, 1976], [Martin, 1982], [Date, 1986] y otros.

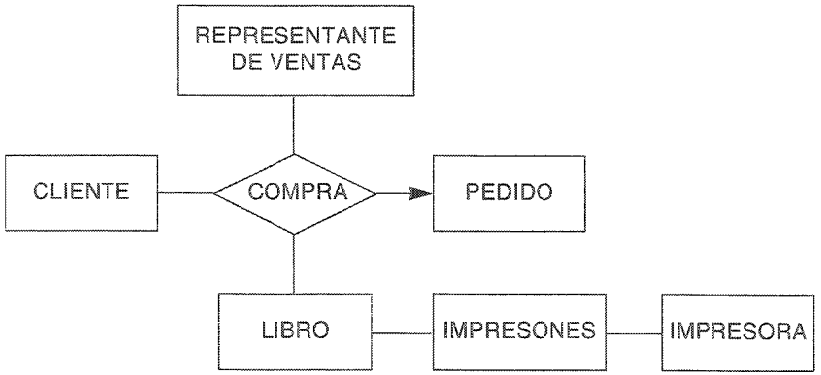


Figura 12.1: **Diagrama de entidad-relación típico**

12.1 LOS COMPONENTES DE UN DER

Hay cuatro componentes principales en un diagrama de entidad-relación:

1. Tipos de objetos
2. Relaciones
3. Indicadores asociativos de tipo de objeto
4. Indicadores de supertipo/subtipo

12.1.1 Tipos de objetos

El tipo de objeto se representa en un diagrama de entidad-relación por medio de una caja rectangular; en la figura 12.2 se muestra un ejemplo. Representa una

colección o conjunto de *objetos* (cosas) del mundo real cuyos miembros individuales (o instancias) tienen las siguientes características:

- *Cada una puede identificarse de manera única por algún medio.* Existe alguna forma de diferenciar entre instancias individuales del tipo de objeto. Por ejemplo, si se tiene un tipo de objeto conocido como **CLIENTE,** debemos ser capaces de distinguir uno de otro (tal vez por un número de cuenta, por su apellido, o por su número de Seguro Social). Si todos los clientes son iguales (si hay un negocio en el que son sólo entes sin cara y sin nombre que entran a la tienda a comprar cosas), entonces **CLIENTE** no sería un tipo de objeto con significado.

Figura 12.2: **Un tipo de objeto**

- *Cada uno juega un papel necesario en el sistema que se construye.* Es decir, para que el tipo de objeto sea legítimo, debe poder decirse que *el sistema no puede operar sin tener acceso a esos miembros.* Si se está construyendo un sistema de ingreso de pedidos para la tienda, por ejemplo, se pudiera pensar que, además de los clientes, la tienda tiene mozos, cada uno de los cuales se identifica de manera individual por su nombre. A pesar de que los mozos juegan un papel útil en la tienda, el sistema de ingreso de pedidos puede funcionar felizmente sin ellos; por tanto, no merecen un papel como tipos de objeto en el modelo del sistema. Obviamente, esto es algo que debe verificarse con los usuarios al construir el modelo.

- *Cada uno puede describirse por uno o más datos.* Es decir, un **CLIENTE** puede describirse por medio de datos tales como nombre, domicilio, límite de crédito y número telefónico. Muchos textos sobre bases de datos describen esto como "asignar datos a un tipo de objeto". Nótese que los atributos deben aplicarse a *cada instancia* del tipo de objeto; por ejemplo, cada cliente debe tener nombre, domicilio, límite de crédito, número telefónico, etc.

En muchos de los sistemas que desarrolle, los tipos de objetos serán la representación en el sistema de *algo material* del mundo real. Esto significa que los clientes, artículos de inventario, empleados, partes manufacturadas, etc, son objetos

típicos. El *objeto* es el algo material del mundo real, y el *tipo de objeto* es su representación en el sistema. Sin embargo, un objeto también pudiera ser algo no material: por ejemplo, horarios, planes, estándares, estrategias y mapas.

Dado que a menudo las personas son tipos de objetos en un sistema, debe tenerse otra cosa en mente: una persona (o para el caso, cualquier cosa material) pudiera ser diversos tipos de objetos distintos en distintos modelos de datos, *o incluso en un mismo modelo.* Juan Pérez, por ejemplo, puede ser **EMPLEADO** en un modelo de datos y **CLIENTE** en otro. También pudiera ser **EMPLEADO** y **CLIENTE** dentro del mismo modelo.

Nótese que en todos los ejemplos de un objeto se ha usado un sustantivo singular (por ejemplo, empleado, cliente). Esto no es necesario, pero es un convenio útil; como veremos en el Capítulo 14, existe una correspondencia entre objetos en el DER y almacenes en el DFD; así, si existe un objeto **CLIENTE** en el DER, debe haber un almacén de **CLIENTES** en el DFD.

12.1.2 Relaciones

Los objetos se conectan entre sí mediante *relaciones.* Una relación representa un conjunto de conexiones entre objetos, y se representa por medio de un rombo. La figura 12.3 muestra una relación sencilla que pudiera existir entre dos o más objetos.

Figura 12.3: **Una relación**

Es importante reconocer que la relación representa un *conjunto de conexiones.* Cada instancia de la relación representa una asociación entre cero o más ocurrencias de un objeto y cero o más ocurrencias del otro. Así, en la figura 12.3, la relación etiquetada como **COMPRAS** puede contener las siguientes instancias individuales:

- instancia 1: el cliente 1 compra el artículo 1
- instancia 2: el cliente 2 compra los artículos 2 y 3
- instancia 3: el cliente 3 compra el artículo 4
- instancia 4: el cliente 4 compra los artículos 5, 6 y 7

- instancia 5: el cliente 5 *no* compra ningún artículo
- instancia 6: los clientes 6 y 7 compran el artículo 8
- instancia 7: los clientes 8, 9 y 10 compran los artículos 9, 10 y 11
- etc.

Como puede verse, entonces, una relación puede conectar dos o más instancias del mismo objeto.

Nótese que la relación representa algo que debe ser *recordado* por el sistema: algo que no pudo haberse calculado ni derivado mecánicamente. Así, el modelo de datos de la figura 12.3 indica que existe alguna razón relacionada con el usuario para recordar el hecho de que el cliente 1 compra el artículo 1, etc. Y la relación también indica que no existe nada *a priori* que hubiera permitido determinar que el cliente 1 compró el artículo 1 y nada más. *La relación representa la memoria del sistema.*[1] (Un objeto representa la memoria del sistema también, desde luego.)

Nótese también que puede existir más de una relación entre dos objetos. La figura 12.4, por ejemplo, muestra dos relaciones distintas entre un **PACIENTE** y un **MEDICO.** A primera vista, pudiera pensarse que esto es rebuscar lo obvio: cada vez que el médico trata a un paciente, le cobra mediante un recibo. Pero la figura 12.4 sugiere que la situación puede ser distinta; pudiera resultar, por ejemplo, que existan diversas instancias de "tratamiento" entre un médico y el mismo paciente (es decir, un tratamiento inicial, tratamientos de seguimiento, etc.). La figura 12.4 implica que la relación de recibos está completamente separada de la relación de tratamiento: tal vez a algunos pacientes se les dan recibos sólo por su primer tratamiento, mientras que a otros se les dan para cada tratamiento y a otros jamás se les dan.

Una situación más común es ver múltiples relaciones entre múltiples objetos. La figura 12.5 muestra la relación que existe típicamente entre un cliente, un vendedor, un agente de bienes raíces, el abogado del cliente y el abogado del vendedor, para la compra-venta de una casa.

Con un diagrama complejo como el de la figura 12.5 (que es típico, y tal vez más simple que los DER que es probable encontrar en un proyecto real), la relación y sus tipos de objetos deben leerse como una unidad. La relación se puede describir desde la perspectiva de *cualquiera* de los tipos de objetos participantes, y *todas*

1 Entre los objetos, pueden existir otras relaciones que no aparezcan en el DER. Dado que el DER es un *modelo de datos almacenados*, no se muestran las relaciones que se pueden calcular o derivar. Por ejemplo, considere el objeto **CONDUCTOR** y el objeto **LICENCIA.** Puede existir entre los dos una relación de fecha de renovación que se ha calculado con base en la fecha de nacimiento del conductor (por ejemplo, un conductor debe renovar su licencia cada año en su cumpleaños). Tal relación no se mostraría en el DER pues no es una relación de datos almacenados. Sin embargo, si la fecha de renovación se escogiera al azar, probablemente tendría que ser recordada por el sistema.

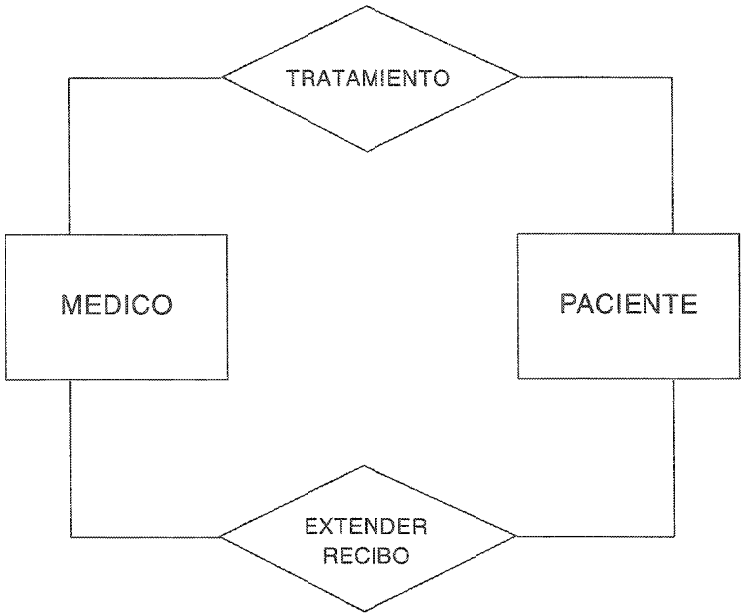


Figura 12.4: **Relaciones múltiples entre objetos**

esas perspectivas son válidas. De hecho, el conjunto de todos estos puntos de vista es el que describe completamente la relación. Por ejemplo, en la figura 12.5 puede verse la relación de negociación de precios en cualquiera de las siguientes tres formas:

1. El agente de bienes raíces negocia el precio entre el cliente y el vendedor.
2. El cliente negocia el precio con el vendedor, mediante el agente de bienes raíces.
3. El vendedor negocia el precio con el cliente, mediante el agente de bienes raíces.

Nótese que, en algunos casos, puede haber relaciones entre diferentes instancias de un mismo tipo de objeto. Por ejemplo, imagine un sistema que se esté desarrollando para una universidad, en el cual **CURSO, ESTUDIANTE** y **PROFESOR** son tipos de objetos. La mayoría de las relaciones de interés serán entre instancias de diferentes tipos de objetos (por ejemplo, las relaciones "se inscribe en", "imparte", etc.). Sin embargo, pudiera requerirse modelar la relación "es prerrequisito para" entre una instancia de **CURSO** y otra.

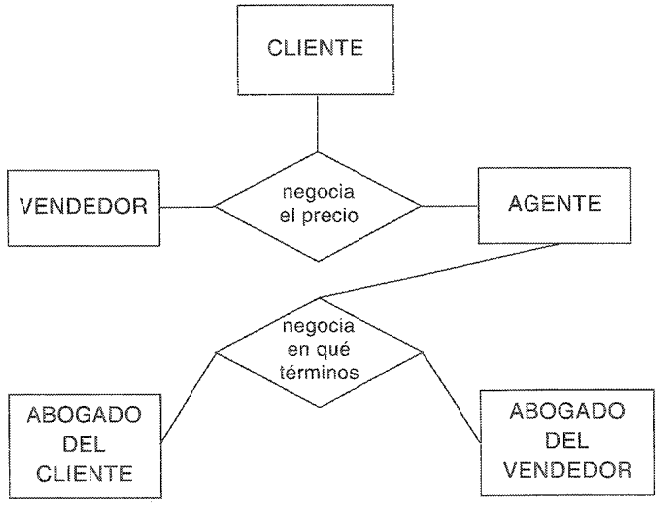


Figura 12.5: **Relaciones múltiples entre múltiples objetos**

12.1.3 Notación alternativa para relaciones

Como se vio en la Sección 12.1.2, las relaciones en el diagrama E-R son *multidireccionales;* pueden leerse siguiendo cualquier dirección. Además, vimos que los diagramas E-R no muestran *cardinalidad;* es decir, no muestran el número de objetos que participan en la relación. Esto es consciente y deliberado: se prefiere dejar tales detalles en el diccionario de datos. Esto se discutirá más a fondo en la Sección 12.3.

Una notación alternativa utilizada por algunos analistas muestra tanto la *cardinalidad* como la *ordinalidad.* Por ejemplo, la figura 12.6(a) muestra una relación entre **CLIENTE** y **ARTICULO** en la cual la notación adicional indica que:

(1) El **CLIENTE** es el punto de ancla, es decir, el objeto primario desde cuyo punto de vista debe leerse la relación.[2]

(2) La relación consiste en un cliente conectado con N artículos. Es decir, un cliente individual puede adquirir 0, 1, 2, ... o N artículos. Sin embargo, la relación incica que sólo puede haber *un* cliente involucrado en cada instancia de la relación. Esto excluye, por ejemplo, la posibilidad de que múltiples clientes estuvieran involucrados en la compra de un solo artículo

2 El término punto de ancla se introdujo en [Flavin, 1981].

Figura 12.6(a): **Notación de punto ancla para diagramas E-R**

Otra notación común aparece en la figura 12.6(b), en donde la flecha de dos puntas seguidas muestra la relación de uno a muchos, mientras que se emplea una flecha sencilla para mostrar relaciones de uno a uno entre objetos.

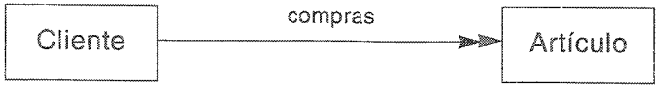


Figura 12.6(b): **Notación alternativa para relaciones de uno a muchos**

Tales diagramas de E-R anotados se discuten con detalle en [Chen, 1976], [Martin, 1982], y [Date, 1986]. Sin embargo, prefiero *no* agregar tales detalles, porque se pueden incluir fácilmente en el diccionario de datos (como se discutirá en la Sección 12.4), y porque tienden a distraer la atención del propósito principal del diagrama E-R, que es dar una visión *global* de los componentes e interfaces entre datos en un sistema. Aunque no hay nada intrínsecamente malo en tener anotaciones de tipo procedimiento en el diagrama, mi experiencia indica que los analistas a menudo llevan más allá una buena idea y atiborran el diagrama con demasiada información.

12.1.4 Indicadores asociativos de tipo de objeto

Una notación especial en el diagrama de E-R es el *indicador asociativo de tipo de objeto;* representa algo que funciona como objeto y como relación. Otra manera de ver esto es considerar que el tipo asociativo de objeto representa una *relación acerca de la cual se desea mantener alguna información.*[3]

Considere, por ejemplo, el caso sencillo de un cliente que adquiere un artículo (o artículos), que ilustra la figura 12.6. Sin tener en cuenta si se incluye o no la anotación de tipo procedimiento, el punto principal es que la *relación de* **COMPRA** *no hace más que asociar un* **CLIENTE** *con uno o más* **ARTICULOS.** Pero suponga que existen datos que deseamos recordar acerca de cada instancia de una compra (por ejemplo, a qué hora del día se hizo). ¿Dónde se podría almacenar dicha información? "Hora del día" definitivamente no es un atributo de **CLIENTE,** ni de **ARTICULO.** Más bien, se asocia "hora del día" con la compra misma, y esto se muestra en un diagrama como el que ilustra la figura 12.7.

3 En diversos libros de bases de datos esto se conoce como datos de intersección.

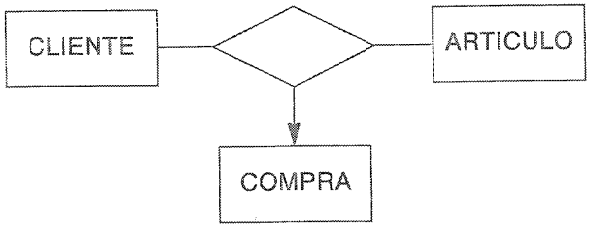


Figura 12.7: **Indicador asociativo de tipo de objeto**

Nótese que **COMPRA** ahora se escribe dentro de una caja rectangular conectada, por medio de líneas dirigidas, a un rombo de relación sin nombre. Esto pretende indicar que **COMPRA** funciona como:

- Un *tipo de objeto,* algo acerca de lo cual se desea almacenar información. En este caso, se desea recordar la hora en la cual se realizó la compra y el descuento que se dio al cliente. (Nuevamente, esto supone que tal información no la puede derivar, después del hecho, el sistema).
- Una *relación* que conecta los dos tipos de objeto **CLIENTE** y **ARTICULO.** Lo significativo aquí es que **CLIENTE** y **ARTICULO** se mantienen solos. *Existirían con o sin la compra.*[4]

Una **COMPRA,** por otro lado, obviamente depende para su misma existencia del **CLIENTE** y del **ARTICULO.** Aparece *sólo como resultado de una relación entre los otros objetos con los cuales está conectada.*

La relación de la figura 12.7 no tiene nombre a propósito. Esto se debe a que el indicador asociativo de objeto **(COMPRA)** *también* es el nombre de la relación.

12.1.5 Indicadores de subtipo/supertipo

Los tipos de objeto de subtipo/supertipo consisten en tipos de objeto de una o más subcategorías, conectados por una relación. La figura 12.8 muestra un subtipo/supertipo típico: la categoría general es **EMPLEADO** y las subcategorías son **EMPLEADO ASALARIADO** y **EMPLEADO POR HORAS.** Nótese que los subtipos se conectan al supertipo por medio de una relación sin nombre; note también que el supertipo se conecta a la relación con una línea que contiene una barra.

4 Un purista pudiera argumentar que esto no se cumple a la larga. Si no hubiera **ARTICULOS** durante varios días seguidos, los **COMPRADORES** desaparecerían de la escena y comprarían en otra parte. Y si no hubiera **COMPRADORES,** la tienda tendría que cerrar y los **ARTICULOS** desaparecerían. Pero en la situación de estado estable a corto plazo, es obvio que compradores y artículos pueden coexistir felizmente sin tener algo que ver necesariamente unos con otros.

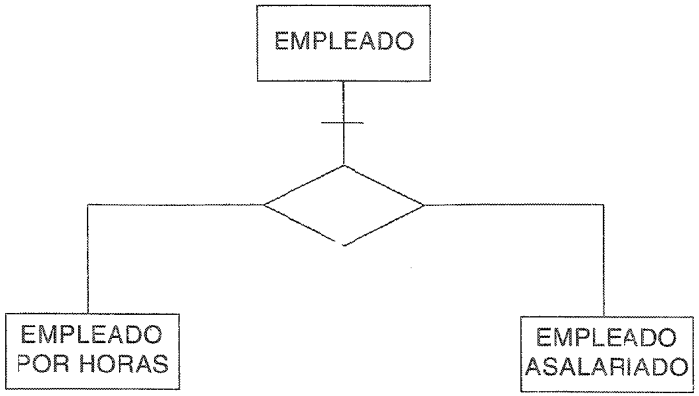


Figura 12.8: **Indicador de subtipo/supertipo**

En esta notación el supertipo se describe por datos que se aplican a *todos* los subtipos. Por ejemplo, en la figura 12.8, se podría imaginar que todos los empleados se describen por hechos tales como:

- Nombre
- Años de servicio
- Domicilio particular
- Nombre del supervisor

Sin embargo, cada subtipo se describe por medio de datos *diferentes;* de otro modo, no tendría caso hacer distinción entre ellos. Por ejemplo, se podría imaginar que un **EMPLEADO ASALARIADO** se describe por cosas tales como:

- Salario mensual
- Porcentaje anual adicional
- Aportación para coche de la empresa

Y el **EMPLEADO POR HORAS** por medio de cosas como:

- Paga por hora
- Cantidad por tiempo extra
- Hora de comienzo

12.2 REGLAS PARA LA CONSTRUCCION DE DIAGRAMAS DE ENTIDAD-RELACION

La notación que se muestra en la sección anterior es suficiente para construir diagramas E-R arbitrariamente complejos. Sin embargo, podría estar pensando en este momento: "¿Cómo descubrir qué son, para comenzar, los objetos y las relaciones?". El modelo inicial de objetos y relaciones usualmente se derivará de 1) su comprensión de la aplicación del usuario, 2) entrevistas con el usuario y 3) cualquier otro tipo de investigación y recolección de información que pueda usar. (En el Apéndice E se discuten técnicas de entrevista y de recolección de datos.)

No espere que el primer diagrama E-R que haga sea el final, que revisará con la comunidad usuaria o que entregará a los diseñadores del sistema. Como los diagramas de flujo de datos y todas las demás herramientas de modelado, los diagramas E-R deben revisarse y mejorarse muchas veces; la primera versión típicamente no será más que un borrador, y las versiones subsecuentes se producirán utilizando una serie de reglas de refinamiento que se presentan en esta sección. Algunas de las reglas de refinamiento llevan a la creación de tipos adicionales de objeto, mientras que otras llevarán a la eliminación de objetos y/o relaciones.

12.2.1 Añadir tipos de objetos adicionales

Como se indicó anteriormente, el primer DER típicamente se creará a partir de entrevistas iniciales con el usuario, y de su conocimiento de la materia en cuanto al negocio del usuario. Esto, desde luego, le dará una buena pista respecto a la identidad de los principales objetos y relaciones.[5]

Después de haber desarrollado el primer DER, el siguiente paso es asignar los datos del sistema a los diversos tipos de objetos. Se supone, desde luego, que sabe cuáles son los datos. Esto puede suceder en cualquiera de tres maneras:

1. Si el modelo del proceso (el DFD) ya se ha desarrollado o se está desarrollando paralelamente al modelo de datos, entonces el diccionario de datos ya existirá. Pudiera no estar completo aún, pero lo que haya será suficiente para comenzar el proceso de asignación.

---

5 Sin embargo, probablemente no identificará *todas* las relaciones entre objetos. Dado un sistema con N objetos, ¡el número de relaciones posibles es proporcional a N!, lo cual típicamente es un número *enorme*. Muchas de las relacicnes (teóricamente) posibles resultarán en 1) no tener un significado legítimo dentro del sistema, o 2) no tener relevancia dentro del contexto en el que se está modelando. Se puede uno imaginar, por ejemplo, un sistema en donde la relación primaria entre compradores y vendedores es **VENDER.** Pudiera también resultar que comprador y vendedora estén casados el uno con la otra; o que la vendedora sea hija del comprador; o que el comprador y el vendedor sean condiscípulos en la escuela. Todo esto pudiera ser muy interesante, pero no se va a representar en el modelo a menos que le sea relevante.

2. Si el modelo del proceso no se ha desarrollado (o, en el caso extremo, si no tiene intención de desarrollar uno), entonces pudiera tener que empezar por entrevistar a todos los usuarios apropiados para construir una lista exhaustiva de datos (y sus definiciones). Al hacer esto, puede asignar los datos a los objetos en el diagrama de E-R. (Sin embargo, note que esto es un proceso ascendente que consume tiempo, y que pudiera ocasionar retrasos y frustración.)

3. Si está trabajando con un grupo activo de administración de datos, hay una buena probabilidad de que ya exista un diccionario de datos, que podría obtenerse pronto durante el proyecto, de manera que en ese momento ya pudiera comenzar el proceso de asignación.

El proceso de asignación puede ofrecer una de tres razones para crear nuevos tipos de objetos:

1. Es posible descubrir datos que se pueden asignar a algunas instancias de un tipo de objeto pero no a otras.

2. Pudieran descubrirse datos aplicables a todas las instancias de dos objetos distintos.

3. Podría descubrirse que algunos datos describen relaciones entre otros tipos de objetos.

Si durante el proceso de asignar datos a tipos de objetos encuentra que algunos datos *no se pueden* aplicar a todas las instancias de algún tipo de objeto dado, necesitará crear un conjunto de subtipos abajo del tipo de objeto con el que ha estado trabajando, y asignar los datos específicos a los subtipos apropiados.

Suponga que, por ejemplo, se está desarrollando un sistema de personal, y se ha identificado (con gran perspicacia y creatividad) un tipo de objeto llamado **EMPLEADO.** Al revisar los datos disponibles se encuentra que muchos de ellos **(edad, estatura, fecha de contratación,** etc.) se aplican a todas las instancias de un empleado. Sin embargo, luego se descubre un dato llamado **número-de-embarazos;** se trata obviamente de un dato relevante para las empleadas, pero no para los empleados varones. Esto llevaría a crear **EMPLEADOS-VARONES** y **EMPLEADAS** como subtipos de la categoría general de empleado.

Obviamente, no estoy sugiriendo que todos los sistemas de personal deban hacer seguimiento del número de embarazos que cada empleada ha tenido; el ejemplo se escogió meramente porque existe un consenso general de que los empleados varones *no se pueden* embarazar. Compare esto, sin embargo, con el dato **nombre-del-cónyuge:** hay varias instancias de **EMPLEADO** para quienes no se aplicaría esto (porque son solteros), pero es una situación muy distinta a la de un dato que *no se puede* aplicar definitivamente.[6]

En la mayoría de los casos el proceso de crear nuevos subtipos y asignarles datos de manera apropiada es bastante directo. Sin embargo, debe tenerse siempre en mente una situación excepcional: pudiera suceder que *todos* los datos relevantes se atribuyan a uno de los subtipos, y que *ninguno* de los datos se pueda asignar al objeto supertipo; es decir, puede suceder que los datos sean mutuamente excluyentes, perteneciendo a un subtipo o a otro pero no a ambos. Suponga, por ejemplo, que los únicos datos que se puede asignar a los empleados son **número-de-embarazos** y **años-de-experiencia-jugando-con-el-equipo-Knicks-de-basquetbol.** Podría decidirse (tras preguntarnos a qué lunático usuario pudiera habérsele ocurrido tal sistema) que el supertipo general **EMPLEADO** no se aplica.

También puede ocurrir la situación inversa: los datos pueden describir instancias de dos (o más) tipos distintos de objetos de la misma manera. Si esto ocurre, debe crearse un supertipo nuevo y asignarle los datos comunes al supertipo. Por ejemplo, tal vez se identificó **CLIENTE-DE-CONTADO** y **CLIENTE-A-CREDITO** como dos tipos de objetos distintos cuando se creó el DER para un sistema de pedidos (tal vez porque el usuario señaló que eran dos categorías distintas). Sin embargo, pronto se hace evidente que los datos **nombre-del-cliente** y **domicilio-del-cliente** describen ambos tipos de cliente de la misma forma, lo cual apoyaría la creación de un supertipo, como muestra la figura 12.9.

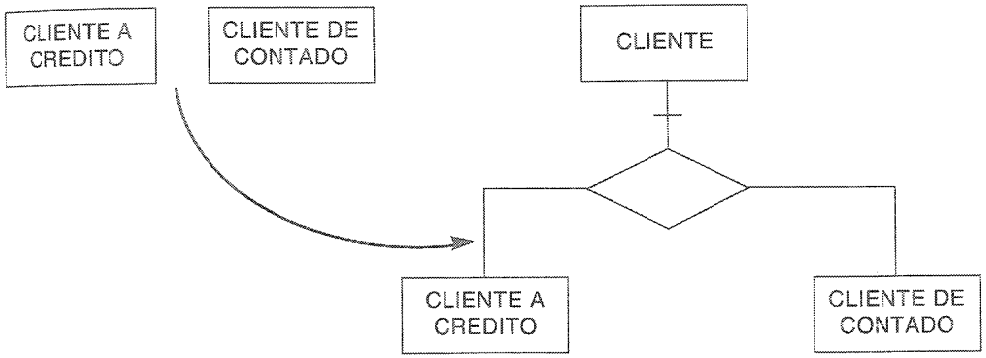


Figura 12.9: **Creación de un nuevo objeto subtipo/supertipo**

6 A lo largo de todo el ejemplo obviamente ignoramos una serie de excepciones obscuras. Ignoramos, por ejemplo, el caso de la empleada que tuvo tres embarazos y luego se hizo una operación de cambio de sexo. Para el caso de los nombres de los cónyuges suponemos que ninguno de los empleados es un niño, porque se supone que les sería imposible estar casados.

Similarmente, si un dato describe la *interacción* de dos o más tipos de objetos, entonces debería reemplazarse la relación "desnuda" entre los dos objetos con un tipo asociativo de objeto. Por ejemplo, en el primer borrador de DER, que se muestra en la figura 12.10(a), existe una relación de **COMPRA** entre **CLIENTE** y **ARTICULO.** Durante la asignación de datos pudiera encontrarse con que hay un dato llamado fecha-de-compra que 1) parece pertenecer a la relación **COMPRA** y 2) obviamente describe, *o proporciona datos acerca de,* la interacción de un **CLIENTE** con un **ARTICULO.** Esto sugiere que debe sustituirse la relación **COMPRA** por un tipo asociativo de objeto, como muestra la figura 12.10(b)

Figura 12.10 (a): **Diagrama E-R inicial**

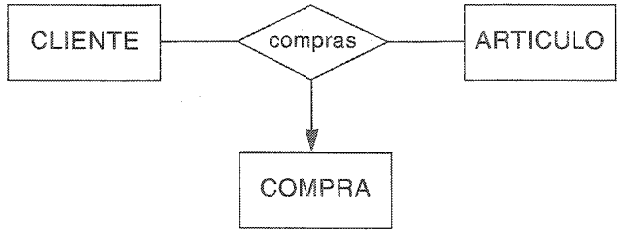


Figura 12.10 (b): **Reemplazo de una relación por un tipo asociativo de objeto**

A veces el diagrama E-R inicial tendrá un tipo de objeto que, visto de cerca, claramente amerita ser un tipo asociativo de objeto. Por ejemplo, la figura 12.11(a) muestra un diagrama E-R con tres objetos relacionados: **CLIENTE, PEDIDO** y **PRODUCTO.** Durante el proceso de asignar datos a los diversos objetos, se encuentra que **fecha-de-entrega** en realidad se aplica al objeto **PEDIDO** porque a los clientes no se les entrega en ningún lado, y los productos se entregan sólo como resultado de un pedido. De hecho, esto hace evidente que **PEDIDO** en sí es una relación entre **CLIENTE** y **PRODUCTO,** además de un objeto acerca del cual interesa recordar algunos hechos. Esto lleva a la figura 12.11(b).

Finalmente, tenemos el caso de grupos que se repiten. Considere, por ejemplo, el tipo de objeto **EMPLEADO,** con los datos obvios como nombre y domicilio. Suponga que hay datos adicionales como **nombre-del-hijo, edad-del-hijo** y **sexo-del-hijo.** Podría argumentarse obviamente que son formas de describir un objeto nuevo llamado **HIJO,** que inadvertidamente se había incluido anteriormente en **EMPLEADO.** Podría también argumentarse que existen (potencialmente) múltiples ins-

tancias de información relacionadas con hijos en cada instancia de un empleado, y que cada instancia de información relacionada con los hijos se define de manera única por el **nombre-del-hijo.** En este caso, el tipo de objeto que inicialmente se imaginó de la forma que muestra la figura 12.12(a) debe transformarse en dos objetos tipo, conectados por una nueva relación, como se muestra en la figura 12.12(b).

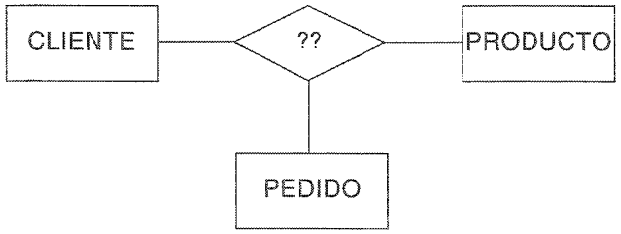


Figura 12.11(a): **DER inicial**

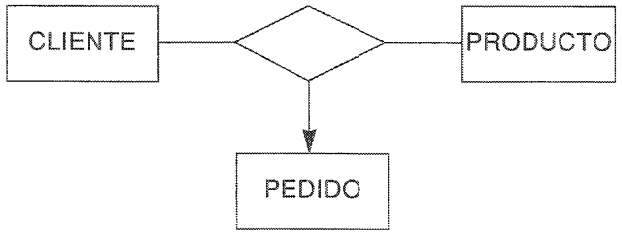


Figura 12.11(b): **Un objeto transformado en objeto asociativo**

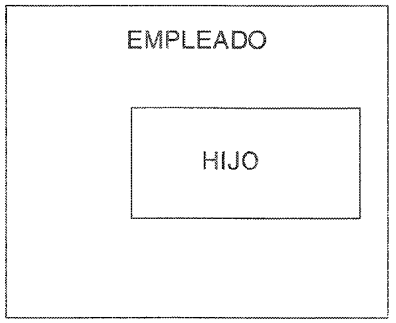


Figura 12.12(a): **Vista inicial de un objeto**

Este proceso de eliminar objetos incluidos en otros es parte de una actividad de refinamiento más general llamada *normalización*. El objetivo de la normalización es producir tipos de objetos, en los que cada instancia (o miembro) consiste en un valor llave primario que identifica a alguna entidad, junto con un conjunto de valores de atributo independientes que describen a la entidad de alguna manera. El proceso de normalización se describe con detalle en el Capítulo 14 de [Date, 1986] y en el capítulo 19 de [DeMarco, 1978].

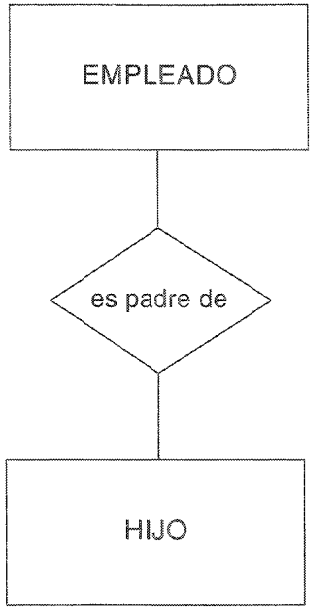


Figura 12.12(b): **Diagrama E-R revisado**

12.2.2 Eliminar tipos de objetos

La sección anterior trató los refinamientos del DER que crean objetos y/o relaciones adicionales. Sin embargo, existe un buen número de situaciones en las que los refinamientos del DER llevan a la eliminación de tipos de objetos y relaciones redundantes o erróneos. Examinaremos cuatro situaciones comunes:

1. Tipos de objetos que consisten sólo en un identificador
2. Tipos de objeto para los cuales existe una sola instancia
3. Tipos asociativos de objetos flotantes
4. Relaciones derivadas

Si se tiene un diagrama E-R en el cual uno de los tipos de objeto tiene sólo un *identificador* asignado como dato, existe la oportunidad de eliminar el tipo de objeto y asignar el identificador, como dato, a un tipo de objeto relacionado. Por ejemplo, imagine que se construyó un DER como muestra la figura 12.13(a) para un sistema de personal:

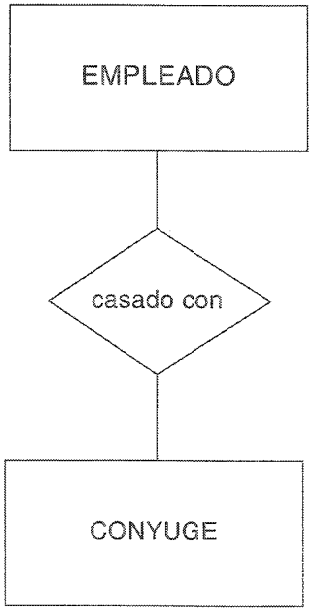


Figura 12.13(a): **Diagrama E-R inicial**

Durante el proceso de asignar datos a los diversos objetos, sin embargo, podría encontrarse que la *única* información que el sistema mantiene acerca del cónyuge es su nombre (es decir, el identificador que distingue a uno de cualquier otro en el sistema). En este caso, un refinamiento obvio sería eliminar **CONYUGE** como tipo de objeto e incluir **nombre-del-cónyuge** como dato dentro del objeto **EMPLEADO.**

Observe que este refinamiento sólo tiene sentido si existe una correspondencia uno a uno entre instancias del objeto que está a punto de ser eliminado e instancias del objeto relacionado. El ejemplo anterior tiene sentido porque la sociedad moderna supone que una persona tendrá cuando más un cónyuge. Esto lleva al diagrama E-R reducido que se muestra en la figura 12.13(b).

Figura 12.13(b): **Diagrama E-R reducido**

Se puede hacer una reducción aún mayor si encontramos que el diagrama E-R inicial contiene un objeto para el cual el único hecho es el identificador, *y* ése es un objeto de una sola instancia. Considere el diagrama E-R inicial que se muestra en la figura 12.14(a).

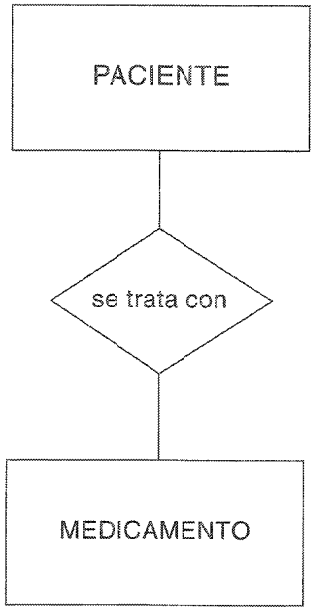


Figura 12.14(a): **Diagrama E-R inicial**

A primera vista parece ser una manera razonable de mostrar la relación entre pacientes y drogas (medicinales, claro) en un hospital. Pero suponga que la única información que se guarda acerca del medicamento es su nombre (identificador); y suponga que el hospital sólo administra un tipo de medicamento (por ejemplo, aspirina). En este caso, el medicamento es una constante y ni siquiera tiene que mostrarse en el diagrama. (Note que esto también significa que el sistema no tendría un almacén de datos llamado medicamentos.) El diagrama reducido se vería como la figura 12.14(b)

Figura 12.14(b): **Diagrama E-R reducido**

Debido a la situación anterior, es posible crear un tipo asociativo de objeto flotante. Considere la siguiente variante del ejemplo del hospital anterior, que muestra la figura 12.15(a). Si, como se sugirió anteriormente, resulta que **MEDICAMENTO** es un objeto de instancia única, sólo con identificador, entonces se eliminaría. Esto resultaría en el diagrama reducido de la figura 12.15(b); nótese que **TRATAMIENTO** todavía es un tipo de objeto asociativo, aunque se conecte sólo con un tipo de objeto. Esto se conoce como tipo de objeto asociativo flotante y es legal (aunque poco usual) en un DER.

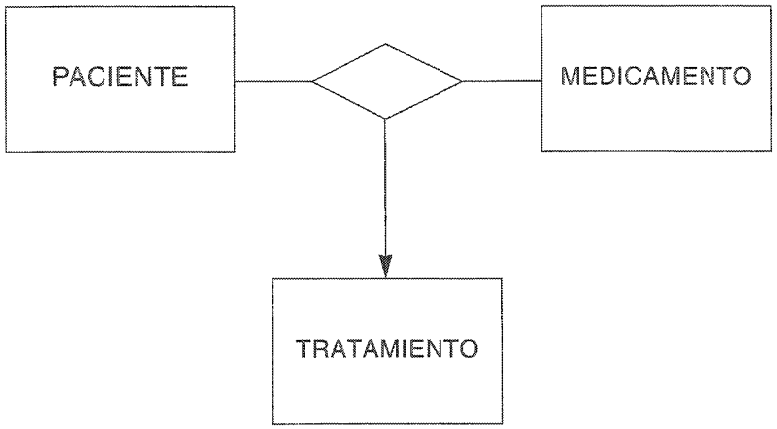


Figura 12.15(a): **Diagrama E-R inicial**

Finalmente, las relaciones que se pueden *derivar*, o calcular, deben eliminarse del diagrama E-R inicial. Como se mencionó anteriormente en este capítulo, el DER debe mostrar los requerimientos para los **datos almacenados.** Por ello, en la figura

12.16(a), si la relación **renovar** entre **CONDUCTOR** y **LICENCIA** se puede derivar (basándose en el cumpleaños del conductor, o en la primera letra de su apellido, o en algún otro esquema usado en la oficina de tránsito), entonces debe eliminarse. Esto lleva a la figura 12.16(b), en la cual los tipos de objeto *no* están conectados. Esto es legal en un DER; no es necesario que todos los tipos de objetos estén conectados entre sí.

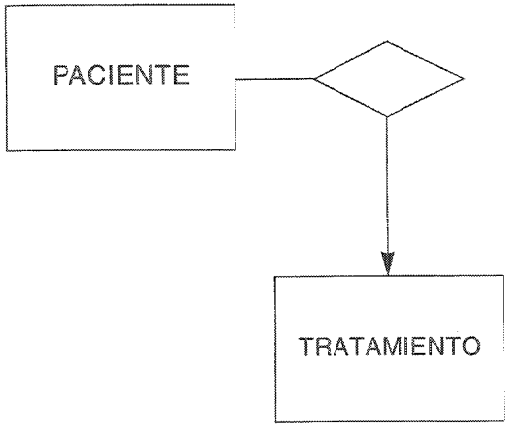


Figura 12.15(b): **Diagrama E-R reducido**

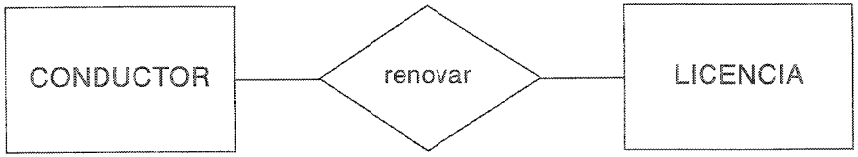


Figura 12.16(a): **DER inicial**

Figura 12.16(b): **DER reducido**

12.3 EXTENSIONES AL DICCIONARIO DE DATOS PARA DIAGRAMAS E-R

Finalmente, observamos que el diccionario de datos que se discutió en el capítulo 10 necesita extenderse para tomar en cuenta la notación de DER discutida en este capítulo. En general, los objetos del DER corresponden con *almacenes* del DFD, lo cual se analizará más a fondo en el capítulo 14. Esto significa que en la definición sacada del diccionario de datos que se da a continuación, **CLIENTE** es tanto definición del tipo de objeto como instancia del almacén **CLIENTES.**

CLIENTES = {CLIENTE}

CLIENTE = @nombre-del-cliente + domicilio + número- telefónico

Nótese también que la definición de un **CLIENTE** incluye la especificación del *campo llave*, que es el dato (o atributo) que diferencia una instancia de un cliente de cualquier otra. El signo de arriba (@) se utiliza para indicar el o los campos llave.[7]

Sin embargo, también hay que incluir en el diccionario de datos una definición de todas las *relaciones* que se muestran en el DER. La definición de relación debe incluir una descripción de su *significado* en el contexto de la aplicación; y debe indicar los objetos que forman la asociación. Los límites superiores e inferiores apropiados deben especificarse para indicar si la asociación es de uno a uno, de uno a muchos o de muchos a muchos. Por ejemplo, la relación **compras** que se muestra en la figura 12.10(a) puede definirse en el diccionario de datos de la siguiente forma:

compras = *la asociación de un cliente y uno o más artículos*
@identidad-del-cliente + 1{@identidad-del-artículo + cantidad-comprada}

12.4 RESUMEN

Para un sistema con múltiples almacenes (objetos) y relaciones complejas entre datos, el DER puede ser una herramienta valiosa. Como se vio en este capítulo, se enfoca totalmente a las relaciones entre datos, sin dar información acerca de las *funciones* que los crean o usan.

A pesar de que en este libro hemos usado el DER como una herramienta gráfica de modelado para mostrar relaciones entre datos, debe saber que se utilizan varias herramientas más para el mismo propósito; [Martin, 1982] y [Date, 1986] muestarn muchas alternativas de herramientas de modelado.

---

7 Algunos textos usan el convenio de subrayar el o los campos clave, por lo cual se puede definir comprador como

COMPRADOR = nombre-del-comprador + domicilio + número-telefónico

Llegando hasta aquí, muchos estudiosos preguntan si debe desarrollarse primero el DFD y luego el DER, o a la inversa. Algunos incluso preguntan si es necesario realmente desarrollar *ambos* modelos, siendo que *cualquiera* de ellos provee tanta información interesante. La respuesta a la primera pregunta es sencilla: no importa qué modelo se desarrolle primero. Uno de los modelos pedirá a gritos ser desarrollado primero según sus propias preferencias, las del usuario, o la naturaleza del sistema mismo (es decir, si es rico en funciones o rico en datos). En otros casos, sin embargo, pudiera encontrarse que ambos modelos se desarrollan paralelamente; esto es particularmente común cuando el equipo del proyecto contiene un grupo bien definido de diseño de bases de datos, o cuando la organización de procesamiento electrónico de datos tiene un grupo administrador que desarrolla modelos de datos corporativos.

La segunda cuestión es más importante: ¿Realmente tiene importancia desarrollar dos modelos distintos de un sistema (y, como veremos en el capítulo 13, un tercer modelo del comportamiento dependiente del tiempo del sistema)? La respuesta es que depende del tipo de sistema que se esté desarrollando. Muchos sistemas clásicos de proceso de datos de negocios desarrollados en los años 60 y 70 parecían (por lo menos superficialmente) consistir en muchas funciones complejas, pero estructuras de datos relativamente triviales, por lo cual el modelo de DFD se enfatizaba y a menudo se ignoraba el de DER. De manera inversa, muchos de los sistemas de apoyo a decisiones y sistemas de bases de datos de investigación ad hoc que se crearon en los años 80 parecían (por lo menos superficialmente) consistir en relaciones complejas entre datos, pero casi nada de actividades funcionales; de aquí que se enfatizara el modelo de DER, y se redujera el de DFD. Las características de tiempo de los sistemas de tiempo real construidos en los años 60 y 70 parecían (por lo menos de manera superficial) dominar sobre cualquier consideración de funciones y relaciones entre datos; en tales sistemas, el modelo de transición de estados (que se discute en el capítulo 13) a menudo se enfatizaba, excluyendo los DFD y DER.

Sin embargo, los sistemas de fines de los años 80 y 90, tienden a ser bastante más complejos que los sistemas de propósito especial de hace una o dos décadas. De hecho, muchos de ellos son entre 100 y 1000 veces más grandes y complejos. Muchos de estos sistemas grandes y complejos tienen funciones, relaciones entre datos y comportamientos dependientes del tiempo increíblemente complejos; considere, por ejemplo, el sistema Guerra de las Galaxias que, según se estima, requiere de *100 millones* de instrucciones de computadora, y que tendrá un comportamiento de tiempo real extraordinariamente complejo. Para un sistema así de elaborado, es obvio que las tres herramientas que se discuten en este libro serán críticamente necesarias. Por otro lado, si Ud. se involucra en un sistema sencillo y unidimensional, puede concentrarse en la herramienta de modelado que enfatiza e ilumina el aspecto más importante de su sistema.

En el capítulo 14 veremos cómo el DER, el DFD, el diagrama de transición de estados, la especificación del proceso y el diccionario de datos pueden compararse entre sí para producir un modelo completo del sistema que sea internamente consistente.

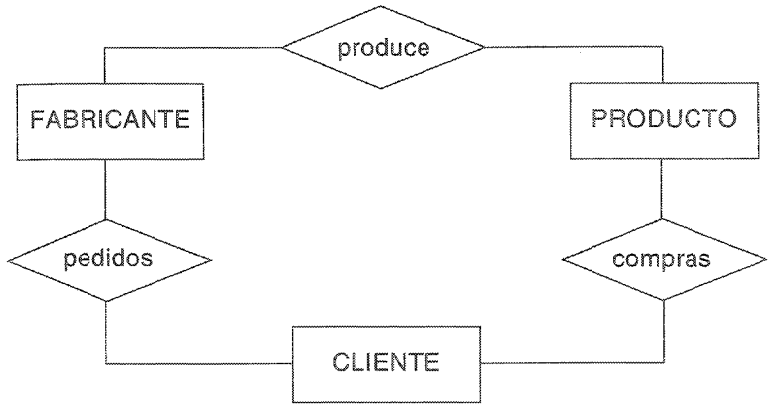
REFERENCIAS

1. Matt Flavin, *Fundamental Concepts of Information Modeling,* Nueva York, YOURDON Press, 1981.
2. Peter Chen, "The Entity-Relationship Model: Toward A Unified View of Data", *ACM Transactions on Database Systems,* Vol. 1, número 1 (marzo de 1976), pp. 9-36.
3. Peter Chen, *The Entity-Relationship Approach to Logical Database Design.* Wellesley, Mass.: Q.E.D. Information Sciences, 1977.
4. D.C. Tsichritzis y F.H. Lochovsky, *Data Models,* Englewood Cliffs, N.J.: Prentice-Hall, 1982.
5. James Martin, *Computer Database Organization,* Englewood Cliffs, N.J.: Prentice-Hall, 1982.
6. *Proceedings of the International Conference on Data Engineering.* Washington, D.C.: IEEE Press, 1984.
7. C.J. Date, *An Introduction to Database Systems,* 4ª edición, Reading, Mass.: Addison-Wesley, 1986.
8. Sally Shlaer y Stephen Mellor, *Object-Oriented Systems Analysis: Modeling the World in Data.* Englewood Cliffs, N.J.: YOURDON Press, 1988.
9. R. Veryard, *Pragmatic Data Analysis.* Oxford, U.K.: Blackwell Scientific Publications, 1984.
10. Jeffrey Ullman, *Principles of Database Systems.* Potomac, Md.: Computer Science Press, 1982.
11. Tom DeMarco, *Structured Analysis and System Specification.* Nueva York, YOURDON Press, 1978.

PREGUNTAS Y EJERCICIOS

1. ¿Qué es un diagrama de entidad-relación? ¿Cuál es su propósito?
2. ¿En qué difiere un DER de un DFD?
3. ¿Por qué se interesa tanto la gente en los modelos de datos?

4. ¿Aparte de los analistas, qué otro grupo dentro de una organización pudiera crear modelos de datos?
5. ¿Por qué se interesa normalmente el grupo DBA en una organización por el modelo de datos?
6. ¿Cuáles son los cuatro principales componentes de un diagrama de entidad-relación?
7. ¿Cuál es la definición de tipo de objeto?
8. ¿Cuál es la diferencia entre un objeto y un tipo de objeto?
9. ¿Cuáles son los tres criterios que debe satisfacer un tipo de objeto?
10. ¿Cuál de los siguientes es probable que sea un tipo de objeto razonable dentro de un sistema de negocios típico? Para aquellos que no considere tipos de objetos razonables, indique por qué:
    (a) "cliente"
    (b) "calcular impuestos de ventas"
    (c) "estatura"
    (d) "producto"
    (e) "jitomate"
    (f) "religión
    (g) "temperatura"
    (h) "editar la transacción"
    (i) "parte manufacturada"
    (j) "mapa"
    (k) "carácter ASCII"
11. ¿Cuál es la definición de relación?
12. ¿Cuántos tipos de objetos pueden conectarse por una relación?
13. ¿Cuáles de las siguientes son relaciones probables en un DER y cuáles no? ¿Por qué sí y por qué no?
    (a) "compras"
    (b) "cliente"
    (c) "pertenece a"
    (d) "peso"
    (e) "produce"
    (f) "cálculo de impuestos de ventas"
14. ¿Cuál es la diferencia entre relación *derivada* y relación *recordada?* ¿Cuál se muestra en un DER?
15. Dé dos ejemplos de una relación derivada entre dos objetos.
16. ¿Cuántas relaciones pueden existir entre dos objetos en un DER?
17. Considere el DER que se muestra.



(a) Escriba una descripción narrativa de objetos y relaciones.

(b) ¿Cuántos pedidos pueden existir en una instancia de **fabricante** y una de **cliente?**

(c) ¿Cuántos **productos** puede comprar un **cliente** en una instancia de la relación **compras?**

18. ¿Muestran cardinalidad los DER?
19. Use la notación de la figura 12.6 para mostrar una versión razonable del diagrama de la figura 12.5.
20. ¿Qué argumentos existen *contra* la ordinalidad y la cardinalidad en un DER?
21. ¿Cuál es la notación alternativa para los DER que muestra tanto la cardinalidad como la ordinalidad?

22. Dibuje un diagrama DER para representar la siguiente situación en una aerolínea:

"La aerolínea XYZ tiene tres recursos principales: aviones, pilotos y miembros de la tripulación. Los pilotos y miembros de la tripulación tienen sus respectivas bases, a las cuales regresan al final de un vuelo. Un vuelo debe tener por lo menos un piloto y uno o más miembros de la tripulación en un avión. Cada avión tiene una base de mantenimiento."

23. Dibuje un DER para describir la siguiente situación de una editorial:

"La editorial ABC trabaja con varios autores diferentes que escriben los libros que publica. Algunos autores han escrito sólo un libro, mientras que otros han escrito varios; además, algunos libros tienen coautoría. ABC también trabaja con múltiples imprentas: sin embargo, un libro dado lo imprime una sola imprenta. Un editor de ABC trabaja con diversos autores al mismo tiempo, editando y produciendo sus libros; es labor del editor dar a la imprenta la copia final lista para la cámara cuando se ha revisado y formado el manuscrito."

24. Dibuje un DER de la siguiente situación para una organización de consultoría de administración:

"Cada representante de ventas trabaja con diversos tipos de clientes y tiene acceso a varios consultores distintos en la organización. Una sesión de consultoría con un cliente puede requerir varios consultores. Durante la sesión, el vendedor no se involucra y los consultores rinden sus informes directamente al cliente."

25. Dibuje un DER de la siguiente situación:

"Un profesor puede impartir varias clases diferentes, siempre que esté calificado para hacerlo. Cada clase debe tener un profesor, pero pueden asistir a ella varios alumnos. Al comienzo de cada semestre, los grupos se asignan a distintos salones, donde se reúnen regularmente."

26. ¿Qué es un tipo asociativo de objeto? ¿Cuál es la diferencia entre un tipo asociativo de objeto y una relación?

27. ¿Qué es un punto de ancla?

28. Dé tres ejemplos de tipo asociativo de objeto.

29. Vea la figura 12.7. Suponga que *no* existen datos sobre la compra que el sistema deba recordar. ¿Cómo cambia esto el diagrama?

30. ¿Qué es un subtipo/supertipo en un DER?

31. Dé tres ejemplos de subtipo/supertipo.

32. ¿Por qué molestarse en tener subtipos/supertipos en un DER? ¿Por qué no se tienen simplemente tipos de objetos "ordinarios"?

33. ¿Qué refinamientos puede esperar hacer el analista después de dibujar el primer borrador del DER?

34. ¿Cuáles son las tres formas probables que el analista usaría para descubrir los datos de un modelo de datos?

35. ¿Qué significa el término asignación en el contexto de este capítulo?

36. ¿Cómo debe el analista proceder con un DER si ya está desarrollado el DFD?

37. ¿Cuáles son las tres razones para crear tipos de objetos adicionales en un DER después de terminar el primer borrador del modelo?

38. ¿Qué debiera hacer el analista si descubre datos que se pueden asignar a algunas instancias de un tipo de objeto pero no a otras?

39. ¿Qué debe hacer el analista si descubre datos que se aplican a todas las instancias de dos tipos de objeto distintos?

40. ¿Qué debe hacer el analista si descubre datos que describen relaciones entre otros tipos de objetos?

41. ¿Qué debe hacer el analista si descubre conjuntos repetidos en un tipo de objeto?

42. Describa el significado de tener conjuntos repetidos en un tipo de objeto. Dé un ejemplo.

43. ¿Cuáles son las cuatro razones comunes para eliminar un tipo de objeto en un borrador de DER?

44. ¿Qué es un tipo asociativo de objeto flotante?

45. ¿Qué debe hacer el analista si descubre un tipo de objeto que consiste sólo en un identificador en un DER?

46. ¿Qué debe hacer el analista si descubre un tipo de objeto para el cual existe una sola instancia en el borrador del DER?

47. ¿Qué debe hacer el analista si descubre una relación derivada en un borrador de DER?

48. ¿Qué extensiones deben hacérsele al diccionario de datos para manejar el DER?

49. ¿Qué significa la notación @ en un diccionario de datos?

13 DIAGRAMAS DE TRANSICION DE ESTADOS

Todo cuerpo permanece en su estado de reposo o de movimiento rectilíneo uniforme mientras no actúe sobre él una fuerza que altere dicho estado.

Sir Isaac Newton,
Philosophiae Naturalis Principia Mathematica,
Leyes del Movimiento, I, 1687

En este capítulo se aprenderá:

1. La notación de diagramas de transición de estados.
2. Cómo dibujar diagramas de transición de estados particionados.
3. Cómo construir un diagrama correcto de transición de estados.
4. La relación que existe entre DTE y otros modelos.

En los capítulos anteriores vimos herramientas de modelado que enfatizan las *funciones* del sistema, así como los datos *almacenados* que el sistema debe recordar. Ahora veremos un tercer tipo de herramienta de modelado, el *diagrama de transición de estados* (también conocido como DTE), que enfatiza el comportamiento dependiente del tiempo del sistema.

Hasta hace poco, los modelos del comportamiento dependiente del tiempo del sistema importaban sólo para una categoría especial de sistemas conocidos como sistemas de tiempo-real. Como ejemplo de estos sistemas (que discutimos brevemente en el capítulo 2) se tiene el control de procesos, sistemas de conmutación telefónica, sistemas de captura de datos de alta velocidad y sistemas de control y mando militares. Algunos de estos sistemas son pasivos, en el sentido de que no buscan controlar el ambiente que los rodea, sino más bien reaccionan a él o capturan datos que le atañen. Muchos sistemas de alta velocidad de captura de datos entran en esta categoría (por ejemplo, un sistema que captura datos científicos de un satélite a alta velocidad). Otros sistemas de tiempo real son más activos, en el sentido de que pretenden mantener el control sobre algún aspecto del ambiente que los rodea. Los sistemas de control de procesos y una variedad de sistemas interconstruidos en otros entran en esta categoría.

Como podrá imaginarse, los sistemas de este tipo manejan fuentes externas de datos de alta velocidad, y deben proporcionar alguna respuesta y datos de salida de manera suficientemente rápida como para manejar el ambiente externo. Una parte importante de la especificación de tales sistemas es la descripción de *qué* sucede *cuando.*

Para los sistemas enfocados a los negocios, esto normalmente no ha sido tan importante. Las entradas pueden llegar al sistema de diferentes fuentes a velocidades relativamente altas, pero habitualmente se pueden detener si el sistema está ocupado haciendo otra cosa. Por ejemplo, un sistema de nómina no tiene que preocuparse por interrupciones y señales de unidades de radar externas. Generalmente los únicos asuntos que involucran tiempos en este tipo de sistemas son especificaciones de tiempo de respuesta, que se incluyen en el modelo de implantación del usuario, que se discutirá en el capítulo 21.

Sin embargo, estamos empezando a ver algunos sistemas grandes y complejos enfocados a los negocios que sí tienen aspectos de comportamiento de tiempo real. Si el sistema maneja entradas de miles de terminales y entradas de alta velocidad provenientes de otros sistemas de cómputo, así como satélites de comunicaciones, entonces pueden surgir asuntos de comportamiento dependiente del tiempo, del tipo que surgen en un sistema clásico de tiempo real. Por ello, aunque no tenga que tratar tales problemas en cada sistema que construya, debiera estar familiarizado con las herramientas de modelado para el comportamiento dependiente del tiempo.

13.1 NOTACION DE LOS DIAGRAMAS DE TRANSICION DE ESTADOS

En la figura 13.1(a) se muestra un DTE típico (aunque es algo más sencillo que los diagramas que se verán más adelante en el capítulo). Este diagrama muestra el comportamiento de una máquina contestadora de teléfono normal.

Los principales componentes del diagrama son estados, y flechas que representan los *cambios de estado.* Existe una variedad de notaciones alternativas para los diagramas de transición de estados; una que es común se muestra en la figura 13.1(b). Aunque es equivalente a la de la figura 13.1(a), tiene la desventaja de parecerse demasiado a un DFD. Para evitar confusiones, usaremos la notación de la figura 13.1(a) en todo este libro.

13.1.1 Estados del sistema

Cada rectángulo representa un estado en el que se puede encontrar el sistema. El *New World Dictionary* de Webster define un "estado" de la siguiente manera:

> Un conjunto de circunstancias o atributos que caracterizan a una persona o cosa en un tiempo dado; forma de ser; condición.

Por tanto, los estados típicos de un sistema pueden ser:

- Esperar a que el usuario dé su contraseña
- Calentar una mezcla de sustancias químicas
- Esperar la siguiente orden
- Acelerar el motor
- Mezclar los ingredientes
- Esperar los datos del instrumento
- Llenar el tanque
- Aguardar en reposo

Nótese que muchos de estos ejemplos implican que el sistema está *esperando* a que algo ocurra, y no se expresan en términos de que la computadora esté *haciendo* algo. Esto se debe a que el diagrama de transición de estados se usa para desarrollar un modelo esencial del sistema,[1] es decir, un modelo de cómo se comportaría el sistema si hubiera tecnología perfecta. Un aspecto de la tecnología perfecta sería que la computadora trabaje de manera infinitamente rápida, de modo que cualquier proceso o cálculo que tenga que hacer, o cualquier acción que deba tomar, se haga en cero momentos. Así, cualquier estado observable en el que el sistema se pueda encontrar sólo puede corresponder a periodos en los que 1) está esperando que algo ocurra en el ambiente externo o, 2) está esperando a que alguna actividad que se esté dando en ese momento en el ambiente (como mezclar, lavar, llenar, acelerar, etc.) cambie a otra.

Esto no significa que nuestros sistemas sean incapaces de ejecutar acciones o que no pretendamos mostrarlas, sino sólo que las acciones, que ocurren instantáneamente en nuestro modelo de tecnología perfecta, no son lo mismo que los estados, que representan condiciones observables en las que el sistema se puede encontrar. Esto es, un estado representa algún comportamiento del sistema que es observable y que perdura durante algún periodo finito.

1 Analizaremos el concepto de modelo esencial con más detalle en el capítulo 17.

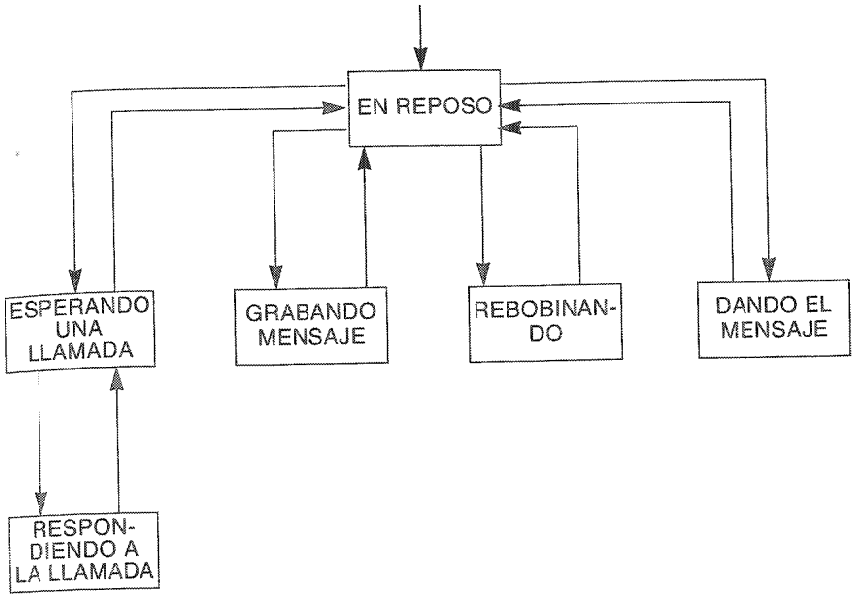


Figura 13.1(a): **Diagrama típico de transición de estados**

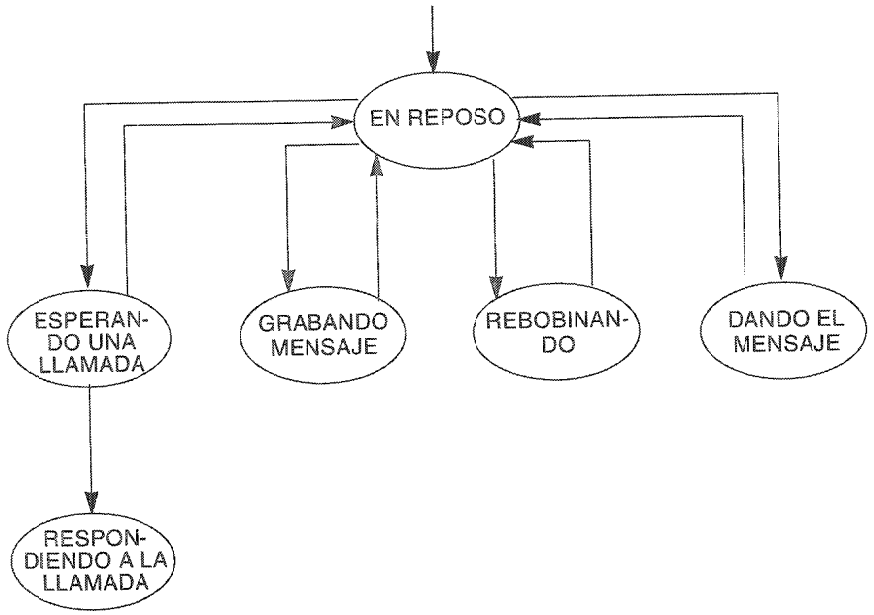


Figura 13.1(a): **Diagrama alternativo de transición de estados**

13.1.2 Cambios de estado

Un sistema que existió en un solo estado no sería un objeto de estudio muy interesante: sería estático. De hecho, los sistemas de información que modelamos normalmente pueden tener docenas de estados diferentes. Pero, ¿cómo cambia un sistema de un estado a otro? Si tiene reglas ordenadas que gobiernan su comportamiento, entonces generalmente sólo algunos tipos de cambio de estado serán significativos y válidos.

Se muestran los cambios de estado válidos en el DTE conectando pares relevantes de estados con una flecha. Así, la figura 13.2 muestra que el sistema puede ir del estado 1 al estado 2. También muestra que cuando el sistema se encuentre en el estado 2 puede ir al estado 3 o regresar al 1. Sin embargo, de acuerdo con este DTE, el sistema no puede ir *directamente* del estado 1 al 3. Por otro lado, el diagrama dice que el sistema *sí puede* ir directamente del estado 3 de regreso al 1. Nótese que el estado 2 tiene dos estados sucesores. Esto es muy común en los DTE; de hecho, cualquier estado puede llevar a un número arbitrario de estados sucesores.

A pesar de que la figura 13.2 proporciona información interesante acerca de comportamiento dependiente del tiempo de un sistema, no dice algo que pudiera resultar ser muy importante: cuáles son los estados *inicial* y *final* del sistema. De hecho, la figura 13.2 es un modelo de estado estable de un sistema que ha estado siempre activo y que continuará siempre estándolo. La mayoría de los sistemas tienen un estado inicial reconocible y un estado final reconocible; esto se muestra en la figura 13.3

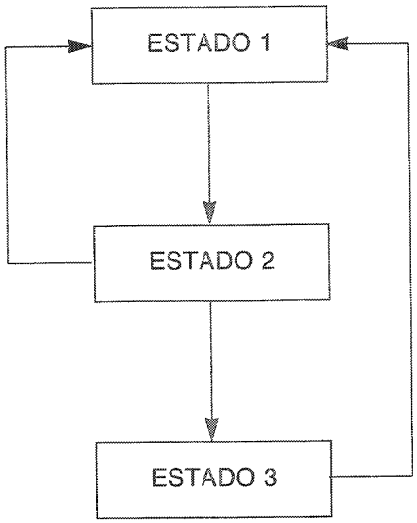


Figura 13.2: **Cambios de estado**

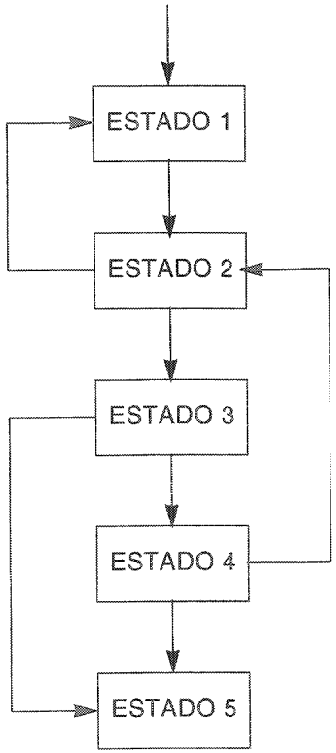


Figura 13.3: **Estados inicial y final**

El estado inicial típicamente suele ser el que se dibuja en la parte superior del diagrama, aunque no es obligatorio. Lo que realmente identifica al estado 1 en la figura 13.3 como inicial es la flecha "desnuda" que no está conectada a ningún otro estado. De manera similar, el estado final suele ser el que se dibuja en la parte inferior, pero tampoco esto es obligatorio. Lo que realmente identifica al estado 5 como final es la ausencia de una flecha que salga de él. En otras palabras, una vez que se llega al estado 5 ya no se puede ir a ninguna otra parte.

El sentido común dice que un sistema sólo puede tener un estado inicial; sin embargo, puede tener múltiples estados finales. Los diversos estados finales son mutuamente excluyentes, lo cual significa que sólo uno de ellos puede ocurrir durante alguna ejecución del sistema. La figura 13.4 muestra un ejemplo en el que los posibles estados finales son el 4 y el 6.

Dado que estamos usando los DTE para construir un modelo esencial, también podemos suponer que los cambios de estado ocurren instantáneamente; es decir, no se requiere tiempo observable para que el sistema cambie de un estado a

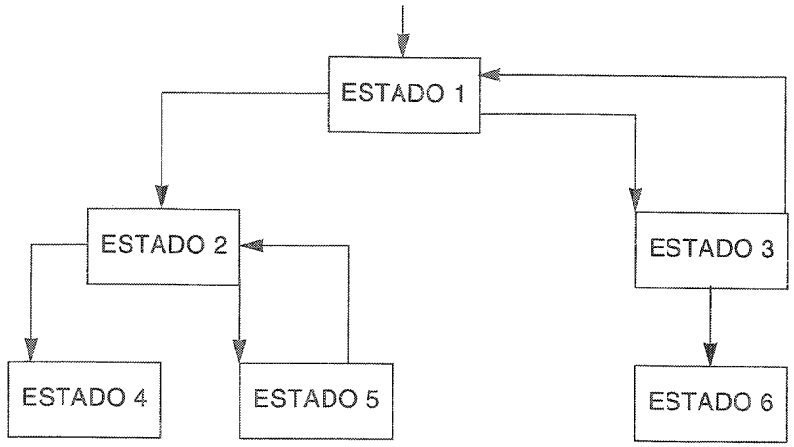


Figura 13.4: **Estados finales múltiples de un sistema**

otro. Cuando los diseñadores y programadores empiezan a construir un *modelo de implantación*, esto será un asunto real: normalmente en una computadora el cambio de una actividad del proceso a otra sí tarda algunos microsegundos, y se debe asegurar que suceda lo suficientemente rápido como para que el ambiente no se salga de control.

13.1.3 Condiciones y acciones

Para completar nuestro DTE necesitamos añadir dos cosas más: las *condiciones* que causan un cambio de estado y las *acciones* que el sistema toma cuando cambia de estado. Como ilustra la figura 13.5, las condiciones y acciones se muestran junto a la flecha que conecta dos estados relacionados.

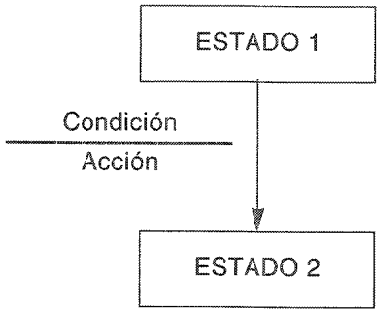


Figura 13.5: **Muestra de condiciones y acciones**

Una condición es un acontecimiento en el ambiente externo que el sistema es capaz de detectar; típicamente es una señal, una interrupción o la llegada de un paquete de datos. Esto usualmente hace que el sistema cambie de un estado de esperar X a un estado de esperar Y; o de llevar a cabo la actividad X a llevar a cabo la actividad Y.

Como parte del cambio de estado, normalmente el sistema hará una o más acciones: producirá una salida, desplegará una señal en la terminal del usuario, llevará a cabo un cálculo, etc. Así que las acciones que se muestran en un DTE son respuestas regresadas al ambiente externo o bien cálculos cuyos resultados el sistema recuerda (típicamente en un almacén de datos que se muestra en el DFD) para poder responder a algún acontecimiento futuro.[2]

13.2 DIAGRAMAS PARTICIONADOS

En un sistema complejo puede haber docenas de estados distintos del sistema; tratar de ponerlos todos en un mismo diagrama sería difícil, si no imposible. Por tanto, tal como se usaron los niveles y las particiones en los DFD, se pueden usar las particiones con los DTE. La figura 13.6(a) muestra un ejemplo de dos niveles de DTE para un sistema complejo.

Nótese que, en este caso, cualquier estado individual de un diagrama de mayor nivel puede convertirse en un estado *inicial* para un diagrama de nivel inferior que describe más a fondo ese estado de mayor nivel; y el o los estados finales en un diagrama de nivel inferior corresponden a las condiciones de salida en el estado asociado de nivel superior. En otros casos, el analista puede requerir mostrar, explícitamente, cómo es que un DTE de menor nivel sale a algún lugar apropiado en el de nivel superior.

Un ejemplo de la necesidad de un DTE por partes puede ser el cajero automático que se encuentra hoy en día en la mayoría de los bancos; un DTE para esto se muestra en la figura 13.6(b).

13.3 CONSTRUCCION DEL DIAGRAMA DE TRANSICION DE ESTADOS

Ahora que se ha visto la notación para los DTE, brevemente discutiremos los pasos a seguir para su construcción. Puede seguirse cualquiera de dos enfoques:

1. Se puede comenzar por identificar todos los posibles estados del sistema y representar cada uno como una caja separada en una hoja de papel. Luego, se pueden explorar todas las conexiones con significado (es decir, los cambios de estado) entre las cajas.

---

2 Observe que para llevar a cabo una acción, el sistema puede requerir entradas adicionales del ambiente externo. Así que puede decirse que cada *condición* corresponde a un acontecimiento externo al cual debe responder el sistema y que usualmente será reconocido por el sistema cuando llegue algún flujo de datos entrante. Sin embargo, no es necesario que cada flujo entrante de datos sea un acontecimiento que corresponda a una condición en el DTE.

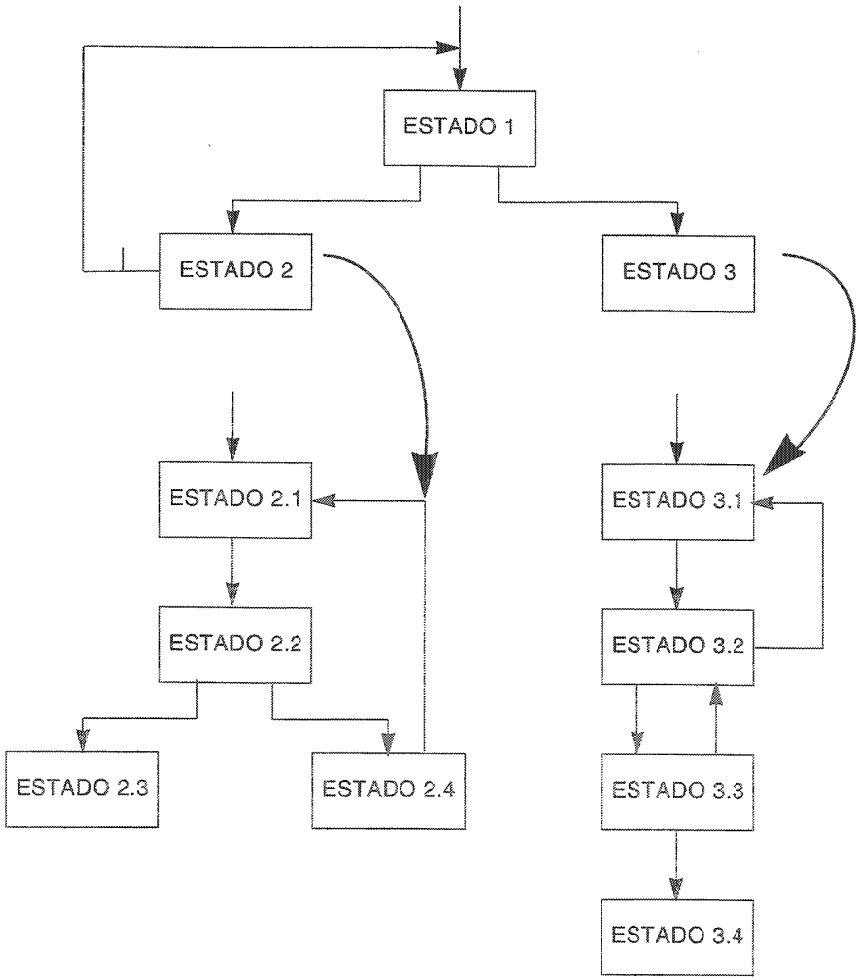


Figura 13.6(a): **Dos niveles de DTE**

2. Como alternativa, se puede comenzar por el estado inicial, y luego metódicamente ir siguiendo un camino hasta el o los estados restantes; luego del o los estados secundarios, proseguir a los terciarios; etc.

El enfoque quedará determinado, en muchos casos, por el usuario con quien esté trabajando, sobre todo si él es el único que está familiarizado con el comportamiento dependiente del tiempo del sistema.

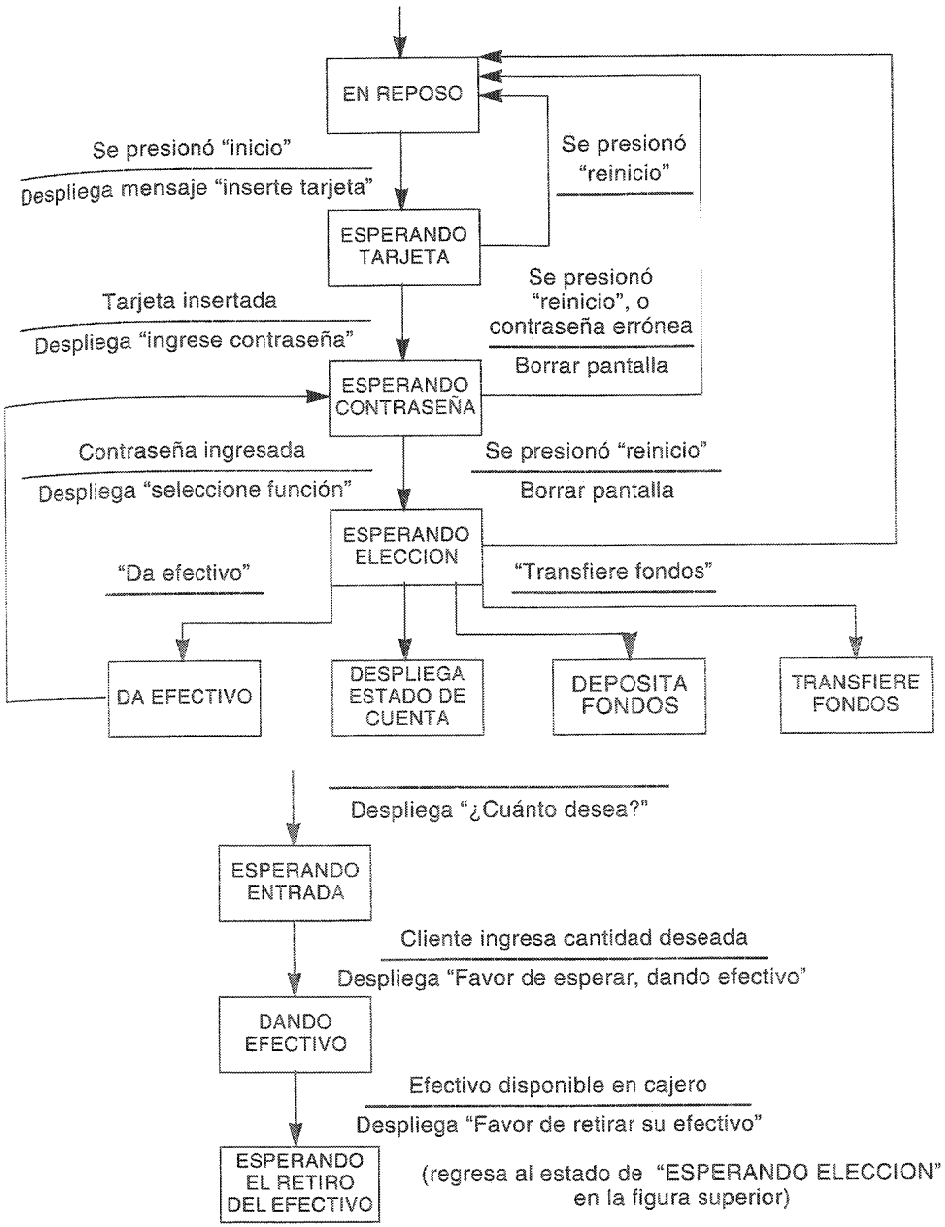


Figura 13.6(b): **DTE particionado para un cajero automático**

Cuando se termine de construir el DTE preliminar, deben seguirse las siguientes reglas para verificar la consistencia:

- *¿Se han definido todos los estados?* Vea con cuidado el sistema para detectar si existe algún otro comportamiento observable, o alguna otra condición en la que el sistema pudiera estar, aparte de las que se han identificado.
- *¿Se pueden alcanzar todos los estados?* ¿Se han definido estados que no tengan caminos que lleven a ellos?
- *¿Se puede salir de todos los estados?* Como se mencionó anteriormente, el sistema puede tener uno o más estados finales con múltiples entradas a ellos; pero todos los demás estados deben tener un sucesor.
- *En cada estado, ¿el sistema responde adecuadamente a todas las condiciones posibles?* Este es el error más común cuando se construye un DTE: el analista identifica los cambios de estado cuando ocurren condiciones normales, pero no especifica el comportamiento del sistema ante condiciones inesperadas. Suponga que el analista modeló el comportamiento de un sistema como el que muestra la figura 13.7; se espera que el usuario presione una tecla de función en su terminal para causar un cambio de un estado 1 a un estado 2, y una tecla *diferente* para ir del estado 2 al 3. Pero ¿qué tal si el usuario presiona la misma tecla dos veces seguidas? ¿O alguna otra tecla? Si no se especifica el comportamiento del sistema, existe una buena posibilidad de que los diseñadores y programadores no lo programen tampoco, y el sistema tendrá un comportamiento impredecible bajo una variedad de circunstancias.

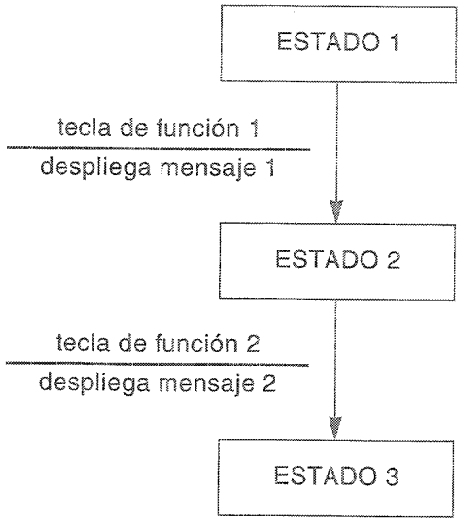


Figura 13.7: **DTE incompleto**

13.4 LA RELACION DEL DTE CON LOS DEMAS COMPONENTES DEL MODELO

El DTE puede usarse por sí solo como herramienta de modelado. Sin embargo, puede, y en general debiera, ser utilizado en conjunto con otras herramientas.

En la mayoría de los casos, el DTE representa una especificación de proceso para una burbuja de control en un DFD. Esto se ilustra en la figura 13.8; note que las *condiciones* en un DTE corresponden a los flujos de control *entrantes* en un DFD, y las acciones en el DTE corresponden a los flujos de control de *salida* en el DFD. Como herramienta de modelado de alto nivel, el DTE puede servir incluso como especificación de proceso para *todo el sistema*. Si se representa todo el sistema con un diagrama de una burbuja,[3] puede usarse el DTE para mostrar la secuencia de actividades en el sistema.

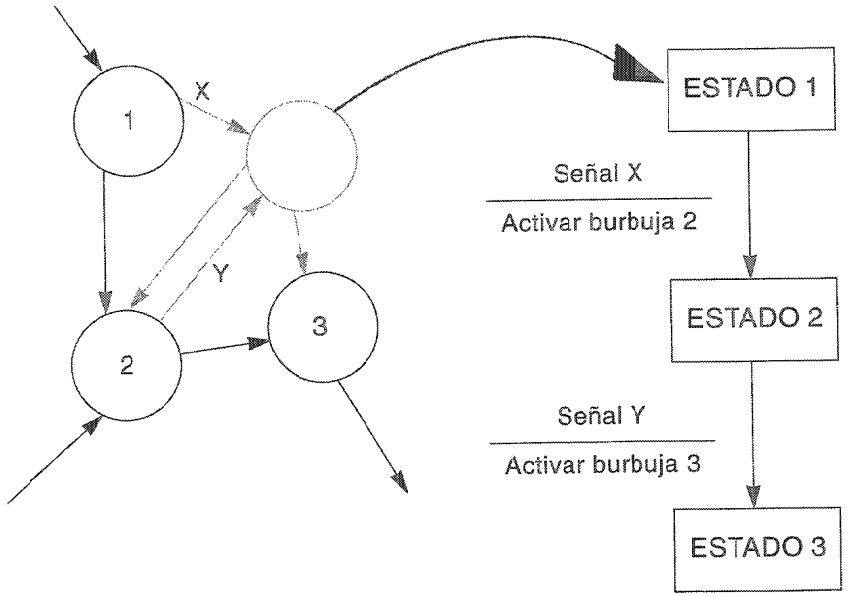


Figura 13.8: **Relación entre un DFD y un DTE**

3 Un diagrama así se conoce como diagrama de contexto. Se analizarán con más detalle en el capítulo 16.

13.5 RESUMEN

Los diagramas de transición de estados son una herramienta poderosa de modelado para describir el comportamiento requerido de los sistemas de tiempo real, al igual que la porción de la interfaz humana que la mayoría de los sistemas en línea tiene. Aunque no son ampliamente conocidos y utilizados en el desarrollo de sistemas de información dirigidos a los negocios, son una herramienta con la que debe familiarizarse, porque en un futuro se espera que cada vez más sistemas, ya sea para negocios o para ciencias e ingeniería, adquieran algunas características de tiempo real.

REFERENCIAS

1. *Webster's New World Dictonary*, Second College Edition, Nueva York: Simon & Schuster, 1980.

PREGUNTAS Y EJERCICIOS

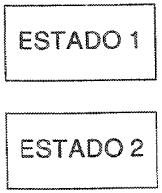
1. ¿Qué es un diagrama de transición de estados? ¿Cuál es su propósito?
2. ¿Qué tipo de sistemas más probablemente emplearán un DTE como herramienta de modelado?
3. ¿Son los DTE herramientas importantes para describir los requerimientos de un sistema de información para negocios típico? ¿Por qué?
4. ¿Son los DTE herramientas importantes para la descripción del diseño/implantación de un sistema de información típico orientado a los negocios? ¿Por qué? Si así es, ¿qué tipo de sistemas?
5. ¿Cuáles son los dos principales componentes de un DTE?
6. Muestre una notación alternativa para un DTE, es decir, una distinta a la estándar que se muestra en este capítulo y en el resto del libro.
7. ¿Cómo se define un estado?
8. Dé tres ejemplos de estado.
9. ¿Qué es un cambio de estado? ¿Cómo se muestra en un DTE?
10. ¿Qué es un estado sucesor?
11. ¿Qué es un estado inicial de un sistema? ¿Cuántos estados iniciales puede tener un sistema?
12. ¿Qué es el estado final de un sistema? ¿Cuántos estados finales pueden existir en un sistema?

13. ¿Qué son las condiciones en un DTE? ¿Cómo se muestran?
14. ¿Qué son las acciones en un DTE? ¿Cómo se muestran?
15. ¿Cuántas condiciones puede haber en un diagrama de transición de estados?
16. ¿Cuántas acciones pueden asociarse con cada condición en un diagrama de transición de estados?
17. ¿Cuáles de los siguientes parecen estados razonables? Para los que no lo parezcan, indique por qué:

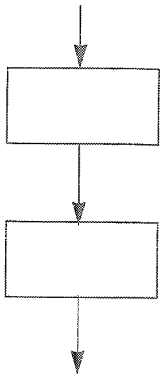
    (a) Calcular impuesto de ventas
    (b) Revisar mezcla reactiva
    (c) Domicilio-para-cobro-al-cliente
    (d) Archivo-de-productos
    (e) Elevador subiendo
    (f) Temperatura de reactivos fuera de rango
    (g) Actualizar total de pedidos
    (h) Detener elevador
    (i) Tecla de interrupción presionada
    (j) Procesando datos

18. ¿Qué es un diagrama de transición de estados por partes?
19. ¿Cuál es la relación entre estados iniciales y estados finales en un DTE por partes?
20. ¿Cuántos niveles puede haber en un DTE por partes?
21. ¿Cuáles son los dos enfoques comunes en la construcción de un DTE?
22. ¿Cuáles son las cuatro reglas para determinar la consistencia de un DTE?
23. ¿Cuál es la relación entre un DTE y un DFD?
24. ¿Qué tiene mal el siguiente DTE?

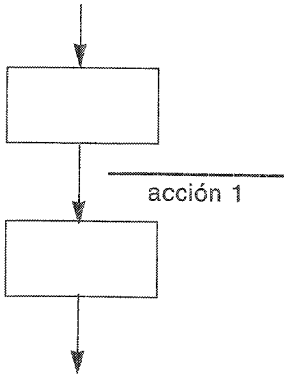
25. ¿Qué tiene mal el siguiente DTE?



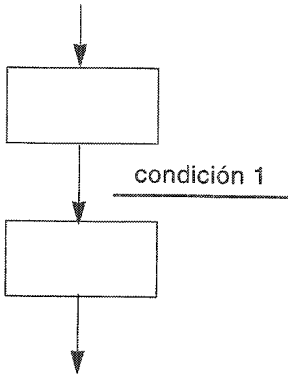
26. ¿Qué tiene mal el siguiente DTE?



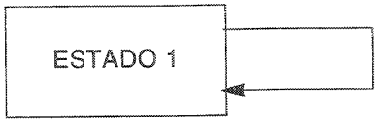
27. ¿Qué tiene mal el siguiente DTE?



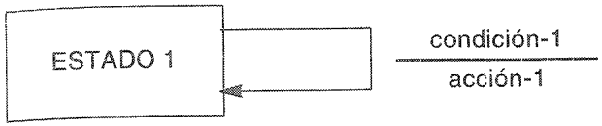
28. ¿Qué tiene mal el siguiente DTE?



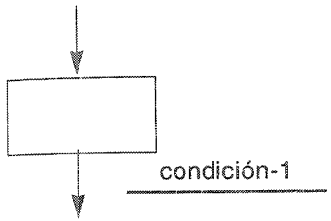
29. ¿Qué tiene mal el siguiente DTE?



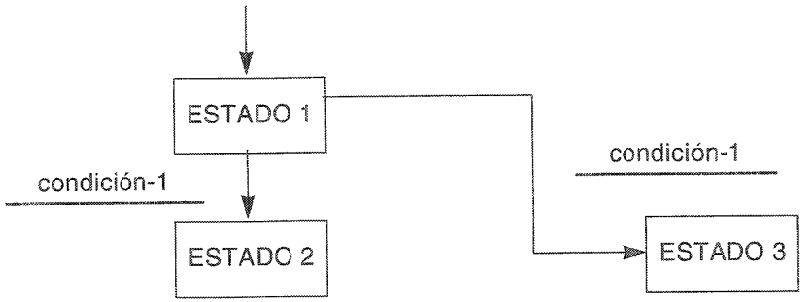
30. ¿Qué tiene mal el siguiente DTE?



31. ¿Qué tiene mal el siguiente DTE? Si no le ve nada mal, describa lo que pudiera estar ocurriendo en el sistema que se está modelando con este DTE.



32. ¿Qué tiene mal el siguiente DTE?



33. En un sistema complejo, ¿debiera empezar el analista por dibujar un conjunto de DFD, comenzar con los DER, o con los DTE?

34. ¿Dónde deben describirse los detalles de las condiciones y acciones del DTE en el modelo del sistema?

35. Dibuje un DTE para una grabadora sencilla o un tocacintas sencillo.

36. Dibuje un DTE para el cajero automático de su banco.

37. Dibuje un DTE para un reloj de pulsera digital (la mayoría de los actuales tienen un modo "normal", además de despertador y cronógrafo).

38. Dibuje un diagrama de transición de estados para un horno de microondas.

39. Dibuje un diagrama de transición de estados para el menú de la interfaz humana de Lotus 1-2-3.

14 BALANCEO DE MODELOS

Todos los hombres corren el riesgo de cometer errores, y la mayoría, por pasión o interés, se sienten con la tentación de hacerlo.

John Locke,
Ensayo sobre el entendimiento humano, 1690

En este capítulo se aprenderá:

1. Cómo balancear el diagrama de flujo de datos y el diccionario de datos.
2. Cómo balancear el diagrama de flujo de datos y la especificación de proceso.
3. Cómo balancear la especificación de proceso y el diccionario de datos.
4. Cómo balancear el DER, el DFD y la especificación de proceso.
5. Cómo balancear el DER y el diccionario de datos.
6. Cómo balancear el diagrama de flujo de datos y el diagrama de transición de estados.

En los últimos cinco capítulos hemos examinado diversas herramientas de modelado para el análisis estructurado:

- Diagrama de flujo de datos
- Diccionario de datos
- Especificación de proceso
- Diagrama de entidad-relación
- Diagrama de transición de estados

Cada una de estas herramientas, como hemos visto, se enfoca en un aspecto crítico del sistema a modelar. Es importante tener esto en mente, pues significa que quien *lee* el modelo también se está enfocando en un aspecto crítico, es decir, el aspecto al cual la herramienta de modelado está atrayendo su atención. Dado que el sistema tiene tantos grados de complejidad, se desea que el diagrama de flujo de datos enfoque la atención del lector en las *funciones* del sistema, sin permitir que las relaciones entre datos distraigan su atención; se desea que el diagrama de entidad-relación enfoque la atención en las relaciones entre *datos,* sin permitir distracción por las características funcionales; y se desea que el diagrama de transición de estados enfoque la atención en las características de *tiempo,* sin la distracción de las funciones o los datos.

Pero llega el momento de reunir todas las herramientas, y de eso trata este capítulo. La situación que enfrenta el modelador del sistema es un tanto análoga a la antigua fábula de los tres sabios ciegos en la India que se tropezaron con un elefante. Como lo ilustra la figura 14.1, llegaron a tres opiniones acerca de la "realidad" con la que estaban tratando, tras tocar distintas partes del elefante:

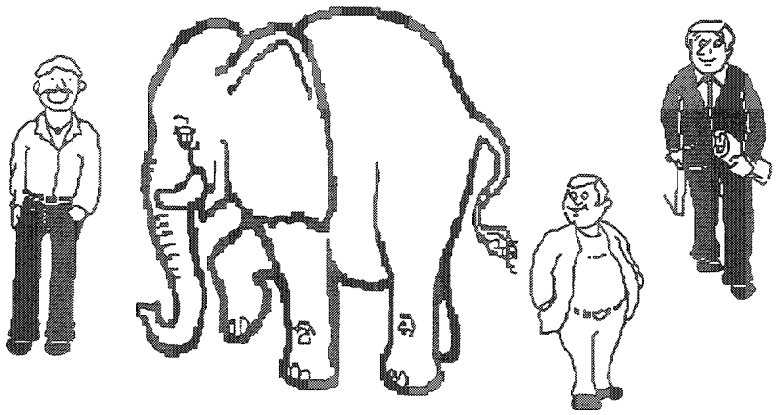


Figura 14.1: **Tres ciegos tocando un elefante**

- Uno tocó la punta de uno de los colmillos del elefante. "Ajá, lo que tenemos aquí es un toro. Siento sus cuernos", dijo.
- El segundo tocó su áspera piel, y dijo: "Sin duda, esto es un...¿qué será? ¿Un puerco espín? Sí, definitivamente, un puerco espín."
- El tercero sintió una de sus gruesas patas y dijo: "Esto debe ser un árbol".

Similarmente, cuando se modelan tres aspectos distintos de un sistema (funciones, datos y tiempos), además de modelar las características detalladas del sistema en un diccionario de datos y un conjunto de especificaciones de proceso, es fácil desarrollar diversas interpretaciones diferentes e *inconsistentes* de esa misma realidad. Este es un peligro particularmente serio cuando se trata de proyectos muy grandes, donde es probable que estén involucrados varios grupos de interés y varias personas. También existe el peligro cuando el equipo que realiza el proyecto (y/o la comunidad usuaria) involucra personas con muy diferente preparación.

Existe otra razón para enfocarse en la consistencia entre modelos: cualesquiera errores que existan tarde o temprano se detectarán, pero se vuelven cada vez más difíciles y caros cuanto más avanza el proyecto. De hecho, es probable que los errores que se puedan introducir en el modelo de requerimientos durante la fase de análisis se propaguen y magnifiquen durante las fases de diseño e implantación del proyecto. Esto se da sobre todo en los proyectos grandes donde el análisis a menudo lo realizan personas diferentes (o incluso distintas compañías) que las que realizan el diseño y la implantación. Martin señala que un 50% de los errores que se detectan en un sistema, y el 75% del costo de su eliminación, se asocian con errores en la fase de análisis. Los estudios de [Boehm, 1981] muestran que el costo de corregir un error aumenta exponencialmente en fases posteriores del proyecto; es diez veces más económico corregir un error del análisis *durante* la fase misma de análisis que durante la fase de diseño.

Algunos de estos errores son, desde luego, errores simples en cada modelo individual (por ejemplo, un diagrama de flujo de datos con un sumidero infinito). Y algunos de los errores se pueden caracterizar como interpretaciones *erróneas* de lo que el usuario realmente quería. Pero muchos de los errores más difíciles e insidiosos son errores entre modelos, es decir, inconsistencias entre un modelo y otro. De una especificación estructurada en la cual todas las herramientas se han verificado entre sí para asegurar su consistencia se dice que está *balanceada.*

El error más común de balanceo involucra alguna definición *faltante:* algo que se define (o describe) en un modelo y no se define apropiadamente en otro. Veremos diversos ejemplos de esto en las siguientes secciones (por ejemplo, un almacén de datos que se muestra en el DFD pero no se define en el diccionario de datos, o un objeto en el DER que no se muestra como almacén de datos en el DFD). El segundo tipo de error común es de *inconsistencia:* la misma "realidad" se describe de dos maneras diferentes y contradictorias en dos modelos diferentes.

Examinaremos varios aspectos importantes del balanceo:

- Balanceo del diagrama de flujo de datos y el diccionario de datos.
- Balanceo del diagrama de flujo de datos y las especificaciones del proceso.
- Balanceo de las especificaciones del proceso y el diccionario de datos.
- Balanceo del DER con el DFD y las especificaciones del proceso.
- Balanceo del DER y el diccionario de datos.
- Balanceo del DFD y el diagrama de transición de estados.

Como veremos, las reglas de balanceo son muy claras; requieren de poca inteligencia o creatividad, pero *deben* seguirse, y diligentemente.

14.1 BALANCEO DEL DFD Y EL DD

Las reglas de balanceo del diagrama de flujo de datos y el diccionario de datos son las siguientes:

- Cada flujo de datos (es decir, cada flecha del DFD) y cada almacén de datos deben estar definidos en el diccionario de datos. Si falta la definición en el diccionario, el flujo o el almacén se considera *indefinido*.
- De manera inversa, cada dato y cada almacén que se define en el diccionario de datos debe aparecer en alguna parte del DFD. Si no aparece, dicho dato o almacén es un "fantasma", es decir, algo definido pero que no se "usa" en el sistema. Esto puede suceder si los datos se definieron para que correspondieran con una versión temprana del DFD; el peligro que se corre es que el DFD pueda cambiarse (es decir, un flujo o un almacén pudiera eliminarse) sin un cambio correspondiente en el diccionario de datos.

Esto significa, desde luego, que el analista debe revisar tanto el DFD como el diccionario cuidadosamente para asegurarse de que estén balanceados. No importa cuál modelo se examine primero, aunque muchos analistas empiezan con el DFD para asegurar que todos los datos estén definidos en el diccionario. Como todas las demás actividades de balanceo que se verán en este capítulo, es una labor tediosa y que se presta para tener un apoyo automatizado.

14.2 BALANCEO DEL DFD Y LA ESPECIFICACION DE PROCESO

He aquí las reglas para el balanceo del DFD y las especificaciones del proceso:

- Cada burbuja del DFD debe asociarse con un DFD de nivel inferior *o* con una especificación de proceso, pero no ambos. Si el DFD muestra una burbuja que se identifica como 1.4.2, entonces debe existir ya sea una *figura* correspondiente identificada como 1.4.2, cuyas burbujas se identifiquen como 1.4.2.1, 1.4.2.2, etc., *o bien* la especificación estructurada debe contener una especificación de proceso para la burbuja 1.4.2. Si *ambas* existen, el modelo es innecesario (y peligrosamente) redundante.
- Cada especificación de proceso debe tener una burbuja de nivel inferior asociada en el DFD. Dado que la especificación de proceso requiere de mucho trabajo, podría pensarse que es altamente improbable que existan especificaciones de proceso "vagabundas" rondando por el sistema. Pero puede suceder: la especificación del proceso pudiera haberse escrito para una versión preliminar del DFD, tras lo cual un proceso de revisión pudo eliminar algunas de las burbujas del DFD.
- Las entradas y salidas deben coincidir. El DFD muestra flujos de entrada y salida para cada burbuja, al igual que las conexiones con los almacenes. Esto debe ser evidente en la especificación de proceso también: así, se puede esperar una declaración READ (o GET, o INPUT o ACCEPT, o algún verbo similar) correspondiente a cada flujo de entrada, y WRITE (o PUT o DISPLAY, etc.) para cada flujo de salida.

Observe que estos comentarios se aplican específicamente a las burbujas de *proceso*. Para las burbujas de *control* en un DFD existe correspondencia entre las burbujas y los diagramas de transición de estados asociados, como se discutió en la sección 14.6

14.3 BALANCEO DE LAS ESPECIFICACIONES DEL PROCESO CON EL DFD Y EL DD

Las reglas para balancear las especificaciones de proceso con el diagrama de flujo de datos y el diccionario de datos son las siguientes: Cada referencia de un dato en la especificación de proceso (típicamente, un sustantivo) debe satisfacer una de las siguientes reglas:

- Coincide con el nombre de un flujo de datos o almacén conectado a la burbuja descrita por la especificación de proceso, *o*
- Es un término local, definido explícitamente en la especificación de proceso, *o*
- Aparece como *componente* en una entrada del diccionario de datos para un flujo o almacén conectado con la burbuja. Así, los datos X y Y aparecen en la especificación de proceso de la figura 14.2, pero no aparecen

como flujo de datos conectado en el DFD que se muestra en la figura 14.3. Sin embargo, el diccionario de datos, del cual se muestra un fragmento en la figura 14.4, indica que X y Y son componentes de Z; y en la figura 14.3 vemos que Z es en efecto un flujo de datos conectado a la burbuja, por lo que concluimos que el modelo está balanceado.[1]

```
ESPECIFICACION DE PROCESO 3.5: CALCULO DEL FACTOR W

* P Y Q SON TERMINOS LOCALES UTILIZADOS PARA OBTENER RESULTADOS INTER-
MEDIOS *

P = 3.14156 * X
.
.
.
Q = 2.78128 * Y - 13
.
.
.
FACTOR-W = P * Q + 2
```

Figura 14.2: **Componente de especificación de proceso de un modelo de sistema**

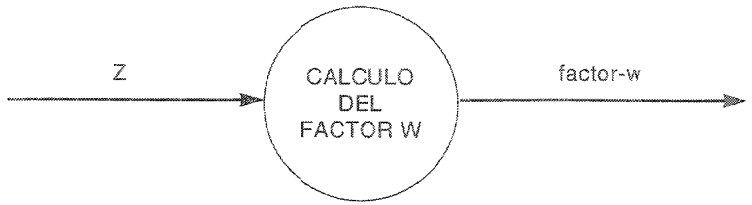


Figura 14.3: **Componente de un DFD de un modelo de sistema**

1 Sin embargo, pudiera valer la pena revisar aún más: si X es el *único* componente de Z que se usa en la especificación del proceso, debe cuestionarse seriamente por qué se mostró Z como entrada desde un principio. Es decir, los demás componentes de Z pudieran ser "datos vagabundos" que deambulan a través de la burbuja sin utilizarse. Esto a menudo refleja el uso de un modelo de *implantación* arbitrario de un sistema, en lugar de un modelo de su comportamiento *esencial*.


```
.
.
X =     * componente horizontal del factor frammis *
        * unidades: centímetros; escala: 0 - 100 *
.
.
.
Y =     * componente vertical del factor frammis *
        * unidades: centímetros; escala: 0 -10 *
.
.
.
Z =     * factor frammis, como lo definió el Dr. Frammis *
        X + Y
```

Figura 14.4: **Componente del diccionario de datos de un modelo de sistema**

14.4 BALANCEO DEL DICCIONARIO DE DATOS CON EL DFD Y LAS ESPECIFICACIONES DEL PROCESO

De la discusión anterior puede verse que el diccionario de datos es consistente con el resto del modelo si obedece la siguiente regla:

- Cada entrada del diccionario de datos debe tener referencia en una especificación de proceso, un DFD, u otro diccionario de datos.

Esto supone, desde luego, que se está modelando el comportamiento *esencial* de un sistema. Un diccionario de datos complejo y exhaustivo de una *implantación* existente de un sistema puede contener algunos datos que ya no se usan.

También se podría argumentar que el diccionario de datos se planee de forma tal que permita una expansión futura; es decir, que contenga elementos que no se ocupen hoy pero que pudieran ser útiles en un futuro. Un buen ejemplo de esto es un diccionario de datos con elementos que puedan ser útiles para consultas *ad hoc*. El equipo del proyecto, tal vez en conjunción con el usuario, debe determinar si este tipo de modelo no balanceado es lo apropiado. Sin embargo, *es* importante por lo menos estar enterado de tales decisiones deliberadas.

14.5 BALANCEO DEL DER CON EL DFD Y LAS ESPECIFICACIONES DE PROCESO

El diagrama de entidad-relación, como vimos en el capítulo 12, presentaba una visión muy distinta del sistema de la del DFD. Sin embargo, existen algunas relaciones que deben darse para que el sistema global sea completo, correcto y consistente:

- Cada almacén del DFD debe corresponder con un tipo de objeto, una relación o una combinación de un tipo de objeto y una relación (es decir, un tipo asociativo de objeto) en el DER. Si en el DFD existe un almacén que no aparece en el DER, algo anda mal; y si hay un objeto o relación en el DER que no aparece en el DFD, algo anda mal.

- Los nombres de objetos en el DER y los nombres de almacenes de datos en el DFD deben coincidir. Como vimos en los capítulos 9 y 12, la convención que sigue este libro es usar la forma *plural* (es decir, **CLIENTES**) en el DFD y la forma *singular* en el DER.

- Las entradas del diccionario de datos deben aplicarse tanto al modelo de DFD como al de DER. Así, la entrada del diccionario de datos para el ejemplo anterior debe incluir definiciones tanto para el objeto del DER como para el almacén del DFD. Esto lleva a una definición de diccionario de datos como la siguiente:

  CLIENTES = {CLIENTE}

  CLIENTE = nombre + domicilio + número-telefónico + ...

Las entradas del diccionario de datos para la forma singular (por ejemplo, **CLIENTE)** deben proporcionar el significado y composición de una sola instancia del *conjunto* de objetos a los que se refiere (en singular) en el DER, y (en plural) en el almacén del DFD. Las entradas del diccionario para la forma plural (por ejemplo, **CLIENTES)** proporcionan significado a la composición de un *conjunto* de instancias.

De manera similar, hay reglas que aseguran que el DER sea consistente con la porción de la especificación de proceso del modelo orientado a las funciones (tenga en mente que las especificaciones de proceso son las componentes detalladas del modelo cuya "encarnación" gráfica es el DFD). Las reglas son que el conjunto combinado de todas las especificaciones de proceso deben, en su totalidad:

- *Crear y eliminar* instancias de cada tipo de objeto y relación que se muestra en el DER. Esto puede entenderse viendo el DFD de la figura 14.5: como se sabe, el almacén **CLIENTES** corresponde al objeto **CLIENTE**. Algo debe ser capaz de crear y eliminar instancias de un *cliente,* lo cual significa que alguna burbuja en el DFD debe tener un flujo de datos conectado con el almacén **CLIENTES.** Pero el *trabajo* mismo de escribir el almacén (es decir, crear o eliminar una instancia del objeto **CLIENTE** relacionado en el DER) debe darse *dentro* de la burbuja, lo cual significa que debe documentarse en la especificación de proceso asociada con ella.[2]

2 Note que la situación puede ser algo complicada: la burbuja que se muestra en el DFD pudiera no ser del nivel inferior. Por ello es posible que la burbuja etiquetada como **INGRESAR-NUEVO-CLIENTE** en la figura 14.5 se describa con un *diagrama de flujo de datos de nivel inferior* y no con una especificación de proceso. De ser éste el caso, una de las burbujas de nivel inferior (posiblemente no sólo de uno, sino de *varios* niveles más abajo) será primitiva y tendrá acceso directo al almacén. Recuerde del capítulo 9 la convención de que el almacén se muestra en el nivel máximo del DFD cuando es interfaz entre dos burbujas, y se repite en cada diagrama de nivel inferior.

- Alguna burbuja de DFD *define valores* para cada dato asignado a cada instancia de cada tipo de objeto, y algún proceso del DFD usa (o lee) valores de cada dato.

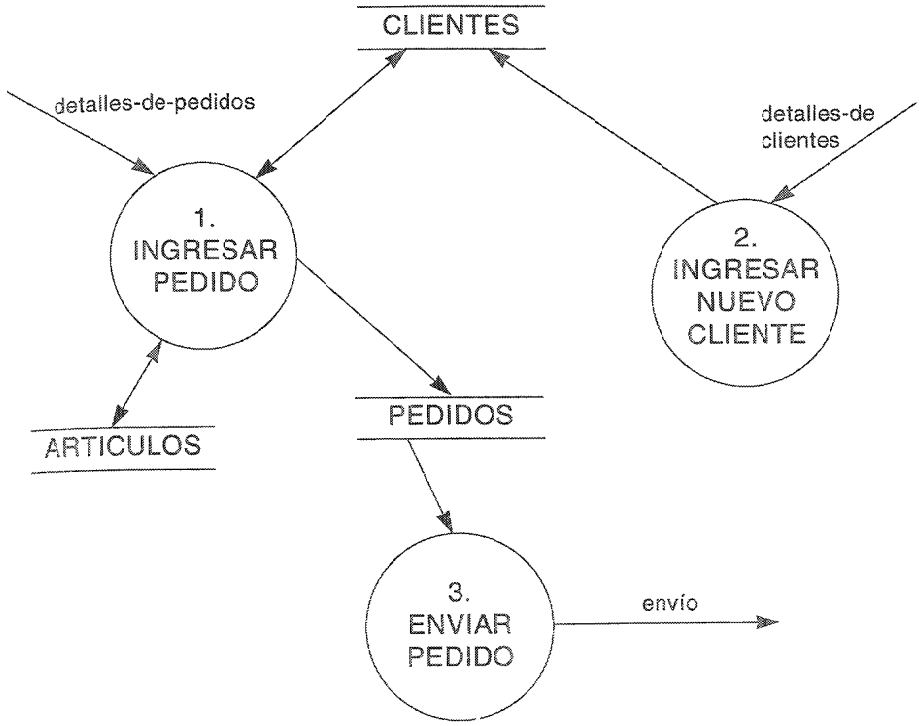


Figura 14.5: **Creación y eliminación de instancias del DER**

14.6 BALANCEO DEL DFD Y EL DIAGRAMA DE TRANSICION DE ESTADOS

La condición de estado puede considerarse balanceada con el diagrama de flujo de datos si se cumple con las siguientes reglas:

- Cada burbuja de control del DFD se asocia con un diagrama de transición de estados como su especificación de proceso. De manera similar, cada diagrama de transición de estados en el modelo global del sistema debe asociarse con un proceso (burbuja) de control en el DFD.

- Cada *condición* del diagrama de transición de estados debe corresponder con un flujo de datos de *entrada* al proceso de control asociado con el diagrama de transición de estados. De manera similar, cada flujo de control que entra en la burbuja de control debe asociarse con una condición apropiada en el diagrama de transición de estados correspondiente.
- Cada acción en el diagrama de transición de estados debe corresponder con un flujo de control de *salida* del proceso de control asociado con dicho diagrama. De manera similar, cada flujo de control de salida de la burbuja de control debe asociarse con una acción apropiada en el diagrama de transición de estados correspondiente.

Estas correspondencias se ilustran en la figura 14.6.

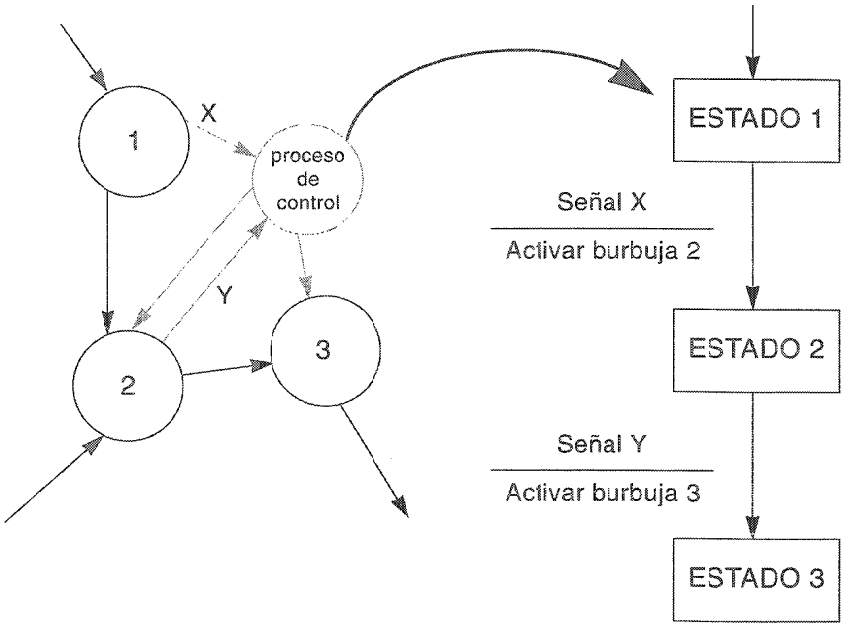


Figura 14.6: **Correspondencias entre el DFD y el DTE**

14.7 RESUMEN

Observe que todas las reglas de balanceo de este capítulo se presentaron como si usted fuera a examinar *personalmente* todos los componentes de un modelo de sistema para detectar errores e inconsistencias en potencia. Esto implicaría que

extendiera en el piso o en un gran pizarrón todos los DFD, las especificaciones de proceso, los ERD, DTE y el diccionario de datos, y luego fuera de uno a otro, revisando cuidadosamente que todo esté en su lugar.

Al estarse preparando este libro, en 1987, eso es *precisamente* lo que se habría hecho en el 98% de las organizaciones de desarrollo de sistemas del mundo. Si tiene la suerte de estar leyendo este libro en los años 90, esto pudiera haber disminuido a un 90%. Y si se está leyendo en 1995 (para cuando el editor ya me deberá haber forzado a producir una nueva edición en la cual se elimine toda esta sección), la cifra puede ser de un 50%. Lo importante es que las reglas de balanceo que hemos presentado en este capítulo pueden automatizarse, y ya existen varias herramientas para estaciones de trabajo basadas en PCs que pueden efectuar mecánicamente parte o toda la revisión de errores.

Pero hemos visto exactamente el mismo fenómeno en una variedad de campos más. Se podría argumentar que la proliferación de sistemas baratos de procesamiento de palabras ya hizo innecesario aprender a escribir a mano; de hecho, se podría argumentar que la disponibilidad de revisores de ortografía incluso hace innecesario el aprender las reglas. Y la disponibilidad universal de las calculadoras de bolsillo ya hace innecesario aprender a dividir. Y la presencia de automóviles de transmisión automática hace innecesario aprender a manejar autos estándar.

De hecho, no se me ocurre ninguna razón fuerte para enseñarle a alguien en Norteamérica a manejar un auto estándar para fines del siglo XX. Ni se me ocurre razón alguna para enfatizar el arte de la caligrafía (excepto, tal vez, como una forma de arte) en una era en la que los sistemas de procesamiento de palabras están a punto de ser reemplazados por sistemas de reconocimiento de voz. Pero sí aprecio la necesidad de aprender los principios básicos de la división, aunque esté uno muy confiado en que nunca le faltará su calculadora de bolsillo; como señala Joshua Schwartz, de la Universidad de Harvard, por lo menos nos ayuda a saber si la respuesta que obtuvimos con la calculadora tiene el punto decimal en la posición correcta.

Se puede discutir incluso los méritos de aprender a escribir a mano en 1987, siendo que (1) menos de la mitad de los niños privilegiados de los Estados Unidos tienen una computadora personal en casa, (2) sólo un 3% de la población global de los E.U.A. tiene una computadora personal en casa; (3) sólo alrededor del 10% de los maestros tienen su propia PC y, (4) sólo un pequeño porcentaje de las escuelas de los E.U.A. están preparadas para enseñar mecanografía. La escritura a mano es *tecnológicamente* obsoleta, y para los padres familiarizados con la computación (sin mencionar a los niños familiarizados con la computación) es penoso verse forzados a aprender esta antigua y primitiva técnica de comunicación; pero probablemente es una técnica todavía necesaria en nuestra sociedad actual. Después de todo, fue sólo hace unos cuantos años que la mayoría de los padres dejaron de enseñarles a sus hijos a reemplazar bujías, cambiar el aceite y cambiar llantas de su automóvil.

De manera similar, estoy convencido de que un analista profesional necesita entender *los principios* del balanceo que se presentan en este capítulo. Como analista, es probable que no tenga más alternativa que seguir mecánicamente estas reglas durante los próximos años hasta que se distribuyan ampliamente las herramientas de ingeniería de software adecuadas. El proceso manual de revisión de errores normalmente se validará en un ambiente de revisión global *(walkthrough)*, que se discute en el apéndice D.

REFERENCIAS

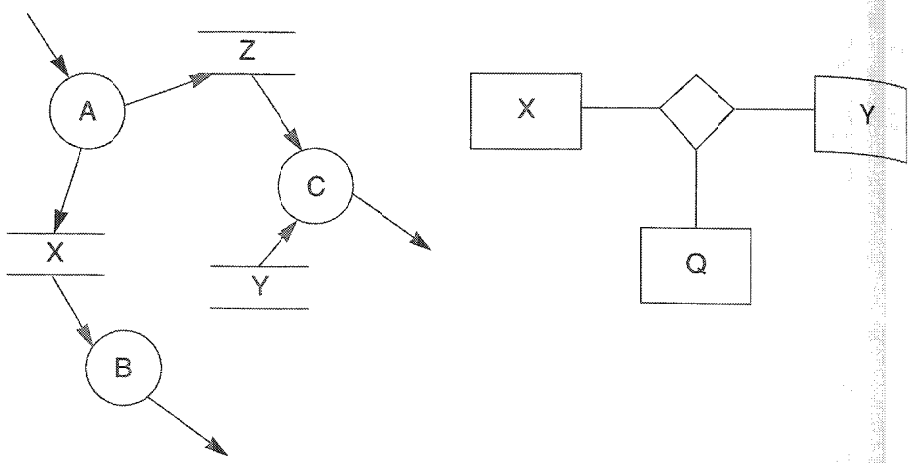
1. James Martin, *An Information Systems Manifesto.* Englewood Cliffs, N.J.: Prentice-Hall, 1984.
2. Barry Boehm, *Software Engineering Economics.* Englewood Cliffs, N.J.: Prentice-Hall, 1981.

PREGUNTAS Y EJERCICIOS

1. ¿Por qué es importante balancear los modelos de una especificación de sistema? ¿Cuáles son los peligros de una especificación no balanceada?
2. ¿Por qué es importante encontrar errores en un modelo de sistema tan pronto como sea posible?
3. ¿Qué porcentaje del costo de la eliminación de errores se asocia con la fase de análisis de un proyecto?
4. ¿Cuáles son los dos errores de balanceo más comunes?
5. ¿Con cuáles partes del modelo del sistema debe balancearse el DFD?
6. ¿Con cuáles partes del modelo del sistema debe balancearse el DER?
7. ¿Con cuáles partes del modelo del sistema debe balancearse el DTE?
8. ¿Con cuáles partes del modelo del sistema debe balancearse el diccionario de datos?
9. ¿Con cuáles partes del modelo del sistema debe balancearse la especificación del proceso?
10. ¿Existen otras componentes del modelo del sistema que deban balancearse?
11. ¿Cuáles son las reglas a seguir para balancear el DFD y el diccionario de datos?
12. ¿Bajo qué condiciones puede definirse un artículo en el DD sin que aparezca en el DFD?

13. ¿Cuáles son las reglas a seguir para balancear el DFD y la especificación de proceso?
14. ¿Qué sucedería si se escribiera una especificación de proceso para una burbuja no primitiva (o no atómica) en el DFD?
15. ¿Debe existir una especificación de proceso para los procesos de control en el DFD? De ser así, ¿deben tener la misma forma de la especificación de proceso para un proceso normal?
16. ¿Cuáles son las reglas a seguir para balancear la especificación del proceso con el DFD y el diccionario de datos?
17. ¿Qué son datos "vagabundos"?
18. ¿Bajo qué condiciones es aceptable que un término (o referencia de dato) en la especificación de proceso no se defina en el diccionario de datos?
19. ¿Cuáles son las reglas a seguir para balancear el diccionario de datos con el DFD y la especificación del proceso?
20. ¿Bajo qué condiciones es posible que el equipo del proyecto *deliberadamente* introduzca elementos en el diccionario de datos que no estén en el DFD?
21. ¿Cuáles son las reglas a seguir para balancear el DER y el DFD?
22. ¿Cuál es la convención que se sigue para hacer coincidir nombres del DER con almacenes en el DFD?
23. ¿Cuáles son las reglas a seguir para balancear el DER y la especificación del proceso?
24. ¿Cuáles son las reglas para balancear el DTE y el DFD?
25. ¿Bajo qué condiciones es válido *no* tener un DTE en un modelo del sistema?
26. ¿Cómo se deben aplicar las reglas de balanceo que se presentan en este capítulo en un proyecto típico de desarrollo de sistemas? ¿Quién debe ser responsable de vigilar que todo se haga?
27. Si tiene una estación de trabajo automatizada de análisis, ¿es necesario aprender las reglas de balanceo presentadas en este capítulo? ¿Por qué?
28. Si se han balanceado los modelos del sistema, ¿se puede confiar en que estén correctos? ¿Por qué?

29. Señale los errores de balanceo del siguiente modelo.



30. ¿Deben balancearse el DTE y el DER? ¿Por qué?

15 HERRAMIENTAS ADICIONALES DE MODELADO

La necesidad de descubrir pronto la ineficiencia vuelve importante *externar* (es decir, hacer visible) un diseño que evoluciona en cada etapa. Los planos de ingeniería, por ejemplo, sirven para eso y le son útiles no nada más al diseñador, a quien señalan los puntos problemáticos e inconsistencias potenciales, sino también al equipo de una organización entera que desarrolla un producto dado. Los planos son el principal medio de comunicación, crítica y refinamiento colectivo. Más aún, los métodos de representación deben ser relativamente sencillos y directos al hacer el puente entre la realidad y el programa, y deben ser útiles durante los múltiples pasos iterativos.

L.A. Belady, prefacio de *Software Design*, [Peters, 1981]

En este capítulo se aprenderá:

1. Cómo identificar diversas variantes de los diagramas de flujo.
2. Cómo dibujar diagramas HIPO y diagramas de estructura.
3. Cómo identificar diversas variantes de los DFD.
4. Cómo identificar diversas variantes de DER.

Las herramientas de modelado que se presentan en los últimos capítulos deben ser suficientes para cualquier proyecto en el que trabaje. Sin embargo, debe también familiarizarse con algunas herramientas adicionales. Aun si no las usa, pu-

diera encontrarlas en su trabajo, y por lo menos debe saber cómo leerlas e interpretarlas.

Las herramientas adicionales que analizaremos en este capítulo incluyen las siguientes:

- Diagramas de flujo y sus variantes
- Diagramas de flujo de sistema
- Diagramas HIPO y diagramas de estructura
- Variantes de los diagramas de flujo de datos
- Variantes de los diagramas de entidad-relación

El propósito de este capítulo no es convertirlo en experto en alguna de estas herramientas, sino simplemente mostrarle que existen como alternativas. Se pueden encontrar detalles adicionales acerca de cada una en las referencias que se mencionan al final del capítulo.

15.1 DIAGRAMAS DE FLUJO Y SUS VARIANTES

15.1.1 El diagrama clásico de flujo

Una de las primeras y mejor conocidas herramientas de modelado es el diagrama clásico de flujo; en la Figura 15.1 se muestra uno típico.

Si ha tenido contacto con computadoras, o con la programación o proceso de datos en cualquiera de sus formas, es probable que ya haya conocido al menos de manera informal los diagramas de flujo. No los discutiremos con detalle en este libro, sino que sólo veremos un subconjunto de la notación. Para conocer más detalles al respecto de la notación de diagramas de flujo, véase [Chapin, 1970].

La notación de la figura 15.1 sólo tiene tres componentes:

- El cuadro representa una instrucción ejecutable o una *secuencia contigua de instrucciones* de la computadora.
- El rombo representa una decisión; en el caso sencillo, representa una decisión *binaria*.
- Las flechas que conectan los cuadros representan el flujo de control. Sólo puede haber una flecha que fluya hacia *afuera* de un rectángulo; es decir cuando la ejecución de una instrucción de computadora concluye, se puede proceder a alguna instruccion o decisión *siguiente* única. De manera similar, puede haber sólo dos flechas que emanen de una decisión.

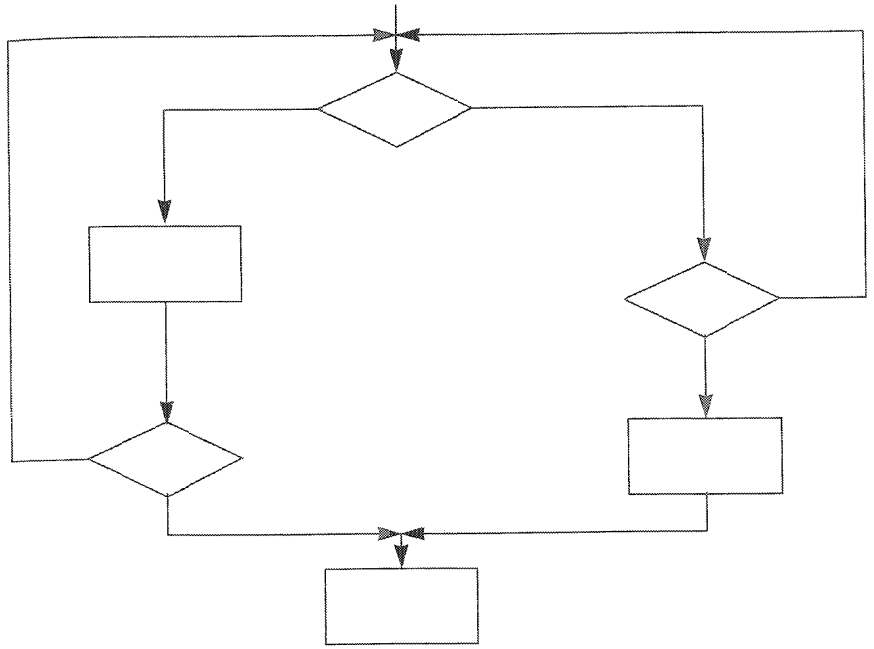


Figura 15.1: **Diagrama típico de flujo**

Como puede verse, el diagrama de flujo permite representar gráficamente la lógica de procedimiento de un programa de computadora. Y es ahí donde los diagramas de flujo se utilizan más, aunque la introducción de los lenguajes de programación de alto nivel en los años 60 y 70 eliminó en gran parte la necesidad de diagramas de flujo detallados.

Pero si son una herramienta de programación, ¿por qué discutirlas en este libro? La respuesta pudiera habérsele ocurrido ya: algunos analistas usan diagramas de flujo como manera de documentar las especificaciones del proceso (es decir, como alternativa del lenguaje estructurado y otras herramientas que se presentaron en el capítulo 11). Como puede ser que recuerde del capítulo 11, siento que *cualquier técnica de documentación que describa de manera precisa la política del usuario y la comunique de manera efectiva es aceptable.* Así, si el usuario disfruta de leer diagramas de flujo y éstos describen de manera precisa la política que realiza una burbuja en un DFD, entonces pueden usarse.

Sin embargo, muy pocos analistas usan realmente diagramas de flujo detallados como especificaciones de proceso. Existen diversas razones para esto:

- A menos que se tenga mucho cuidado, el diagrama de flujo puede volverse increíblemente complicado y difícil de leer.[1] La figura 15.2 muestra un ejemplo de un diagrama de flujo no estructurado.
- Aunque ya se encuentra disponible apoyo automatizado (en estaciones de trabajo basadas en PCs), todavía requiere mucho trabajo desarrollar los graficados de un diagrama de flujo. Y si la política detallada del usuario cambia, o el analista tiene que cambiarla varias veces antes de obtener algo que el usuario acepte como correcto, consumirá mucho tiempo y será tedioso volver a dibujar el diagrama cada vez. Si la especificación del proceso se ha representado en alguna forma textual que pueda manipularse con un procesador de palabras usualmente los cambios serán más fáciles.
- Los modelos gráficos usualmente son la manera más efectiva de ilustrar una realidad *multi*dimensional. Los diagramas de flujo de datos, por ejemplo, ilustran bien el hecho de que todas las burbujas del sistema se pueden activar al mismo tiempo. Pero el flujo de control de un programa o especificación de proceso individual puede describirse en una forma *uni*dimensional; es decir, la lógica puede acomodarse de manera que fluya

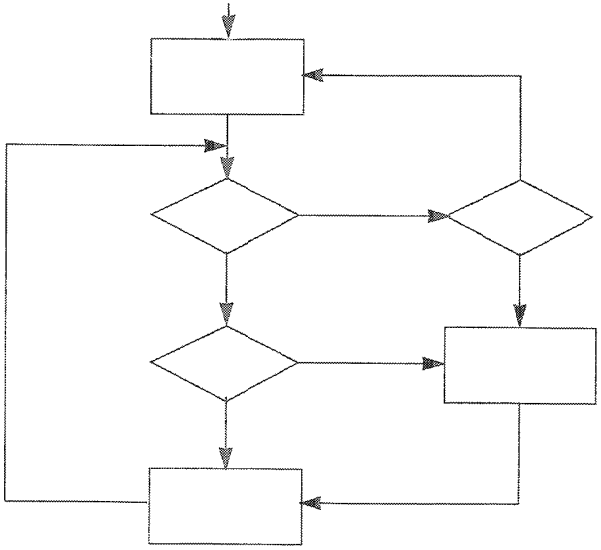


Figura 15.2: **Diagrama de flujo no estructurado**

1 Como consecuencia de esto, una *especificación* desarrollada con diagramas de flujo sería muchísimo más grande que una especificación desarrollada con otras herramientas discutidas en este libro.

uniformemente de manera "descendente".[2] Debido a esto, los graficados son innecesarios.

15.1.2 Variantes de los diagramas de flujo

Aunque los diagramas clásicos de flujo son los más comúnmente usados, si es que se usan, existen variantes que se deben conocer. Mencionaremos cuatro:

1. Diagramas de Nassi-Shneiderman
2. Diagramas de Ferstl
3. Diagramas de Hamilton-Zeldin
4. Diagramas de análisis de problemas

Los *diagramas de Nassi-Shneiderman* (a veces conocidos como diagramas de Chapin) se introdujeron en los años 70 (véase [Nassi y Shneiderman, 1973], y [Chapin, 1974]) como forma de obligar a un enfoque estricto de programación estructurada. Un diagrama de Nassi-Shneiderman típico se muestra en la figura 15.3. Como

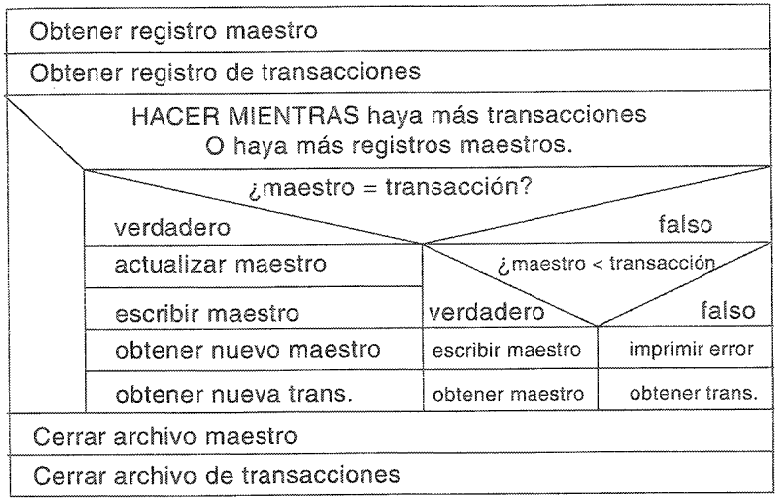


Figura 15.3: **Diagrama típico de Nassi-Shneiderman**

2 El hecho de que cualquier lógica arbitraria de diagramas de flujo pueda reacomodarse en un flujo descendente equivalente es la base de la *programación estructurada*. Böhm y Jacopini[2] probaron que esto se podía hacer en términos de diagramas de flujo por primera vez; en términos de programación, significa que cualquier programa puede escribirse en un lenguaje tipo Pascal sin declaraciones GOTO.

puede verse, el diagrama es fácil de leer. Sin embargo, se podría argumentar que los diagramas de Nassi-Shneiderman son sólo declaraciones del lenguaje estructurado encerradas en cuadros.

Los *diagramas de Ferstl* son otra variante del diagrama clasico de flujo. En [Ferstl, 1978] se proporciona una descripción completa. Un diagrama de Ferstl típico se muestra en la figura 15.4(a). Además de mostrar la lógica normal y secuencial del programa, el diagrama de Ferstl puede usarse para mostrar procesos en paralelo; la notación de Ferstl para procesos paralelos se muestra en la figura 15.4(b).

Los diagramas de Hamilton-Zeldin se produjeron como parte de las actividades de desarrollo de software del proyecto del transbordador espacial de la NASA; véase [Hamilton y Zeldin, 1972]. Un diagrama típico de Hamilton-Zeldin, a veces conocido como diagrama de diseño estructurado, se muestra en la figura 15.5. En los diagramas de Hamilton-Zeldin, los rectángulos tienen el mismo significado que los de un diagrama de flujo ANSI: una declaración ejecutable o un grupo de declaraciones ejecutables contiguas. Un pentágono alargado se usa para mostrar tanto las declaraciones SI como las iteraciones HACER-MIENTRAS/REPITE-HASTA. El control normalmente fluye de arriba hasta abajo del diagrama, excepto en el caso de pruebas SI e iteraciones (HACER y REPITE), que proceden de izquierda a derecha.

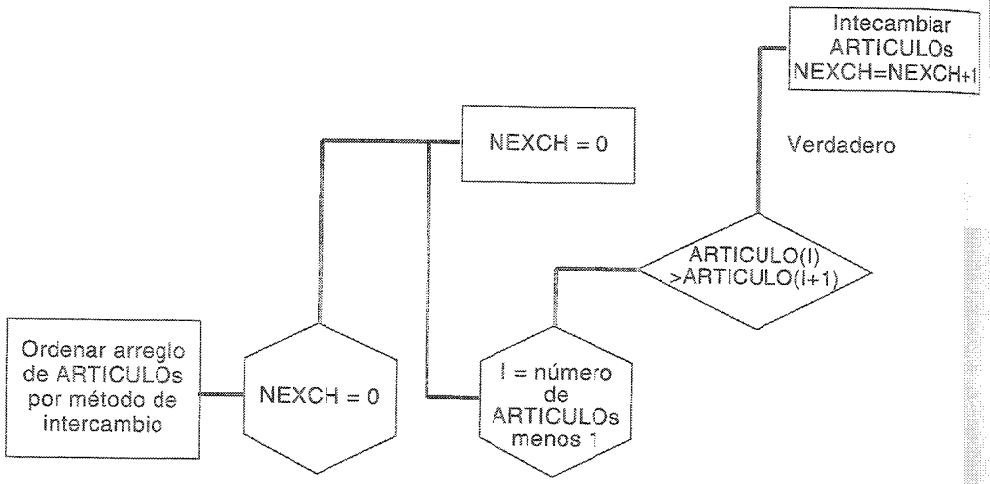


Figura 15.4(a): **Diagrama típico de Ferstl**

Los diagramas de análisis de problemas (PAD, por sus siglas en inglés), que desarrolló la corporación Hitachi (véase [Futamura, Kawai, Horikoshi y Tsutsumi, 1981]), son una representación bidimensional y arborescente de la lógica de programas. Los componentes de un PAD se muestran en la figura 15.6(a). Como los diagramas de Hamilton-Zeldin, los PAD se leen de arriba abajo, y las construcciones SI se muestran de izquierda a derecha; hay un ejemplo en la figura 15.6(b).

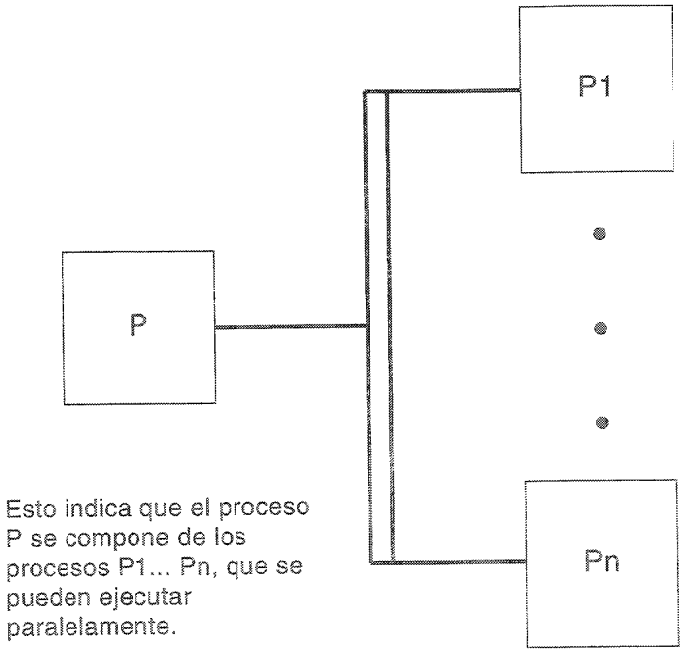


Figura 15.4(b): **Notación de procesos en paralelo en diagramas de Ferstl**

Como los diagramas de Ferstl, un PAD puede mostrar procesos en paralelo; también usa una combinación de despliegue vertical de flujo secuencial con despliegue horizontal de niveles de anidamiento (por ejemplo, ciclos dentro de ciclos) que es similar a los diagramas de Hamilton-Zeldin.

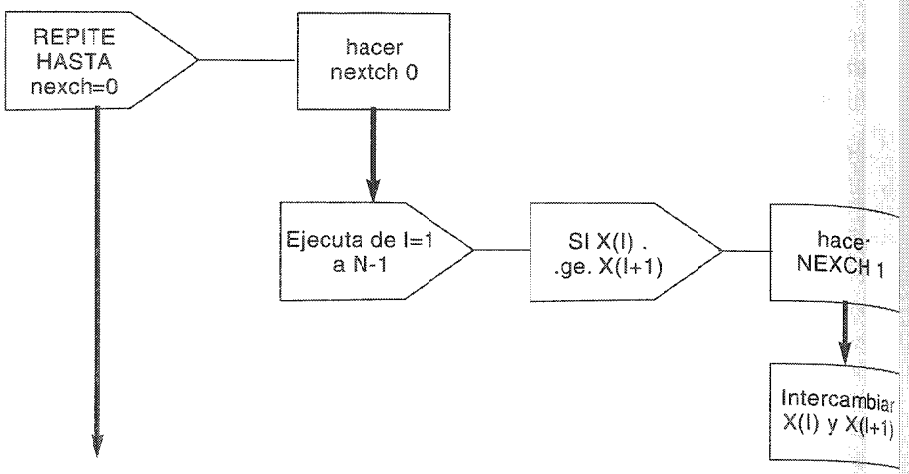


Figura 15.5: **Diagrama típico de Hamilton-Zeidin**

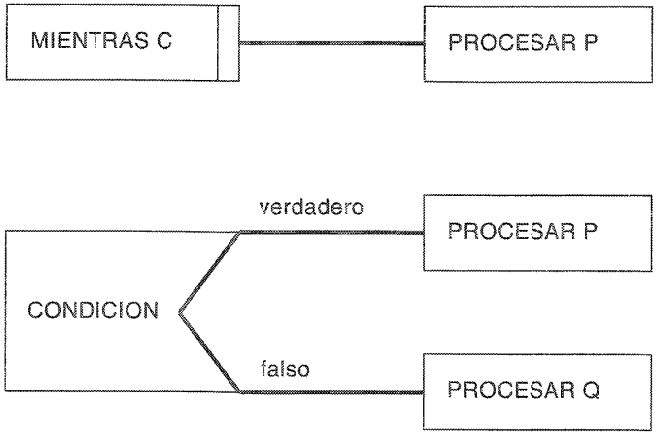


Figura 15.6(a): **Componentes de un PAD**

15.2 DIAGRAMAS DE FLUJO DE SISTEMA

Los diversos enfoques de diagramas de flujo de la sección anterior son útiles para mostrar la lógica detallada, dentro de un programa o dentro de una especificación de proceso para una burbuja individual en un DFD. También se puede mostrar una visión de mayor nivel de la organización de un sistema por medio de otro tipo de

diagrama de flujo: el *diagrama de flujo del sistema*. Un diagrama de flujo típico de sistema se muestra en la figura 15.7.

Observe que los rectángulos representan *agregados* operativos de software de computadora (por ejemplo, programas, pasos de la tarea, ejecuciones u otras unidades de software). El diagrama de flujo del sistema también muestra varios tipos de archivos físicos (por ejemplo, archivos de cinta magnética o de disco). Y puede mostrar la presencia de terminales en línea o enlaces de telecomunicaciones.

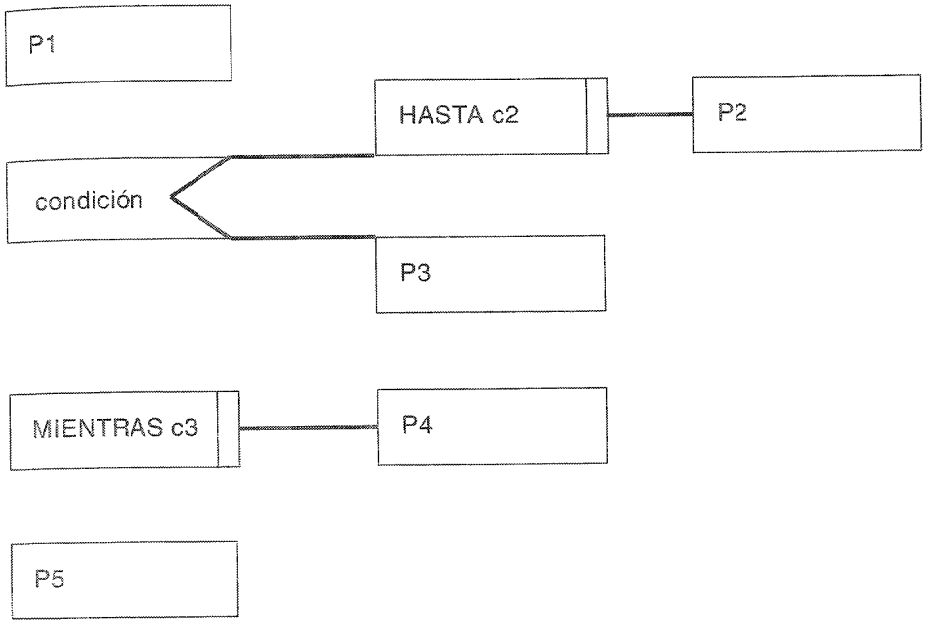


Figura 15.6(b): **PAD típico**

El diagrama de flujo del sistema es a menudo muy útil para los diseñadores que deben desarrollar una arquitectura global del hardware y software del sistema para implantar los requerimientos del usuario. Sin embargo, siento que no es una herramienta de modelado apropiada para el análisis de sistemas, por la sencilla razón de que enfatiza los detalles de implantación *física* que no debieran estar discutiendo el analista y el usuario. Por ejemplo, en lugar de hablar acerca de un archivo en disco debieran estar discutiendo su *contenido;* en lugar de hablar acerca de los programas individuales, deberían discutir las *funciones* a realizar.

Existe una situación en la que el diagrama de flujo del sistema pudiera ser útil para modelar: al final de la actividad de análisis, cuando se está desarrollando el *modelo de implantación del usuario*. En este momento, el usuario, el analista y el equipo de implantación (diseñadores y programadores) discuten las limitaciones de implantación que tendrá el sistema; se incluyen cosas como la determinación de la frontera de automatización (qué partes del sistema serán automatizadas y cuáles serán manuales) y la interfaz humana (detalles de la interacción entre el sistema y sus usuarios humanos).

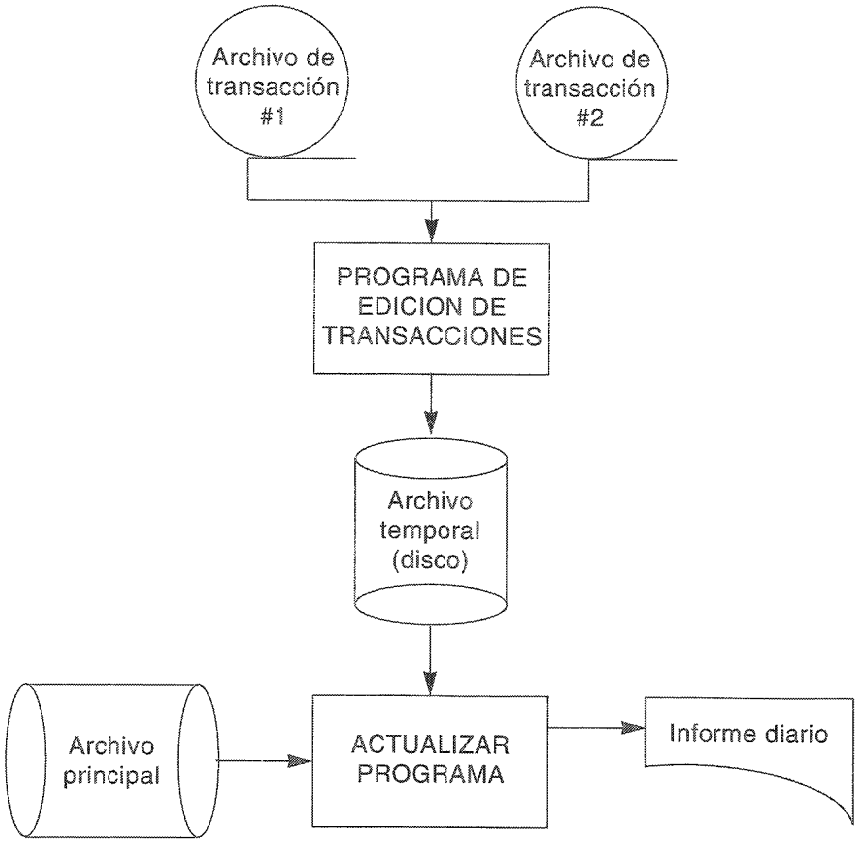


Figura 15.7: **Diagrama de flujo del sistema típico**

15.3 DIAGRAMAS HIPO

IBM desarrolló los diagramas HIPO en los años 70 (véase [HIPO, 1974] y [Katzan, 1976]) y algunos analistas los han usado para presentar una visión de alto nivel de las funciones que realiza el sistema, al igual que la descomposición de funciones en subfunciones, etc. Un diagrama HIPO típico se muestra en la figura 15.8.

En algunos medios de usuarios, los diagramas HIPO pueden ser herramientas de modelado útiles porque se parecen al diagrama de organización ya familiar que describe la jerarquía de gerentes, subgerentes, etc. Sin embargo, el diagrama no muestra los datos utilizados o producidos por el sistema; aunque es comprensible que se quisiera desenfatizar relaciones entre datos en un modelo, no siento que eliminar toda la información sobre los datos sea útil.

En realidad, existe un segundo componente del diagrama HIPO que sí muestra los datos. El diagrama que se muestra en la figura 15.8 es un VTOC, o tabla visual de contenidos. Cada función que se representa por medio de un rectángulo puede describirse con mayor detalle en un diagrama IPO (siglas en inglés de entrada-proceso-salida); un diagrama IPO típico se muestra en la figura 15.9.

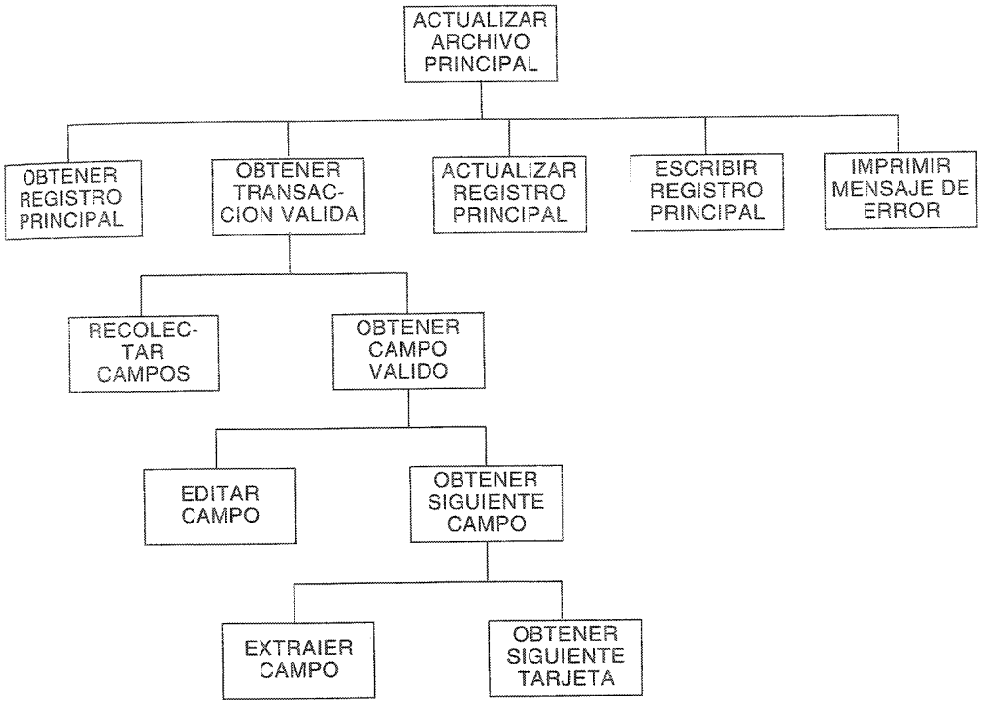


Figura 15.8: **Diagrama típico HIPO**

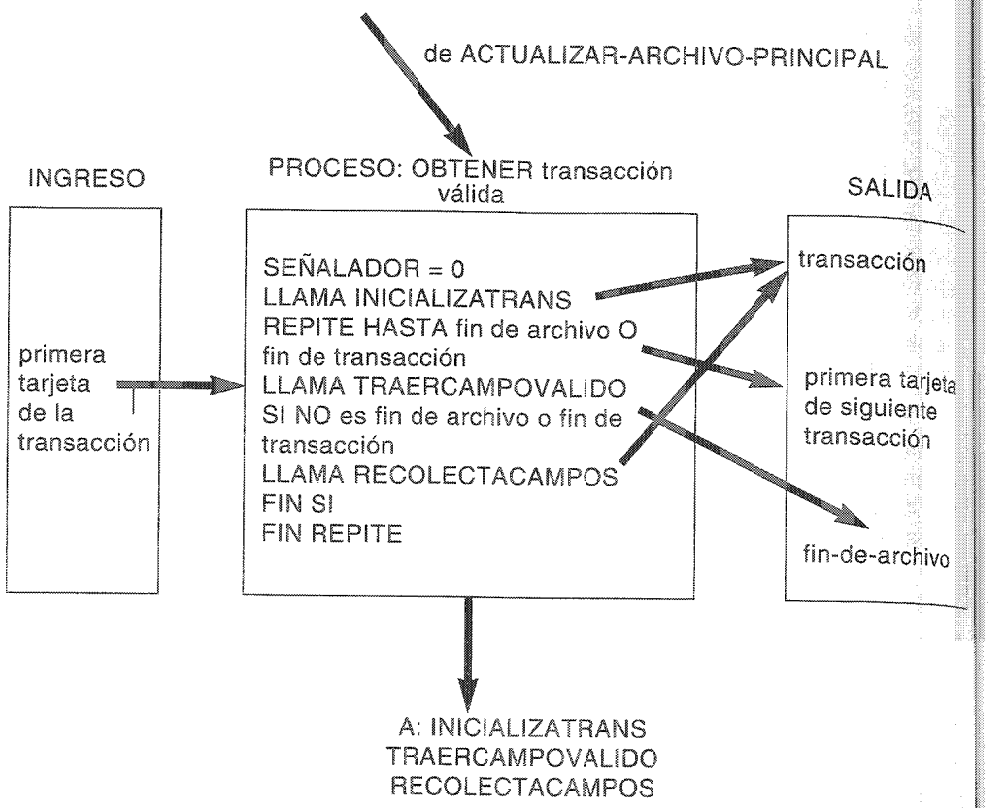


Figura 15.9: **Un diagrama IPO típico**

Aunque los detalles de los datos se muestran de hecho en este nivel, no aparecen en el diagrama VTOC de alto nivel. Así, no es fácil que alguien que vea un panorama global del sistema detecte las interfaces entre sus diversos componentes.

15.4 DIAGRAMAS DE ESTRUCTURA

Una variante de los diagramas HIPO que se utiliza ampliamente son los *diagramas de estructura* . Uno típico aparece en la figura 15.10; observe que además de mostrar la jerarquía funcional muestra las interfaces de *datos* entre los componentes.

A diferencia de la mayor parte de los diagramas anteriores, el rectángulo en un diagrama de estructura no representa una declaración computacional ni un grupo contiguo de declaraciones, sino que representa un *módulo.* (Ejemplos comunes de módulos son las subrutinas de FORTRAN, los procedimientos de Pascal, los subprogramas y las SECTIONes de COBOL). Las flechas que conectan los módulos no representan declaraciones GOTO sino llamados de subrutinas; la notación implica que una subrutina terminará o regresará a donde se llamó cuando finalice de realizar su función.

Aunque el diagrama de estructura generalmente se prefiere frente al diagrama HIPO, todavía no tiene un uso verdadero en el área del análisis de sistemas. ¿Por qué? Porque se utiliza como herramienta de diseño para modelar una *jerarquía sincronizada de módulos* en un sistema; por sincronizada entendemos que sólo un módulo se ejecuta en un tiempo dado, lo cual es una representación precisa de la manera en la que la mayor parte de las computadoras comunes trabajan hoy en día. Por otro lado, el analista necesita una herramienta que le permita mostrar una *jerarquía de redes asincrónicas de procesos;* esto se logra de manera efectiva con un conjunto de DFD por niveles.

El diagrama de estructura se utiliza extensamente en el diseño de programas; se dicutirá con más detalle en el capítulo 22.

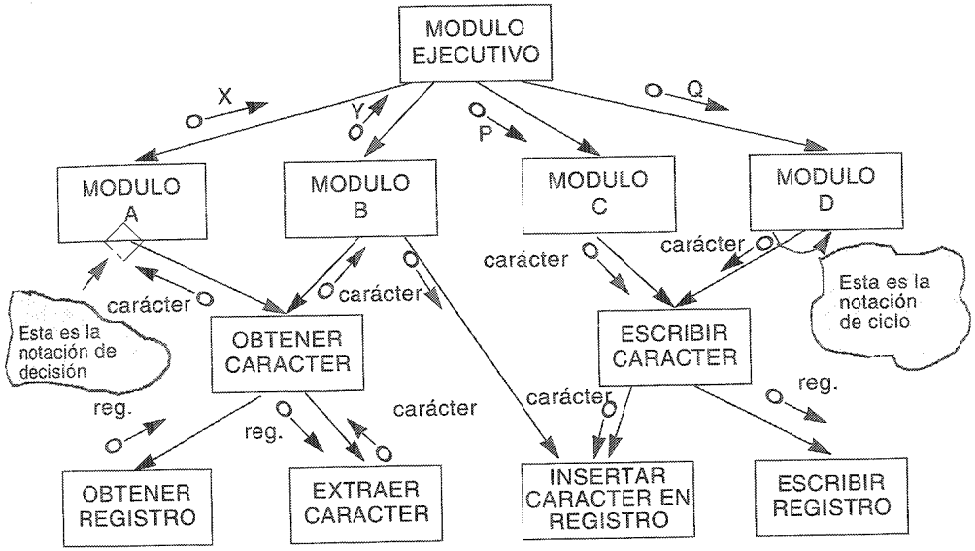


Figura 15.10: **Diagrama típico de estructura**

15.5 VARIANTES DE LOS DIAGRAMAS DE FLUJO DE DATOS

Como se mencionó en el capítulo 9, hay varias diferencias "cosméticas" entre los diagramas de flujo de datos de este libro y los que se muestran en otros. Las diferencias primarias usualmente involucran cosas tales como el uso de un rectángulo

o un óvalo en lugar de una burbuja para mostrar las funciones que realiza el sistema; los diagramas de flujo de datos dibujados con óvalos usualmente se conocen como diagramas de Gane-Sarson.

Sin embargo, existe por lo menos una variante significante del diagrama de flujo de datos clásico; se conoce como diagrama SADT y se desarrolló en Softech (véase [Ross y Schoman, 1977]). La figura 15.11 muestra un diagrama SADT típico.

Aunque similares en naturaleza a los diagramas de flujo de datos que se presentan en este libro, los diagramas SADT distinguen entre *flujos de datos* y *de control* por la manera en la que se colocan las flechas en los rectángulos. Aunque esto sí se puede realizar, impone limitaciones topológicas en el diagrama, que muchos analistas encuentran incómodas.

15.6 VARIANTES DE LOS DIAGRAMAS DE ENTIDAD-RELACION

Los diagramas de entidad-relación que se presentaron en el capítulo 12 son considerados por la mayoría de los analistas como la forma más general y abstracta de representar relaciones entre datos. Sin embargo, existen por lo menos otras tres notaciones populares de estructuras de datos:

- El diagrama de Bachman
- Los diagramas de estructura de datos de DeMarco
- Los diagramas de estructura de datos de Jackson

Una de las formas más comunes de modelo de datos es el diagrama de Bachman, que primeramente desarrolló Charles Bachman en los años 60. La figura 15.12 muestra un diagrama de Bachman típico. Observe que es similar al diagrama de entidad-relación que se discutió en el capítulo 12, pero no muestra explícitamente la

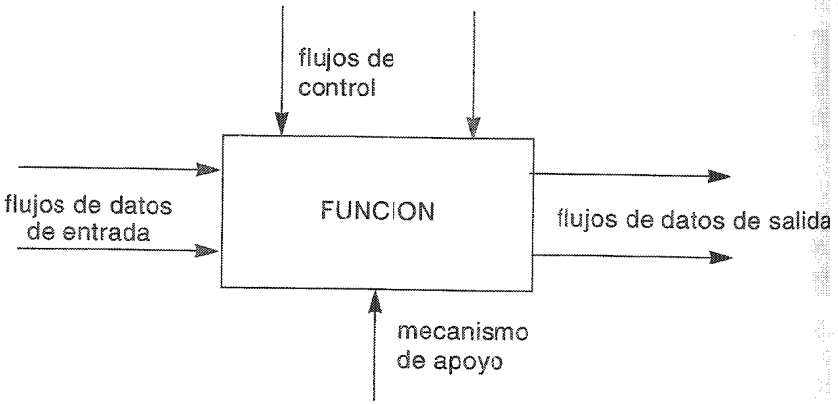


Figura 15.11: **Diagrama SADT típico**

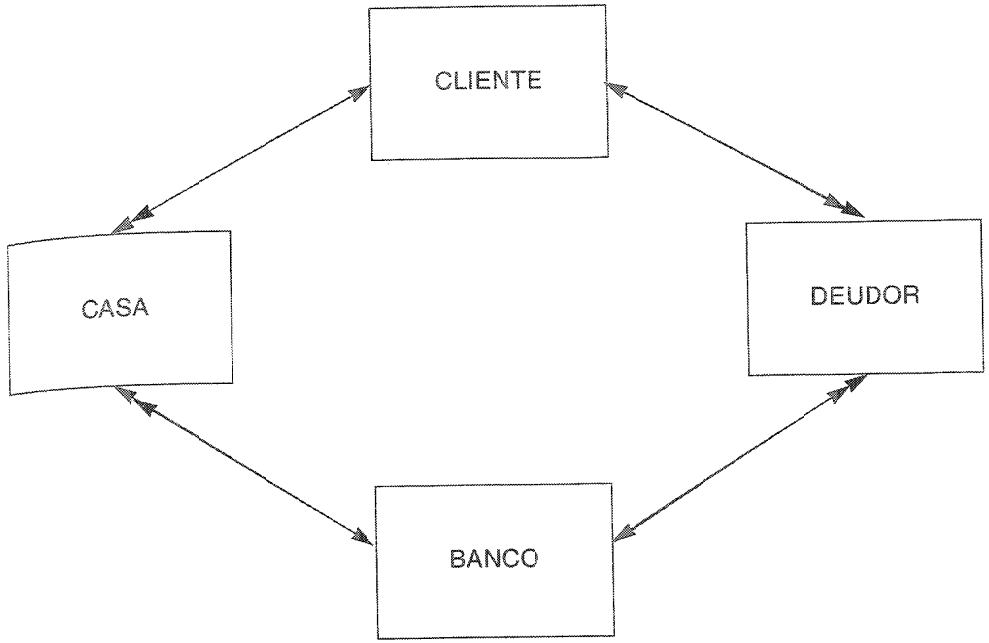


Figura 15.12: **Diagrama típico de Bachman**

relación entre objetos. Note también la flecha de dos cabezas: indica una relación de uno-a-muchos (por ejemplo, un cliente puede poseer más de una casa, pero (en este modelo) una casa sólo puede tener un dueño).

Los diagramas de estructura de datos de DeMarco han logrado bastante popularidad durante los últimos diez años; un diagrama típico de estructura de datos se muestra en la figura 15.13. Observe que además de mostrar cada objeto del modelo de datos, se muestra el *campo llave*; como se recordará, la convención que este libro usa es mostrar el *campo llave* en el diccionario de datos.

Aunque los diagramas de estructura de datos de Jackson no se utilizan mucho en los E.U. por ahora, son bastante populares en Inglaterra, Europa y otras partes del mundo; Jean-Dominique Warnier, [Warnier, 1976] y Ken Orr [Orr, 1977] desarrollaron modelos de datos similares, que son un tanto más populares en los E.U. En lugar de enfocarse en la relación entre distintos objetos en un sistema, los diagramas de Jackson ofrecen medios gráficos para mostrar la estructura jerárquica de un solo objeto. Los componentes de un diagrama de Jackson se muestran en la figura

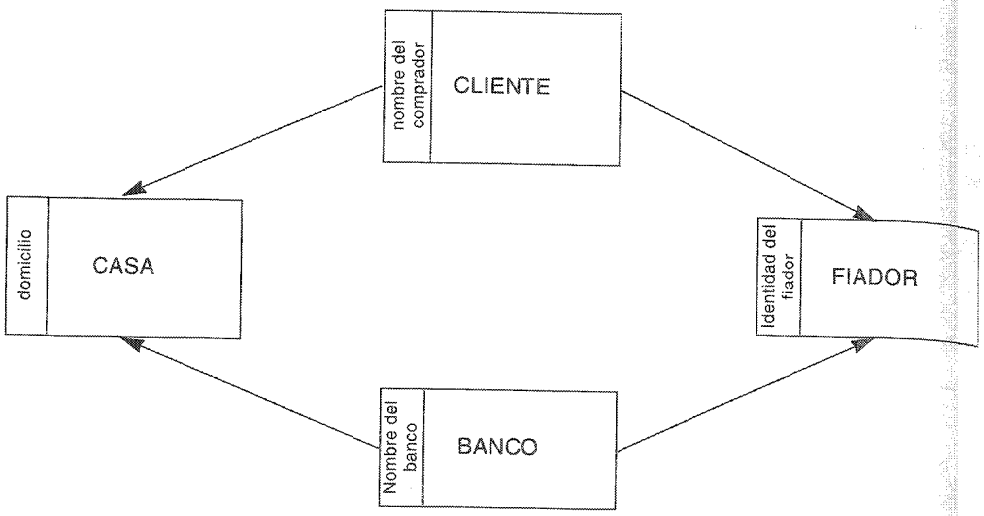


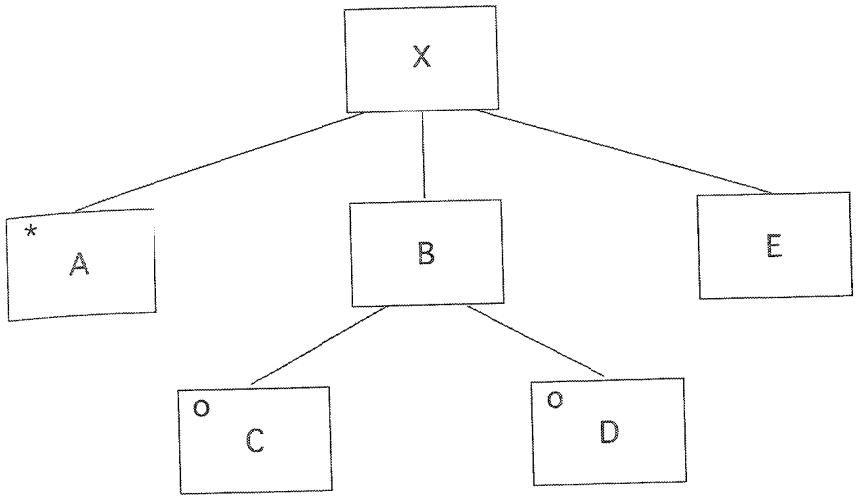
Figura 15.13: **Diagrama típico de estructura de datos de DeMarco**

15.14(a); note que esta misma estructura jerárquica puede documentarse directamente en el diccionario de datos usando la notación que se presenta en el capítulo 11, como muestra la figura 15.14(b).

15.7 RESUMEN

La lista de herramientas de modelado mostrada en este capítulo no está completa, ni se han discutido con detalle estas herramientas alternativas; para conocer más se requiere consultar las referencias que se mencionan al final del capítulo. Sin embargo, tenga en mente que pudiera nunca ver alguno de estos diagramas en un proyecto real (con la probable excepción del siempre presente diagrama de flujo); Por ello, le recomiendo que no se familiarice mucho con los diagramas de Hamilton-Zeldin, los PAD, los diagramas de Ferstl, etc., a menos de que se encuentre trabajando en un proyecto que los requiera.

Tenga presente que pudiera enfrentarse a técnicas de construcción de diagramas totalmente específicas y particulares que no se discuten en este libro (y posiblemente en ninguno). Esto no debe preocuparle: no hay nada particularmente sagrado acerca de las herramientas usadas aquí. Sin embargo, existe una diferencia entre herramientas de modelado *buenas* y *malas;* si se encuentra ante nuevas técnicas, vuelva a leer el capítulo 8 para identificar los criterios que distinguen a las herramientas buenas.



El asterisco, "", se utiliza para mostrar iteración, y la "O" para mostrar "uno u otro"*

Asi, este modelo indica que el dato (u objeto) X consiste en 0 o más ocurrencias de A, seguidas de una B, seguido por una E. El dato B consiste en una C o una D.

Figura 15.14(a): **Diagrama típico de estructura de datos de Jackson**

X = {A} + B + E

B = [C | D]

Figura 15.14(b): **Notación de diccionario de datos correspondiente a la estructura de datos de Jackson de la figura 15.14 (a)**

REFERENCIAS

1. Ned Chapin, "Flowcharting with the ANSI Standard: A Tutorial", *ACM Computing Surveys,* volumen 2, número 2 (junio de 1970), pp. 119-146.

2. Corrado Böhm y Guiseppe Jacopini, "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules," *Communications of the ACM,* volumen 9, número 5 (mayo 1966), pp. 336-371. También publicado en *Classics in Software Engineering*, E. Yourdon (editor). Nueva York: YOURDON Press, 1979.

3. I. Nassi y B. Shneiderman, "Flowchart Techniques for Structured Programming", *ACM SIGPLAN Notices,* volumen 8, número 8 (agosto de 1973), pp. 12-26.

4. Ned Chapin, "New Formal Flowcharts", *Software: Practice and Experience,* volumen 4, número 4 (octubre-diciembre 1974), pp. 341-357.

5. O. Ferstl, "Flowcharting by Stepwise Refinement", *ACM SIGPLAN Notices,* volumen 13, número 1 (enero de 1978), pp. 34-42.

6. M. Hamilton y S. Zeldin, *Top-Down, Bottom-Up, Structured Programming, and Program Structuring,* Charles Stark Draper Laboratory, documento E-2728. Cambridge, Mass: Massachusetts Institute of Technology, ciciembre de 1972.

7. Y. Futamura y otros, "Development of Computer Programs by PAD (Problem Analysis Diagram)", *Proceedings of the Fifth International Software Engineering Conference.* Nueva York: IEEE Computer Society, 1981, pp. 325-332.

8. HIPO: A *Design Aid and Documentation Technique,* IBM Corp., manual número GC20-1851-0. White Plains, N.Y.: IBM Data Processing Div., octubre de 1974.

9. Harry katzan, Jr., *Systems Design and Documentation: An Introduction to the HIPO Method.* Nueva York: Van Nostrand Reinhold, 1976.

10. Doug Ross y Ken Schoman, "Structured Analysis for Requirements Definition", *IEEE Transactions on Software Engineering,* volumen SE-3, número 1, enero de 1977, pp. 6-15. También reimpreso en *Classics in Software Engineering,* E. Yourdon (editor). Nueva York: YOURDON Press, 1979.

11. C.W. Bachman, "Data Structure Diagrams", *Data Base, The Quarterly Newsletter of the Special Interest Group on Business Data Processing of the ACM,* volumen 1, número 2 (verano de 1969), pp. 4-10.

12. Tom DeMarco, *Structured Analysis and Systems Specification,* Nueva York: YOURDON Press, 1978.

13. Michael Jackson, *Principles of Program Design.* Londres: Academic Press, 1975.

14. Larry Peters, *Software Design: Methods and Techniques.* Nueva York: YOURDON Press, 1981.

15. Ken Orr, *Structured Systems Development.* Nueva York: YOURDON Press, 1977.

16. Jean-Dominique Warnier, *Logical Construction of Programs,* 3ª edición, traducida por B. Flanagan, Nueva York: Van Nostrand Reinhold, 1976.

PREGUNTAS Y EJERCICIOS

1. ¿Por qué es importante familiarizarse con otras herramientas además de DFD, ERD y DTE?
2. ¿Cuáles son los tres componentes principales de un diagrama de flujo?
3. Proyecto de Investigación: ¿Qué otras imágenes se usan a veces en diagramas de flujo? Consulte [Chapin 1970] para más información.
4. ¿Cuántas flechas pueden emanar de una caja de processo en un diagrama de flujo?
5. ¿Cuál es la diferencia entre un diagrama de flujo y un diagrama de flujo de datos?
6. Dibuje un diagrama de flujo para un algoritmo de búsqueda binaria.
7. Dibuje un diagrama de flujo para un algoritmo sencillo de ordenamiento por intercambio.
8. Dibuje un diagrama de flujo para la aproximación de Newton-Raphson para el cálculo de la raíz cuadrada.
9. ¿Cuáles son las tres principales razones por las que no se usan los diagramas de flujo?
10. ¿Cuáles son las cuatro variantes principales de los diagramas de flujo?
11. ¿Qué es un diagrama de Nassi-Shneiderman? ¿Cuál es un sinónimo común del diagrama de Nassi-Shneiderman?
12. Dibuje un diagrama de Nassi-Shneiderman para un algoritmo de búsqueda binaria.
13. Dibuje un diagrama de Nassi-Shneiderman para un ordenamiento por intercambio sencillo.
14. Dibuje un diagrama de Nassi-Shneiderman para el método de Newton-Raphson de aproximación de la función raíz cuadrada.
15. ¿Qué es un diagrama de Ferstl?
16. Dibuje un diagrama de Ferstl para un algoritmo de búsqueda binaria.
17. Dibuje un diagrama de Ferstl para un ordenamiento por intercambio sencillo.
18. Dibuje un diagrama de Ferstl para el método de Newton-Raphson de aproximación de la raíz cuadrada.

19. ¿En qué difiere un diagrama de Ferstl de un diagrama de flujo? ¿Qué puede mostrar que el diagrama de flujo no pueda?

20. ¿Qué es un diagrama de Hamilton-Zeldin? ¿Cuál es un sinónimo de diagrama de Hamilton-Zeldin? ¿En dónde se desarrolló?

21. Dibuje un diagrama de Hamilton-Zeldin para un algoritmo de búsqueda binaria.

22. Dibuje un diagrama de Hamilton-Zeldin para un ordenamiento por intercambio sencillo.

23. Dibuje un diagrama de Hamilton-Zeldin para el método de Newton-Raphson para aproximar la raíz cuadrada.

24. ¿Qué es un diagrama PAD? ¿Dónde se desarrolló?

25. Dibuje un diagrama PAD para un algoritmo de búsqueda binaria?

26. Dibuje un diagrama PAD para un ordenamiento por intercambio sencillo.

27. Dibuje un diagrama PAD para el método de Newton-Raphson de aproximación de la raíz cuadrada.

28. ¿Qué características tienen en común los PAD y los diagramas de Ferstl?

29. ¿Qué es un diagrama de flujo de sistema? ¿Para qué se usa?

30. ¿En qué fase del desarrollo de un sistema de información es probable que se use un diagrama de flujo de sistema?

31. ¿Qué es un diagrama HIPO? ¿Dónde se desarrolló?

32. Dibuje un diagrama HIPO que muestre el diseño de un programa para jugar al gato (tic-tac-toe).

33. ¿Qué es un diagrama de entrada-proceso-salida? ¿Qué relación tiene este diagrama, conocido como IPO por sus siglas en inglés, con el concepto HIPO?

34. Dibuje un diagrama IPO para un algoritmo de búsqueda binaria.

35. Dibuje un diagrama IPO para un ordenamiento por intercambio sencillo.

36. Dibuje un diagrama IPO para el método de Newton-Raphson para aproximar la raíz cuadrada.

37. ¿Qué es un diagrama de estructura?

38. ¿Cuál es la diferencia entre un diagrama de estructura y un diagrama HIPO?

39. Dibuje un diagrama de estructura para un programa sencillo que juegue gato.

40. ¿Porqué resulta usualmente insuficiente usar el diagrama de estructura como herramienta de modelado en el análisis de sistemas?

41. ¿Qué es un diagrama SADT? ¿Cuál es la diferencia entre un diagrama SADT y un diagrama de flujo de datos?

42. ¿Qué es un diagrama de Bachman? ¿Cuál es la diferencia entre un diagrama de Bachman y un diagrama de entidad-relación?

43. ¿Qué es un diagrama de estructura de datos de DeMarco? Cuál es la diferencia con un diagrama de entidad-relación?

44. ¿Qué es un diagrama de estructura de datos de Jackson? Cuál es la diferencia con un diagrama de entidad-relación?

16 HERRAMIENTAS DE MODELADO PARA ADMINISTRACION DE PROYECTOS

Para beneficio de las personas de distintos tipos, la verdad científica debiera presentarse en formas diferentes, y considerarse igualmente científica, ya sea que aparezca en la forma robusta y de gran colorido de una ilustración física, o en la tenuidad y palidez de una expresión simbólica.

James Clerk Maxwell
Discurso a la Sección de Física y Matemáticas,
Asociación Británica para el Desarrollo de la Ciencia, 1870

En este capítulo se aprenderá:

1. Por qué la administración necesita herramientas propias.
2. Cómo dibujar diagramas PERT.
3. Cómo dibujar diagramas de Gantt.
4. Las relaciones entre herramientas de administración y otras herramientas de modelado.

6.1 INTRODUCCION

Aunque éste no es un libro sobre administración de proyectos, es apropiado hacer una pausa en este último capítulo sobre herramientas de modelado, para presentar algunas que son útiles para administrar un proyecto de desarrollo de sistemas; nos enfocaremos principalmente en las herramientas de modelado conocidas como *diagramas de PERT* y *de Gantt.*

¿Por qué deben interesarle las herramientas de modelado de administración? Existen varias razones posibles:

- Además de su papel como analista, tal vez también sea el administrador del proyecto, y los analistas de nivel inferior y los programadores le reporten directamente. Podría desarrollar modelos administrativos para presentarlos a los niveles superiores de la administración, o podría pedirle a uno de sus subordinados que los desarrolle.
- Como técnico superior del equipo, el administrador del proyecto podría pedirle que desarrolle modelos administrativos. En este caso, usted sería el administrador aprendiz, a quien el administrador del proyecto recurre para pedir ayuda y consejo sobre diversos asuntos de naturaleza tanto administrativa como técnica.
- Incluso si no es responsable del desarrollo de los modelos es importante conocerlos, pues reflejan la percepción de la administración sobre cómo debe estarse realizando su trabajo. Puede sacar sus propias conclusiones sobre si el proyecto tendrá éxito o no comparando los modelos con su propia percepción de la realidad.
- La organización del trabajo y la asignación de personas a diversas actividades, proceso a menudo conocido como división del trabajo del proyecto, usualmente será paralela a la división *técnica* del sistema. Dado que estará íntimamente involucrado con la descomposición del sistema en sus funciones y objetos de datos, podría tener influencia en la forma en que el proyecto se organice.[1]

En el resto del capítulo se examinarán las herramientas de modelado administrativo más comunes, y veremos cómo tratan los asuntos importantes a los que se enfrentan los administradores de proyectos. Se verá también cómo se relacionan con las otras herramientas de modelado de sistemas que se discutieron en los últimos capítulos.

1 A veces es justo lo contrario. Como dijo George Mealy, consultor de IBM, acerca de algunos proyectos: "La estructura del sistema es isomorfa a la estructura de la organización que lo crea".

16.2 ¿POR QUE REQUIERE MODELOS LA ADMINISTRACION?

Existen tres razones principales por las cuales la administración de proyectos requiere de modelos asociados con un proyecto de desarrollo de sistemas:

1. Para estimar el tiempo, dinero y personal necesario para desarrollar el proyecto.
2. Para actualizar y revisar dichas estimaciones a mecida que el proyecto avanza.
3. Para administrar las tareas y actividades de quienes laboran en el proyecto.

Las estimaciones de presupuesto son, desde luego, una actividad principal de los administradores en cualquier proyecto; en muchos proyectos de desarrollo de sistemas son aún más difíciles pues cada uno es (o por lo menos parece ser) único. En el apéndice B se discuten varias fórmulas y métodos para estimar la cantidad de trabajo a realizar en un proyecto y el número de personas que se ocuparán. Aun con eso, la administración necesita modelos —gráficos, de preferencia, por las mismas razones por las cuales son útiles en otros aspectos del análisis de sistemas— para ver cuándo estarán disponibles las personas para realizar las tareas del proyecto, qué sucederá si no están, etc. Examinaremos esto con más detalle en la Sección 16.5 .

Sin embargo, aun el mejor plan tiene probabilidades de fallar si se implanta a ciegas. Las circunstancias cambian continuamente durante un proyecto: los recursos críticos pudieran no estar disponibles en el preciso momento en que se requieren; miembros importantes del personal pueden enfermarse o retirarse; la estimación original de la cantidad de trabajo a realizar podría no ser exacta; el usuario anuncia de repente que necesita que el sistema arranque un mes antes de lo previsto; el administrador se da cuenta que el trabajo se está haciendo mucho más lentamente de lo esperado. Por todo ello, para el administrador es importante tener modelos que le permitan explorar las consecuencias de cambios en sus planes.

Y, finalmente, el administrador no sólo debe administrar *tareas, sino personas*. Debe asegurarse que todos los analistas, programadores, diseñadores y demás miembros del personal estén haciendo lo *que* deben hacer, *cuando* deben hacerlo. Por eso requiere herramientas de modelado que se enfoquen a las personas, además de herramientas que se enfoquen a las tareas.

16.3 DIAGRAMAS PERT

PERT es un acrónimo dado por las siglas en inglés de Técnica de Revisión de la Evaluación de Programas. Originalmente se desarrolló en los años 60 como herramienta de administración para el proyecto de submarinos Polaris de la fuerza naval de los Estados Unidos; suele darse el crédito a Booz Allen (la empresa consultora), a Lockheed Aircraft y a la Marina por haber desarrollado el concepto. Los diagramas PERT se han utilizado ampliamente en proyectos de la industria y el gobierno desde entonces; muchos los conocen como diagramas de actividad.

En la Figura 16.1 se muestra un diagrama PERT típico para un proyecto hipotético. Cada rectángulo representa una *tarea* o *actividad* (es decir, un fragmento reconocible de trabajo que debe hacerse). Los cuadros con esquinas redondeadas se conocer como señalamientos y tienen un significado obvio dentro del contexto de un proyecto típico. Las líneas que conectan los cuadros muestran *dependencias;* es decir, muestran qué actividades deben terminarse antes de comenzar otra. Las líneas más gruesas y obscuras que forman un camino contiguo del principio al final del proyecto representan el camino crítico, es decir, aquellas actividades cuyo retraso obligaría al retraso del proyecto global (las actividades que *no* están en el camino crítico tienen "tiempo de ocio" disponible; pueden comenzarse hasta después de un tiempo igual a éste luego de la fecha programada, si así se desea, sin afectar al proyecto en conjunto).

Observe que el administrador del proyecto(o un subordinado) es quien determina cuáles tareas dependen de otras. En muchos casos, la dependencia se relaciona con los datos: la actividad N+1 requiere como entrada algo que se produce como salida de la actividad N. En otros, la dependencia representa un punto de revisión del proyecto (por ejemplo, el señalamiento N pudiera ser una junta de revisión administrativa en la que se debe aprobar la labor realizada hasta ese momento, antes de comenzar la actividad N+1).

Observe que el ejemplo de la Figura 16.1 es realmente trivial: contiene sólo diez actividades y termina cuando concluye la actividad de análisis del sistema. Desde luego, un proyecto típico continúa aún después de terminarse la labor de análisis, e invierte una considerable cantidad de tiempo en el área de diseño, codificación, prueba, etc. De hecho, es probable que un proyecto típico tenga varios *cientos* de actividades que se mostrarían en un diagrama PERT. Tal vez quepa en la pared de la oficina del administrador del proyecto, pero ciertamente no cabría de manera conveniente en este libro.

De mayor importancia aún es el hecho de que muchos proyectos identifican actividades principales, las cuales posteriormente se dividen en otras menores. Por ejemplo, la Figura 16.1 muestra una actividad etiquetada "realizar actividades de análisis de sistemas". Como hemos visto en todo este libro, hay muchas cosas que entran en esta clasificación; de hecho, se esperaría ver un gran diagrama PERT que entre en detalles de estas subactividades. Como se vio en el concepto de diagramas de flujo de datos por niveles en el Capítulo 9, podría imaginarse el concepto de diagramas PERT por niveles para ayudar a organizar la complejidad de varios cientos, o incluso miles, de tareas en un proyecto grande.

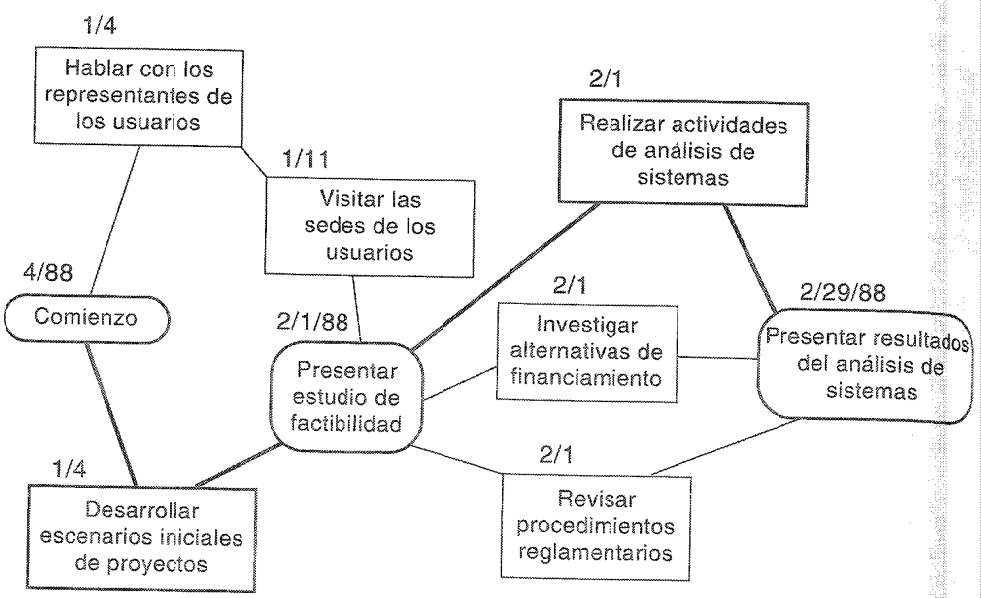


Figura 16.1: **Diagrama PERT**

Note, por cierto, que el diagrama PERT se enfoca a las actividades y sus interdependencias, pero dice poco o nada acerca de otros aspectos del proyecto que son de interés para un administrador. No indica, por ejemplo, qué persona o grupo debe llevar a cabo las diversas actividades: tampoco dice nada explícitamente acerca de la cantidad de tiempo (o número de días-persona) que cada actividad requiere. Y tampoco muestra qué productos o salidas se obtienen por cada actividad. Parte de esta información, sin embargo, se enfatiza en los modelos de administración que se discuten a continuación.

Finalmente, observe que el diagrama PERT parece suponer que todo se mueve hacia adelante, como indica la secuencia de izquierda a derecha de las actividades. De hecho, a menudo es necesario volver atrás y rehacer algunas de las actividades anteriores si se encuentran posteriormente. Este tipo de actividad iterativa no se muestra bien en un diagrama PERT típico.

Por otro lado, el diagrama PERT *sí* muestra, de manera clara, el hecho de que muchas actividades pueden llevarse a cabo paralelamente en un proyecto real. Esto es importante porque muchos de los otros modelos implican fuertemente que las actividades deben hacerse secuencialmente (véase por ejemplo, la Figura 5.1). Los administradores del proyecto generalmente quieren aprovechar lo más posible el "paralelismo", pues puede ayudar a reducir el tiempo necesario para el proyecto.

16.4 DIAGRAMAS DE GANTT

Un segundo tipo de modelo de administración de proyectos es el diagrama de *Gantt*, que a veces se conoce como itinerario de tareas. La Figura 16.2 muestra un diagrama de Gantt para el mismo proyecto hipotético que se usó en la Figura 16.1. Observe que cada actividad se muestra con una indicación de cuándo comienza y cuándo termina; el área sombreada indica tiempos de ocio posible, mientras que las actividades encerradas en rectángulos blancos pertenecen al camino crítico.

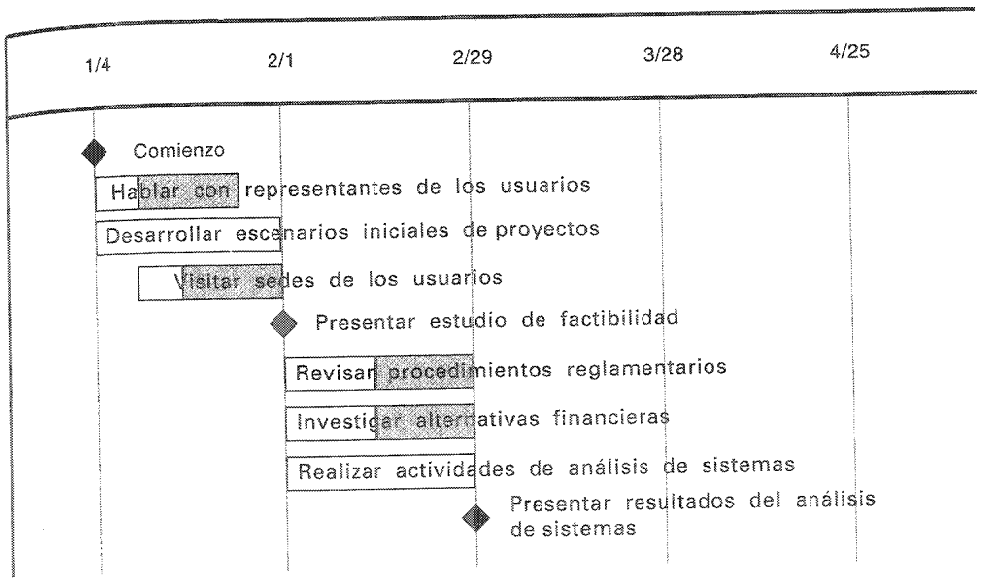


Figura 16.2: **Diagrama de Gantt**

Como puede verse, el diagrama de Gantt presenta en gran medida el mismo tipo de información que el diagrama PERT; su principal diferencia es el hecho de que muestra la duración de cada actividad,[2] mientras que el diagrama PERT no lo hace. Dado que es una representación un tanto más tabular del proyecto, a menudo puede usarse para presentar una gran cantidad de información en una forma relativamente compacta.

2 No hemos indicado exactamente cómo determina el administrador del proyecto la duración de cada tarea. En el caso sencillo puede estimarla por sí solo o pedir a quienes realizan el trabajo que hagan sus propias estimaciones. Si la tarea es grande y compleja, generalmente se dividirá en subactividades más pequeñas. En el Apéndice B se dan fórmulas para estimar el tiempo y recursos de programación, entre otras cosas.

16.5 HERRAMIENTAS ADMINISTRATIVAS ADICIONALES DE MODELADO

Además de las dos herramientas principales de modelado que se analizaron anteriormente, a los administradores de proyectos les agrada usar diagramas y tablas adicionales para ayudarles a estar al tanto de su labor. Por ejemplo, en lugar de mostrar las *tareas* como se hizo en la Figura 16.2, se puede fácilmente producir un diagrama que muestre la actividad de cada *recurso* del proyecto.[3] La Figura 16.3 muestra un listado de recursos para el mismo proyecto hipotético. Obviamente, esto es útil para estar al tanto de las diversas actividades de las que es responsable cada miembro del proyecto. De manera similar, podría decidirse que resultaría conveniente tener un listado tabular de las diversas actividades del proyecto, tal vez como indicación de cuál es la fecha más temprana en la que cada actividad puede iniciar, y la fecha más tardía en la que puede iniciar sin perturbar otras tareas ni retrasar la terminación del proyecto.

Debiera ser evidente que la información de la Figura 16.4 es otra visión más de los aspectos administrativos del proyecto; debe ser compatible con otros aspectos, como los diversos modelos de DFD, DER y DTE para un sistema de información son compatibles entre sí. De hecho, habiendo creado cualquiera de los modelos de administración del proyecto, debería poderse derivar los demás de manera mecánica; regresaremos a este tema en la Sección 16.7.

16.6 RELACIONES ENTRE HERRAMIENTAS DE ADMINISTRACION DE PROYECTOS Y OTRAS HERRAMIENTAS DE MODELADO DE SISTEMAS

¿Cuál es la relación entre los diagramas PERT, diagramas de Gantt y los diversos modelos del sistema (DFD, DER, DTE, etc.) que hemos discutido a lo largo de este libro? La relación más fuerte parece ser la existente entre el diagrama PERT y el diagrama de flujo de datos: ambos muestran actividades (o funciones) que se están llevando a cabo, y ambos muestran algo acerca de su interacción. Sin embargo, el DFD no muestra *nada* sobre la secuencia en la que dichas funciones se realizan, mientras que el diagrama PERT sí muestra cuáles actividades pueden realizarse de manera concurrente y cuáles deben realizarse de manera secuencial. Además, vimos que el diagrama PERT no muestra la salida producida por cada actividad, ni indica las entradas requeridas por cada actividad.

Como vimos en el Capítulo 5, los DFD se pueden usar para mostrar activicades en un proyecto, al igual que las entradas y salidas, por lo cual sería posible utilizarlo como herramienta de modelado en lugar del diagrama PERT. Sin embargo, la mayoría de los administradores de proyectos quisieran que el diagrama tuviera anotaciones para mostrar el camino crítico; y necesitarían información adicional, tal como la duración de cada actividad y las personas que estarían trabajando en cada una. De aquí que es más común ver la combinación del diagrama PERT clásico junto con el de Gantt y el calendario de recursos que discutimos anteriormente.

3 En la mayor parte de los proyectos los recursos que más nos interesa manejar son las personas, pero un recurso también pede ser una máquina, un cuarto de conferencias o cualquier otra cosa que sea (1) necesaria para el proyecto en algún momento dado y, (2) suficientemente escaso como para que se necesite administrar.

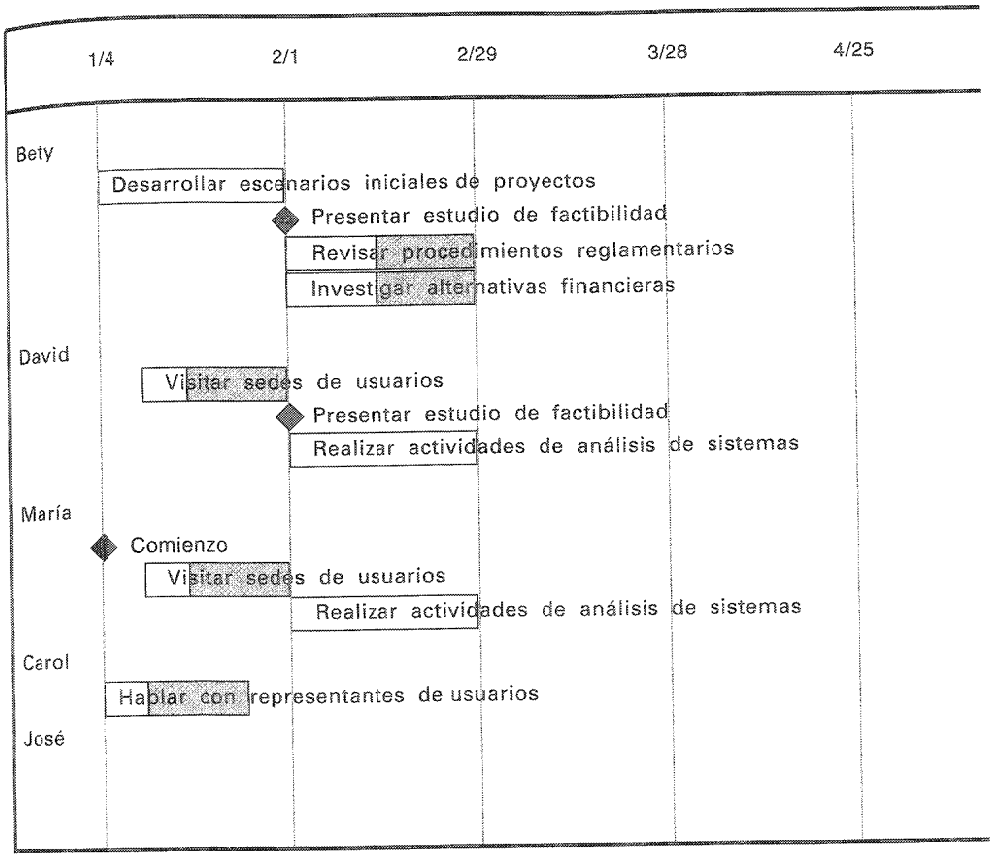


Figura 16.3: **Listado de recursos**

Es más significativo aún el hecho de que las actividades que se muestran en un diagrama PERT típico son las diversas actividades de dibujar DFD, DER, etc. Así, mientras que la Figura 16.1 muestra una actividad de alto nivel como "realizar análisis de sistemas", un diagrama PERT realista probablemente mostraría una lista de actividades como la siguiente:

- Dibujar diagramas de flujo de datos para el nuevo sistema
- Dibujar diagramas de entidad-relación para el nuevo sistema

- Dibujar diagramas de transición de estados para el nuevo sistema
- Desarrollar el diccionario de datos para el nuevo sistema
- Escribir las especificaciones del proceso para las burbujas de nivel inferior
- Balancear los modelos
- Hacer los cálculos de costo-beneficio
- Etc.

	Tarea	Días	Inicio más temprano	Final más temprano	Inicio más tardío	Final más tardío
1	Comienzo	0	1/4/88	1/4/88	1/4/88	1/4/88
2	Hablar con los representantes de los usuarios	5	1/4/88	1/11/88	1/19/88	1/26/88
3	Desarrollar escenarios iniciales	20	1/4/88	2/1/88	1/4/88	2/1/88
4	Visitar sedes de usuarios	4	1/11/88	1/15/88	1/26/88	2/1/88
5	Presentar estudio de factibilidad	0	2/1/88	2/1/88	2/1/88	2/1/88
6	Revisar procedimientos reglamentarios	10	2/1/88	2/15/88	2/15/88	2/29/88
7	Investigar alternativas de financiamiento	10	2/1/88	2/15/88	2/15/88	2/29/88
8	Realizar actividades de análisis de sistemas	20	2/1/88	2/29/88	2/1/88	2/29/88
9	Presentar resultados	0	2/29/88	2/29/88	2/29/88	2/29/88

Figura 16.4: **Listado de tareas**

Como veremos en la parte IV, las herramientas de modelado para los diagramas de flujo de datos y otros se utilizan para construir una serie de modelos diferentes del nuevo sistema. Así, es probable encontrar las siguientes actividades de alto nivel:

- Desarrollar modelos ambientales
- Desarrollar borradores del modelo de comportamiento

- Refinar modelo de comportamiento
- Desarrollar modelo de implantación del usuario

Hasta aquí, ninguno de estos términos tiene sentido; en el Capítulo 18 se discute el modelo ambiental, el de comportamiento en los Capítulos 19 y 20, y el de implantación del usuario en el Capítulo 21.

El punto principal a recordar es que las actividades que se muestran en los diagramas PERT y Gantt corresponden a las actividades de construcción de modelos que hemos discutido en todo este libro. Desde luego, un verdadero diagrama PERT de un proyecto, que cubre todo su ciclo de vida, debe también mostrar las actividades de diseño, programación, prueba, conversión de bases de datos e instalación.

16.7 EL ASUNTO DE LA AUTOMATIZACION

De la breve discusión sobre herramientas de administración de proyectos de este capítulo deben ser evidentes tres cosas:

1. Varias de las herramientas involucran gráficos; por ello, para que funcionen, alguien o *algo* tiene que dibujar las figuras.
2. Para un proyecto real grande, los modelos se vuelven inmensos. Y a medida que las cosas cambian (cosa que inevitablemente sucede durante un proyecto), los modelos tienen que volverse a dibujar. Esto implica una tremenda cantidad de trabajo.
3. Los modelos se relacionan todos entre sí por lo que, teniendo suficiente información acerca del proyecto, se debiera poder crear un diagrama PERT *o* de Gantt, *o bien* un itinerario de recursos, además del apoyo narrativo adecuado.

Esto lleva a una conclusión muy obvia: sería de tremenda ayuda si las herramientas de administración de proyectos pudieran computarizarse. Y de hecho, lo han sido; existe una variedad de paquetes de administración de proyectos disponibles para computadoras personales, además de para minicomputadoras o *mainframes*.[4] Así que un administrador de proyecto de inicios de los años 90 sería tonto si administrara cualquier proyecto, excepto los muy triviales, sin tales herramientas automatizadas. Además de las actividades de modelado simples discutidas en este capítulo, generalmente las herramientas computarizadas tienen las siguientes características:

4 Los diagramas que se muestran en este capítulo se hicieron con McProject en la computadora Apple Macintosh. En las PC IBM hay una variedad un tanto mayor de dónde escoger.

- La posibilidad de especificar el *costo* de cada recurso del proyecto. Esto es de gran ayuda en las actividades de presupuesto.
- La posibilidad de describir el calendario con el que deberá trabajar el proyecto (por ejemplo, vacaciones, horas hábiles normales, etc.). De hecho, algunos programas permiten que cada recurso tenga su propio calendario, y toman en cuenta que las distintas personas tienen diferentes períodos de vacaciones, etc.
- La posibilidad de programar hacia atrás o hacia adelante. En un proyecto normal, la fecha de inicio es conocida, y el objetivo es estimar cuándo estará terminado. Pero en otros casos se conoce la fecha de finalización (porque se ha impuesto externamente una fecha límite para su conclusión), y el objetivo es determinar la última fecha posible para el inicio de cada una de las actividades.[5]
- La posibilidad de ofrecer una variedad de reportes en diversos formatos.
- La posibilidad de tener interfaces con otros programas (por ejemplo, hojas de cálculo y programas gráficos).
- La posibilidad de comparar el desempeño real contra el desempeño estimado, para que el administrador pueda ver qué tan precisas son sus estimaciones y tal vez usar esto como medio para revisarlas en el futuro.

Para los proyectos pequeños o medianos suele ser bastante adecuado un paquete de administración de proyectos basado en una PC; entre los más populares están Microsoft Project, Timeline y el Harvard Project Manager. Para los proyectos grandes, donde se administran miles de tareas y cientos de recursos, pudiera requerirse una computadora más grande. Además, muchas organizaciones integran los planes de los proyectos individuales en modelos y presupuestos agregados; por ello es importante que todos usen sistemas de modelado basados en PC que sean compatibles, o bien que compartan algún sistema mayor basado en un *mainframe*.

16.8 RESUMEN

Obviamente, hay más sobre la administración de proyectos que dibujar diagramas PERT. Un administrador de proyecto típico tiene que ver con contrataciones y despidos, negociaciones y motivación, además de comunicación con programadores, analistas, usuarios y los niveles administrativos superiores. Sin la ayuda de las herramientas de modelado del tipo que se describieron en este capítulo es prácticamente imposible que siga la pista de todas las actividades, costos y recursos involucrados.

5 A Tom DeMarco le gusta llamarle a esto: "pedirle deseos al pasado".

REFERENCIAS

1. Philip Metzger, *Managing a Programming Project,* 2ª edición, Englewood Cliffs, N.J.: Prentice-Hall, 1983.
2. Tom Gildersleeve, *Successful Data Processing Systems Analysis,* 2ª edición, Englewood Cliffs, N.J.: Prentice-Hall, 1985.
3. Marvin Gore y John Stubbe, *Elements of Systems Analysis,* 3ª edición Dubuque, Iowa: William C. Brown, 1983.

PREGUNTAS Y EJERCICIOS

1. Dé tres razones por las cuales los administradores de proyectos requieren modelos asociados con el proyecto de desarrollo de sistemas.
2. ¿De qué es acrónimo PERT?
3. Dé una definición breve de diagrama PERT.
4. ¿Qué es el camino crítico en un diagrama PERT? ¿Por qué es importante?
5. ¿Cuál información *no* muestra el diagrama PERT acerca de un proyecto?
6. Dé una breve definición de diagrama de Gantt. ¿Qué sinónimo tiene?
7. ¿Qué información proporciona un diagrama de Gantt que el de PERT no proporcione?
8. Dé una definición breve de listado de recursos.
9. ¿Qué relación existe entre diagramas PERT, de Gantt y modelos de DFD de un sistema?
10. ¿Por qué es útil tener herramientas automatizadas para producir diagramas PERT y de Gantt?

PARTE III: EL PROCESO DE ANALISIS

17 EL MODELO ESENCIAL

Busca la esencia de una cosa, sea un punto de doctrina, de práctica, o de interpretación.

Marco Aurelio, *Meditaciones VIII*

En este capítulo se aprenderá:

1. Los cuatro principales modelos de sistemas en el ciclo de vida.
2. Por qué es peligroso modelar el sistema actual del usuario.
3. La distinción entre modelos esenciales y de implantación.
4. Cómo definir en términos lógicos un modelo de implantación.

En la sección anterior (Capítulos 9 a 16), examinamos varias *herramientas* de modelado que todo analista debe tener a su disposición. Sin embargo, dadas estas herramientas, ¿qué *tipo* de modelo debemos construir? ¿Debemos construir un modelo de la implantación actual del sistema del usuario? ¿Debemos construir uno de la implantación nueva que se propone? ¿Un modelo independiente de la tecnología de implantación? ¿Las tres cosas? Estas preguntas se contestan en los siguientes capítulos.

Comenzamos por examinar el enfoque del análisis estructurado clásico para desarrollar modelos de sistemas. Como veremos, hay grandes problemas con este enfoque. Luego, discutiremos el modelo *esencial,* que es el modelo de análisis de sistemas primario que recomendamos construir. Finalmente, discutiremos algunas reglas para la construcción de un modelo esencial a partir de un modelo de implantación existente.

17.1 EL ENFOQUE DEL MODELADO CLASICO Y POR QUE NO FUNCIONO

17.1.1 Los cuatro modelos de sistemas

Cuando se introdujo el análisis estructurado, comúnmente se argumentaba que el analista debería desarrollar los cuatro modelos distintos que se muestran en la Figura 17.1.

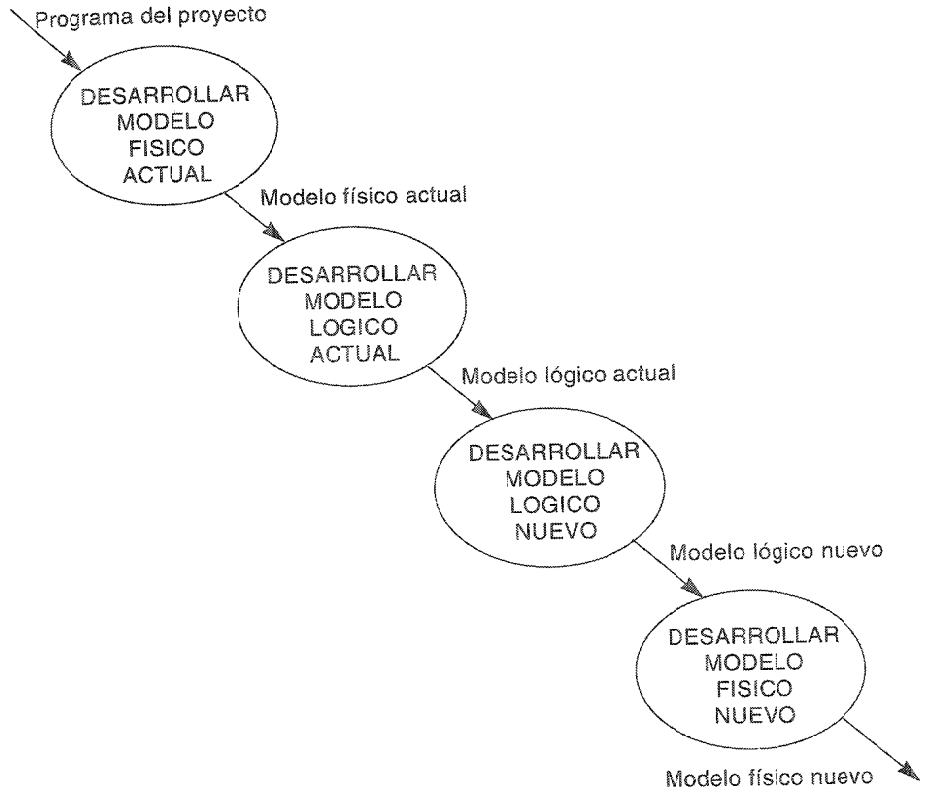


Figura 17.1: **Los cuatro modelos de sistemas**

El modelo *físico actual* es un modelo del sistema que actualmente está empleando el usuario. Puede ser un sistema manual, automatizado o mezcla de ambos. Típicamente, los procesos (burbujas) del diagrama de flujo de datos para el sistema físico actual se titulan con nombres de personas, de unidades organizacionales o de sistemas de cómputo que hacen la labor de transformar entradas en salidas. En la Figura 17.2 se muestra un ejemplo. Note también que los flujos de datos típicamente muestran la forma física de transporte de datos entre burbujas; además, los almacenes de datos pueden representarse con carpetas, archivos de cinta magnética o alguna otra tecnología.

El *modelo lógico nuevo* es un modelo de los requerimientos puros o esenciales del sistema *nuevo* que el usuario quiere. En el caso ideal (desde el punto de vista del analista), es igual que el *modelo lógico actual;* es decir, contiene las mismas funciones y datos. Esta situación es factible si el usuario está completamente satisfecho con la funcionalidad del sistema actual, pero no con su implantación.[1] En la

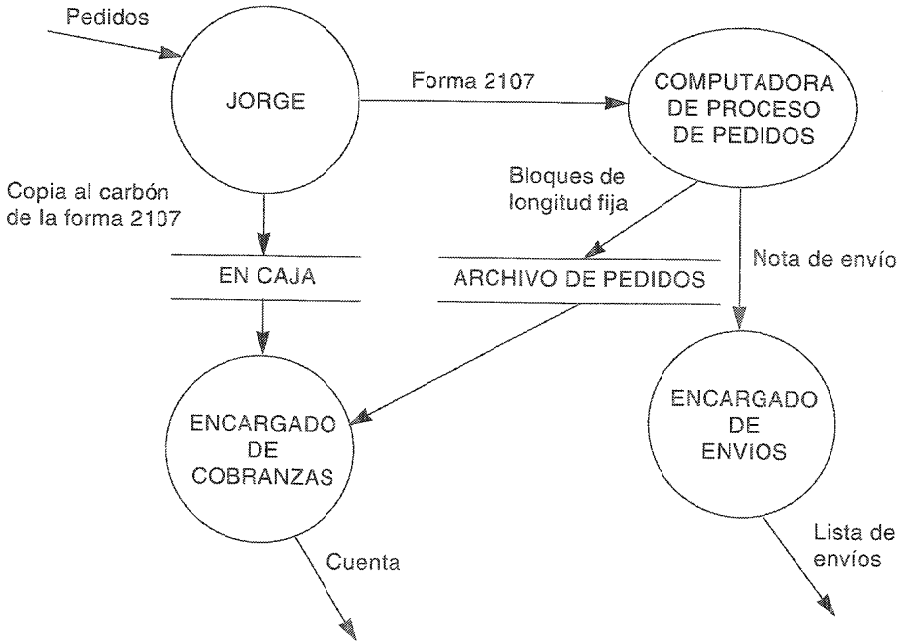


Figura 17.2: **Un modelo de sistema actual**

1 Existen muchas razones posibles para esto. El sistema puede estar implantado en hardware obsoleto o cuyo fabricante ya no exista. O la respuesta o el rendimiento del sistema podrían no ser los adecuados. O el usuario pudiera pedir que algunos de los datos que se mantienen manualmente (por ejemplo, archivos en papel) se computaricen. O, como es cada vez más común hoy en día, el software podría estar tan pobremente documentado que ya no se pueda mantener o modificar.

mayoría de los casos, sin embargo, el usuario pedirá funciones adicionales: "Aprovechando la ocasión, ¿no podría añadir otra transacción para cubrir la siguiente situación...." O el usuario podría pedir que el sistema maneje nuevo tipo de datos. Por eso, aunque un 80% a 90% del modelo lógico nuevo pudiera parecer idéntico al actual, es probable que haya algunos cambios y adiciones por lo menos.

El *modelo lógico actual* es el modelo de los requerimientos puros o esenciales que realiza el sistema actual del usuario. De esa forma se eliminan los detalles de la implantación arbitraria, y el modelo que resulta muestra lo que el sistema haría si hubiera disponible una tecnología perfecta.[2] En la Figura 17.3 se muestra un ejemplo de un modelo lógico actual.

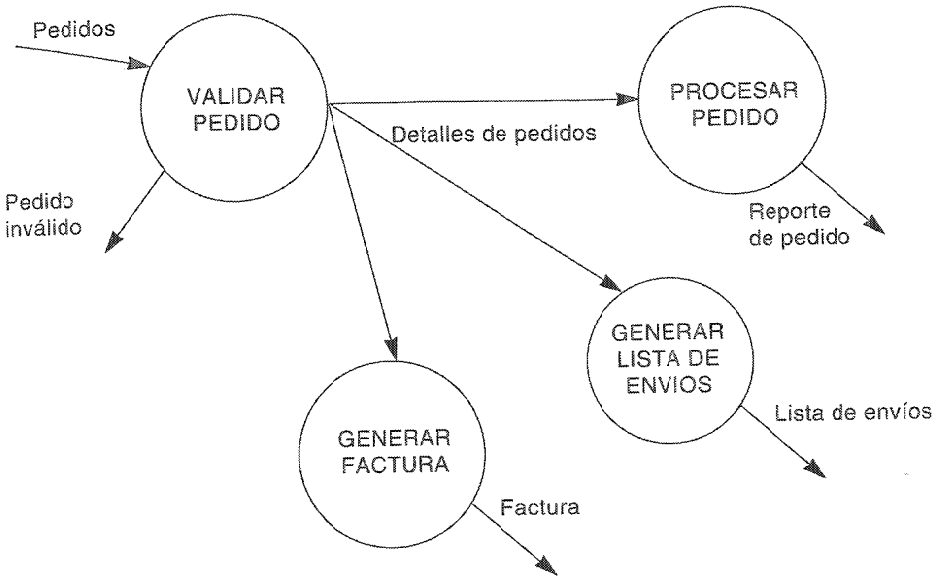


Figura 17.3: **El modelo lógico actual**

2 Se puede interpretar por tecnología perfecta el hardware que no cuesta nada, no consume energía, no genera calor, trabaja a velocidad infinita (es decir, realiza cualquier cálculo instantáneamente), almacena una cantidad infinita de datos que se pueden recuperar empleando cero tiempo, y que jamás se descompone ni comete errores.

El *nuevo modelo físico* es un modelo que muestra las *limitaciones* de implantación impuestas por el usuario. Una de las limitaciones más importantes es la determinación de la *frontera de automatización* (es decir, la determinación de cuáles funciones del nuevo sistema se automatizarán y cuáles se harán manualmente). El nuevo modelo físico corresponde a lo que ahora llamamos el *modelo de implantación del usuario,* que se discute con más detalle en el Capítulo 21.

17.1.2 Por qué no funcionó el enfoque clásico

El enfoque clásico descrito anteriormente se basaba en tres suposiciones principales:

1. El analista pudiera no estar familiarizado con el área de aplicación o del negocio: tal vez sea un experto en tecnología computacional, pero con sólo conocimientos superficiales de la banca, seguros, control de inventarios o cualquiera que sea el área en la que el usuario trabaja. Por ello es importante que el analista comience con un modelo físico actual como medio para educarse. El modelo que dibuje será relativamente fácil de verificar, porque contendrá varios señalamientos físicos que pueden observarse en el ambiente físico actual del usuario. Habiendo reunido esta información, el analista puede continuar, transformando el modelo físico en un modelo lógico.

2. El usuario pudiera estar renuente o imposibilitado para trabajar con el nuevo modelo lógico al principio del proyecto. La razón más común de esto es la duda sobre la capacidad del analista para desarrollar un modelo lógico del nuevo sistema. Aun si el analista cree que es un experto en el área de negocios del usuario, éste pudiera no estar de acuerdo. "¿Por qué he de confiarle el diseño de un nuevo modelo para mí, si ni siquiera entiende cómo trabaja mi negocio actualmente?", podría preguntarse. Además, algunos usuarios tienen dificultad con un modelo abstracto del sistema sin señalamientos reconocibles; podrían requerir un modelo del sistema físico actual como manera de familiarizarse con el proceso de análisis estructurado y asegurar que el analista no haya dejado de tomar en cuenta algo. (Una alternativa es el enfoque de prototipos, que se discute en el Capítulo 5.)

3. La transformación de un modelo lógico actual en un modelo lógico nuevo no requiere mucho trabajo y, en lo particular, no requiere de mucho trabajo desperdiciado. Como se indicó anteriormente, el usuario típicamente añadirá algunas nuevas funciones, o nuevos datos, al sistema que ya tiene, pero en su mayor parte (si no todo) el sistema *lógico* (o esencial) existente permanece intacto.

Estas suposiciones de hecho resultaron ser correctas en muchos proyectos. Sin embargo, ignoran un peligro mucho mayor: *el proceso de desarrollar un modelo del sistema actual puede requerir tanto tiempo y esfuerzo que el usuario se frustre e*

impaciente y termine por cancelar el proyecto. Para darse cuenta de esto, se debe tener en mente que:

- Algunos usuarios (y algunos administradores y programadores-analistas) consideran cualquier tipo de análisis de sistemas como una pérdida de tiempo, es decir, lo consideran una forma de "descansar" hasta que el *verdadero* trabajo del proyecto se presente (es decir, la codificación).

- Muchos usuarios comprensiblemente dudan de los méritos que pueda tener el modelado cuidadoso de un sistema que, *por definición, se superará y reemplazará como resultado del desarrollo del nuevo sistema.*

El problema ocurre más frecuentemente porque el analista se distrae con la tarea de modelar el sistema actual y empieza a pensar en él como un fin en sí mismo. Así, en lugar de sólo dibujar el o los DFD y documentar sólo unas cuantas especificaciones clave, a menudo dibuja todos los DFD y documenta cada especificación de proceso, a la vez que desarrolla un diccionario de datos *completo.*

Desafortunadamente, este enfoque casi siempre involucra un gran desperdicio de tiempo. De hecho, normalmente podrá esperarse que hasta un 75% del modelo físico se deseche en la transición del modelo físico al modelo lógico actual; o, dicho de otra manera, el modelo físico actual es típicamente tres o cuatro veces más grande que el modelo lógico actual. Esto se debe a la redundancia (el que la misma función se lleve a cabo en varias partes diferentes del sistema y se dupliquen o tripliquen varios datos), y a la verificación, validación y revisión de errores que son apropiadas en el sistema físico actual pero no en el sistema lógico actual.[3]

Todo esto pudiera parecer bastante obvio para el lector casual. Sin embargo, proyecto tras proyecto, se ha observado que los analistas se involucran tanto en el *proceso* de modelar que olvidan el objetivo último del usuario: producir un sistema que funcione. Como lo señala Steve McMenamin (coautor de [McMenamin y Palmer, 1984], "Las burbujas no se compilan".[4]

---

3 Sin importar si se está construyendo un modelo lógico (esencial) o físico (de implantación), usualmente resulta apropiado realizar alguna revisión de errores en los datos que *llegan al sistema desde el mundo exterior.* Sin embargo, al transmitir los datos de un lugar a otro *dentro* del sistema, el modelo lógico (esencial) *no* revisa errores, pues supone que el sistema se realizará con tecnología perfecta. En el modelo físico (de implantación), *sobre todo* un modelo del sistema físico *actual,* la revisión de errores es vital pues (1) parte del proceso es propenso a errores, sobre todo si lo llevan a cabo humanos, (2) el transporte de datos de un proceso a otro es propenso a errores, dependiendo del medio de comunicación que se use y, (3) el almacenado y recuperación de datos de los almacenes físicos pudiera ser una actividad propensa a errores.

4 Algún día sí se compilarán las burbujas. Es decir, la combinación de diagramas de flujo de datos, diccionarios de datos y especificaciones rigurosas de procesos pueden convertirse en entradas de un generador de código que produzca programas ejecutables. Sin embargo, aun en este caso, el esfuerzo de producir un modelo *físico* detallado es un desperdicio de tiempo. Nadie quiere una réplica computarizada del sistema actual.

Consecuentemente, este libro recomienda que el analista *evite* modelar el sistema actual de ser posible. Las herramientas de modelado que se discuten en la Parte II deben usarse para comenzar, *tan pronto como sea posible,* a desarrollar un modelo del nuevo sistema que el usuario desea. Este nuevo sistema, conocido en los libros clásicos de análisis estructurado como el nuevo sistema lógico, se conoce aquí como el *modelo esencial* del sistema.

Ocasionalmente existirá alguna situación en la que el analista *deba* construir un modelo del sistema actual del usuario; esto sucede, por ejemplo, si el analista necesita modelar el sistema físico actual para poder descubrir los procesos esenciales. Esta situación se discute más a fondo en la Sección 17.3 .

17.2 EL MODELO ESENCIAL

17.2.1 Qué es

El modelo esencial del sistema es un modelo de *lo que* el sistema debe hacer para satisfacer los requerimientos del usuario, diciendo lo mínimo posible (de preferencia *nada*) acerca de *cómo* se implantará. Como se mencionó antes, esto significa que nuestro modelo del sistema supone que se tiene disponible una tecnología perfecta y que se puede obtener fácilmente y sin costo.

Específicamente, esto significa que cuando el analista habla con el usuario acerca de los requerimientos del sistema, debe evitar describir implantaciones específicas de procesos (burbujas en un diagrama de flujo de datos) en el sistema; es decir, no debe mostrar las funciones del sistema que están siendo realizadas por humanos o por sistemas de cómputo existentes. Como lo ilustran las Figuras 17.4(a) y (b), éstas son opciones arbitrarias de cómo *podría* implantarse el sistema; pero esta decisión debería retrasarse hasta que haya comenzado la actividad de diseño del sistema.[5] La Figura 17.4(c) muestra un modelo esencial más apropiado de lo que la función del sistema debe realizar sin importar su implantación final.

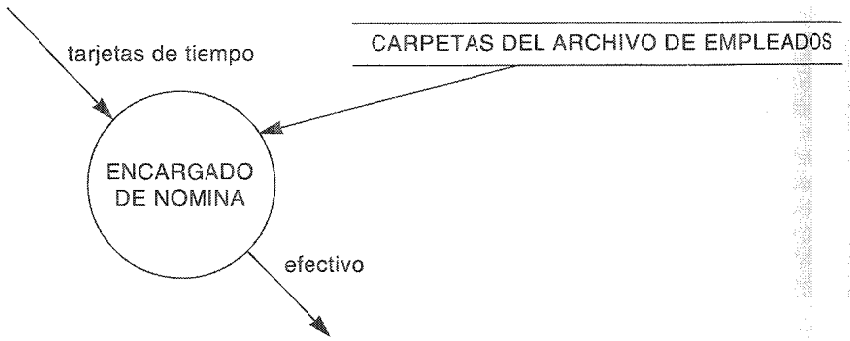


Figura 17.4(a): **Modelo de *cómo* hará su labor una función del sistema**

Lo mismo se da para los flujos y almacenes de datos: el modelo esencial debe describir el *contenido* de los flujos o almacenes de datos, sin describir el medio (por ejemplo, disco o cinta) u organización física de los datos.

17.2.2 Dificultades en la construcción de un modelo esencial

Aunque las reglas anteriores parecen simples y obvias, a menudo resulta muy difícil eliminar completamente todos los detalles de la implantación en el modelo esencial. Los ejemplos más comunes de detalles de implantación son:

- *Secuenciado arbitrario de las actividades en un modelo de flujo de datos.* El único secuenciado en el diagrama de flujo de datos debe ser el que requieren los *datos* (por ejemplo, la burbuja 2 puede requerir un dato producido por la burbuja 1 y por tanto no trabajar sino hasta que aquella haya terminado) o los acontecimientos externos al sistema.
- *Archivos innecesarios,* es decir, los almacenes de datos que no se requerirían de existir una tecnología perfecta. Los archivos temporales (o intermedios) se requieren en un modelo de implantación porque los procesos están programados para hacer su trabajo en distintos tiempos (por ejemplo, un programa nocturno por lotes produce un archivo que el sistema en línea diurno emplea); también se introducen para propósitos de respaldo y recuperación, porque la tecnología de implantación es propensa a errores, así como las personas que operan las computadoras.

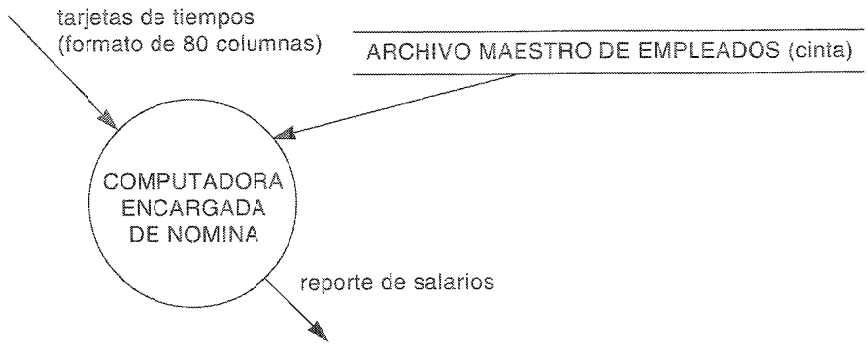


Figura 17.4(b): **Otro modelo de cómo se realizará la función del sistema**

5 Un término popular para referirse a esto es "aplazamiento constructivo". Mi colega Steve Weiss prefiere "diferimiento seguro" y es, de hecho, el principio en el cual se basa el enfoque descendente.

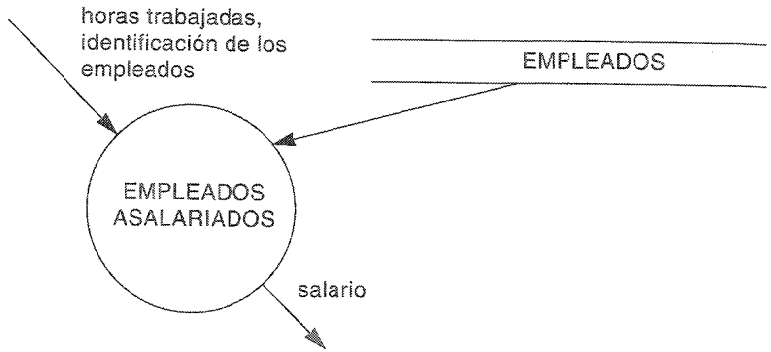


Figura 17.4(c): **Un modelo de cuál es la función del sistema**

- *Revisión de errores y validación innecesarias de datos y procesos dentro del sistema.* Dichas actividades de validación se necesitan en un modelo de implantación, porque se debe trabajar con procesos propensos a errores (por ejemplo, algunas funciones las realizan humanos, que son notablemente propensos a errores) y canales ruidosos de datos entre procesos.

- *Datos redundantes o derivados.* A veces se incluyen datos redundantes en los almacenes de datos para propósitos de eficiencia; aunque esto usualmente es razonable, debe hacerse durante la fase de diseño del proyecto, y no durante el modelado de las funciones y datos esenciales. Además, sin darse cuenta, el analista puede incluir datos que sean derivables o calculables a partir de los valores de otros datos.

17.2.3 Componentes del modelo esencial

El modelo esencial consiste en dos componentes principales:

1. Modelo ambiental
2. Modelo de comportamiento

El modelo ambiental define la *frontera* entre el sistema y el resto del mundo (es decir, el ambiente en el cual existe el sistema). Esto se discute con más detalle en el Capítulo 19. Como veremos, consiste en un diagrama de contexto, una lista de acontecimientos y una descripción breve del propósito del sistema.

El modelo de comportamiento describe el comportamiento que del sistema se requiere para que interactúe de manera exitosa con el ambiente. Los capítulos 20 y 21 describen una estrategia para derivar el modelo del comportamiento; el modelo consiste en los ya familiares diagramas de flujo de datos, de entidad-relación, de transición de estados y diccionarios y especificaciones del proceso que se han discutido anteriormente en el libro.

17.3 QUE HACER SI *SE DEBE* CONSTRUIR UN MODELO DE IMPLANTACION

Como se mencionó en este capítulo, existen circunstancias en las cuales podría ser deseable o necesario construir un modelo de implantación antes de construir el modelo esencial del sistema. Típicamente, esto se deberá a que el usuario no esté convencido de que se entiende su negocio lo suficientemente bien como para modelar un sistema nuevo, o porque el analista mismo haya decidido que necesita estudiar el ambiente actual antes de proponer un sistema nuevo.

Si decide proceder de este modo, lo principal a recordar es que su primer objetivo es llegar a un entendimiento y una visión generales del sistema existente. *No se trata de documentar el sistema actual con detalle.* Así, probablemente sea útil o apropiado crear uno o más niveles de diagramas de flujo de datos del sistema actual. Probablemente sea apropiado generar un diagrama de entidad-relación y podría resultar útil escribir las especificaciones del proceso para algunas de las funciones más críticas (u obscuras) del sistema. *Podría* ser útil reunir algunos de los documentos físicos que representarían un diccionario de datos físico. Pero *no* intente escribir especificaciones de proceso para todas las funciones, ni trate de desarrollar un diccionario de datos completo para el sistema existente.

Cuando haya terminado de desarrollar el modelo de la implantación actual, su siguiente tarea será definirlo en términos lógicos (es decir, eliminar cuantos detalles de implantación sea posible). Esto usualmente abarcará los siguientes pasos:

- *Buscar y separar flujos esenciales que hayan sido empaquetados de manera arbitraria en el mismo medio.* Por ejemplo, podría encontrarse con que en el sistema actual diversos datos se transmiten juntos de una computadora a otra mediante algún enlace común de telecomunicaciones; o que varios datos no relacionados se copian a papel para transmitirse a diversas funciones.

- *Buscar flujos empaquetados o agregados que se envían a burbujas (que representan a personas, computadoras, etc.) que no requieren de todos los datos que hay en dichos flujos.* La Figura 17.5(a) muestra un proceso, **CALCULAR FACTOR FRAMMIS,** que requiere sólo de un dato **X**; mientras tanto, otro proceso, **CALCULAR FACTOR-W,** requiere sólo del dato **Y**. Por conveniencia, la implantación actual ha empaquetado **X** y **Y** en un dato agregado **Z;** la definición de este modelo en términos lógicos resultaría en el diagrama de flujo de datos que se muestra en la Figura 17.5(b).

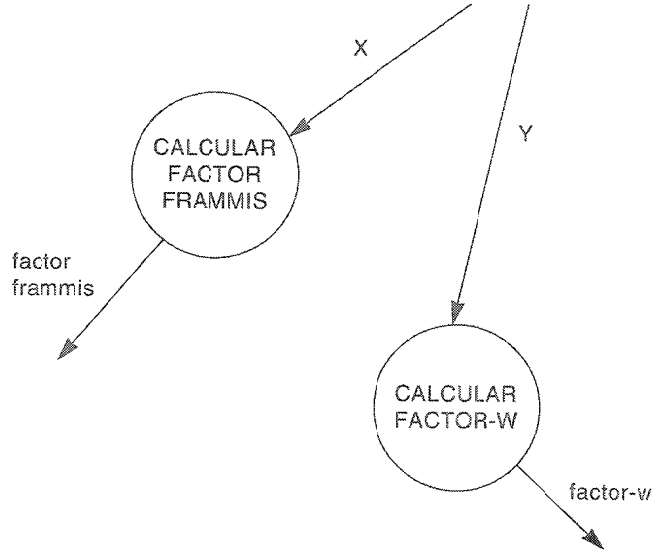


Figura 17.5(a): **Modelo físico**

- *Distinguir entre el trabajo esencial realizado por un proceso y la identificación del procesador que aparece en el modelo de implantación.* El procesador puede ser una persona o una computadora, o alguna otra forma de tecnología; un procesador individual podría estar ejecutando fragmentos de uno o más procesos esenciales, o bien de múltiples procesos esenciales en su totalidad. Como veremos en el Capítulo 20, los procesos esenciales deben agruparse si son iniciados por el mismo acontecimiento externo.

- *Eliminar procesos cuyo único propósito sea transportar datos de un lugar a otro dentro del sistema.* Además, elimine las burbujas responsables de entradas y salidas físicas entre el sistema y el ambiente exterior. El modelo físico de un sistema podría mostrar, por ejemplo, una función de correspondencia o mensajería; debe eliminarse del modelo esencial. Muchos DFD físicos tienen procesos con nombres como "obtener entradas del usuario" o "imprimir reporte"; también deben eliminarse.

- *Eliminar procesos cuya labor sea verificar datos que se producen y usan dentro del sistema.* Dado que en el modelo esencial se supone una tecnología perfecta, esa verificación y revisión interna no es necesaria. Es apropiado, sin embargo, proporcionar alguna revisión de errores para los datos provenientes del exterior. Así, se debe sospechar de cualesquiera procesos cuyos nombres sean "volver a revisar..." o "verificar..." o "validar..." o "editar..." a menos que existan en la frontera del sistema y manejen entradas externas.

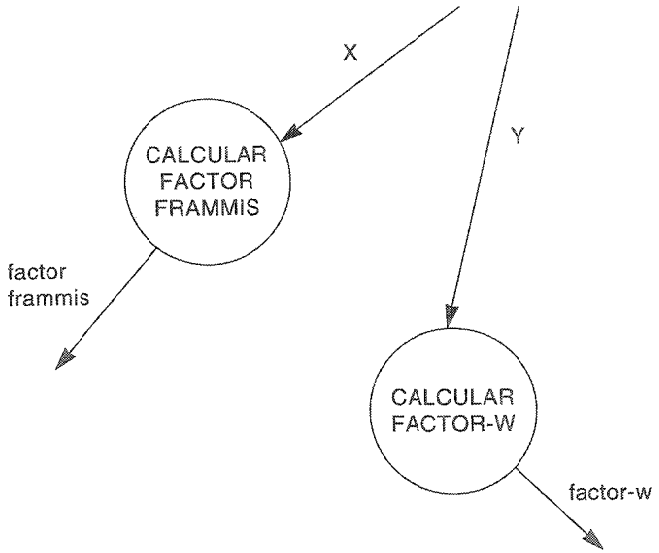


Figura 17.5(b): **La versión lógica**

- *Buscar situaciones donde los almacenes esenciales se hayan empaquetado en el mismo almacén de implantación* (por ejemplo, archivos en disco, en cinta o en papel); esto es muy común en los sistemas de segunda generación y en sistemas que se han optimizado a lo largo de años para manejar grandes volúmenes de datos de manera eficiente. Separe el contenido del almacén del medio de almacenaje.

- *Eliminar datos de los almacenes si ningún proceso los usa;* además, elimine datos de los almacenes si se pueden calcular o derivar directamente a partir de otros. (Note que los datos derivados y las copias redundantes de datos pueden reinsertarse posteriormente cuando se desarrolle el modelo de *implantación* durante el diseño del sistema).

- *Finalmente, eliminar almacenes de datos que sólo existan como separadores de tiempo entre procesos, y que sean dependientes de la implantación.* Esto incluye archivos intermedios, archivos de reportes, archivos de impresión diferida y otros similares.

17.4 RESUMEN

El concepto de modelo esencial parece bastante natural, pero no es tan fácil de lograr en proyectos reales como pudiera pensarse. La mayoría de los usuarios están tan involucrados en los detalles de la implantación de su sistema actual que les es difícil enfocarse a la visión de "tecnología perfecta" de un sistema. Y es igualmente difícil para muchos analistas veteranos, pues han pasado tantos años construyendo sistemas que les es difícil evitar hacer suposiciones relacionadas con la implantación al describir el sistema.

Recuerde que es críticamente importante desarrollar el modelo esencial de un sistema, pues(como se recalcó varias veces a lo largo de este libro) muchos sistemas de información grandes tienen una vida media de unos 10 a 20 años. Durante ese período se puede esperar que el hardware mejore por lo menos por un factor de mil, y probablemente se acerque más a un *millón* o más. Una computadora un millón de veces más rápida, pequeña y barata que las actuales es en verdad algo muy cercano a la tecnología perfecta; debe empezarse *hoy* a modelar sistemas como si se tuviera esa tecnología a la disposición.

REFERENCIAS

1. Tom DeMarco, *Structured Analysis and Systems Specification.* Nueva York: YOURDON Press, 1978.
2. Chris Gane y Trish Sarson, *Structured Systems Analysis: Tools and Techniques.* Englewood Cliffs, N.J.: Prentice-Hall, 1978.
3. Edward Yourdon, *Managing the Systems Life Cycle.* Nueva York: YOURDON Press, 1982.
4. Victor Weinberg, *Structured Analysis.* Nueva York: YOURDON Press, 1978.
5. Steve McMenamin y John Palmer, *Essential Systems Analysis.* Nueva York: YOURDON Press, 1984.

PREGUNTAS Y EJERCICIOS

1. ¿Cuáles son los cuatro modelos que recomiendan los libros de análisis de sistemas clásico?
2. ¿Qué es un modelo físico actual?
3. Dé tres ejemplos de procesos físicos (burbujas).
4. Dé tres ejemplos de almacenes físicos.
5. Dé tres ejemplos de flujos de datos físicos.

6. ¿Qué es un modelo lógico actual?
7. ¿Cuál es la diferencia entre un modelo físico actual y un modelo lógico actual?
8. ¿Qué es tecnología perfecta en el contexto de este capítulo?
9. ¿Qué es un modelo lógico nuevo?
10. ¿Cuál es la diferencia entre modelo lógico actual y nuevo?
11. ¿Bajo qué circunstancias podrían ser iguales el modelo lógico actual y nuevo de un sistema?
12. ¿Qué grado de traslape debe esperar el analista entre el modelo lógico actual y el nuevo?
13. ¿Qué es un modelo físico nuevo?
14. ¿Qué otro nombre hay para el modelo físico nuevo?
15. ¿Cuál es la principal restricción que describe el modelo físico nuevo?
16. ¿Cuáles son las tres suposiciones principales en las que se basa el enfoque clásico del análisis estructurado?
17. Proyecto de investigación: En su organización, ¿qué porcentaje de proyectos tienen analistas no familiarizados con el área de negocios del usuario? ¿Es razonable este porcentaje en su opinión? ¿Está cambiando?
18. ¿Cuáles son las dos principales razones por las que el usuario podría tener dificultades al leer y entender un modelo lógico?
19. ¿Cuál es el principal problema del enfoque clásico del análisis estructurado?
20. ¿Por qué dudan algunos usuarios de los méritos de modelar su sistema actual?
21. ¿Cuánto del modelo físico actual es probable que se deseche en la transición a un modelo lógico actual?
22. ¿Cuáles son las razones por las cuales el modelo físico actual es mucho mayor que el modelo lógico actual del sistema?
23. ¿Qué sinónimo hay de modelo lógico nuevo?
24. ¿Qué tipo de revisión de errores es apropiado en un modelo lógico? ¿Qué tipo es inapropiado? ¿Por qué?
25. Dé una definición de modelo esencial de un sistema.

26. ¿Qué significa aplazamiento constructivo en el contexto de este capítulo?

27. En un proyecto de desarrollo de sistemas, ¿cuándo debe tomarse la decisión de implantar una función (es decir, un proceso en un DFD) con una persona en lugar de con una computadora?

28. Cuáles son los cuatro errores comunes que suelen cometer los analistas cuando tratan de crear un modelo esencial?

29. ¿Por qué no deben mostrarse los archivos temporales en un modelo esencial?

30. ¿Cuándo *deben* mostrarse los archivos temporales en un modelo de sistema? ¿Por qué?

31. ¿Cuándo deben mostrarse datos redundantes en un modelo de sistema?

32. ¿Cuándo deben mostrarse datos derivados en un modelo de sistema?

33. ¿Cuáles son los dos componentes del modelo esencial de un sistema?

34. ¿Cuál es el propósito del modelo ambiental de un sistema?

35. ¿Cuál es el propósito del modelo de comportamiento de un sistema?

36. Si tuviera que documentar la implantación actual de un sistema, ¿qué debería tener cuidado de evitar?

37. ¿Es buena idea documentar todos los flujos de datos de la implantación actual del sistema? ¿Por qué?

38. ¿Es buena idea documentar todas las especificaciones de procesos en la implantación actual del sistema? ¿Por qué?

39. ¿Es buena idea documentar todos los elementos del diccionario de datos en la implantación actual del sistema? ¿Por qué?

40. Cuando se define en términos lógicos un modelo físico actual, ¿qué debe hacerse con los flujos esenciales que se empaquetaron en el mismo medio?

41. Cuando se define en términos lógicos un modelo físico actual, ¿qué se debe hacer con los flujos empaquetados enviados a procesos que no requieren todos los datos?

42. Cuando se define en términos lógicos un modelo físico actual, ¿qué debe hacerse con los procesos cuyo único propósito es transportar datos de un lugar a otro?

43. Cuando se define en términos lógicos un modelo físico actual, ¿qué se debe hacer con las burbujas cuyo único propósito es verificar los datos que se crean *dentro* del sistema?

44. Cuando se define en términos lógicos un modelo físico actual, ¿qué debe hacerse con los almacenes esenciales que se empaquetaron juntos en el mismo medio?

45. Cuando se define en términos lógicos un modelo físico actual, ¿qué se debe hacer con los datos que existen en almacenes pero no se usan en ninguna parte del sistema?

46. Cuando se define en términos lógicos un modelo físico actual, ¿qué debe hacerse con los archivos temporales que se encuentran en el sistema físico actual?

18 EL MODELO AMBIENTAL

La estabilidad del medio interno es una condición primaria para la libertad e independencia de ciertos cuerpos vivos en relación con el ambiente que los rodea.

Claude Bernard, *Leçons sur les Phenomenes de la Vie Communs aux Animaux et aux Vegetaux*, 1875-1879

En este capítulo se aprenderá:

1. Por qué la frontera del sistema es arbitraria pero crítica.
2. Cómo dibujar un diagrama de contexto para un sistema.
3. Cómo producir una lista de acontecimientos para un sistema.
4. Cómo usar el diagrama de contexto y la lista de acontecimientos para construir el modelo ambiental.

Para el analista, la labor más difícil en la especificación de un sistema es a menudo determinar qué es parte del sistema y qué *no*. Cualquier sistema que desarrolle, no importa lo ambicioso ni grandioso, será parte de un sistema aún mayor. Como vimos en el Capítulo 2, casi todos los sistemas con los que tenemos experiencia humana son meramente subsistemas de sistemas mayores: aun si nuestra labor fuera "diseñar el mundo", tendríamos que reconocer que éste es sólo parte del sistema solar, el cual es parte de una galaxia pequeña y obscura, la cual es parte del universo.

Así, el primer modelo importante que se debe desarrollar como analista es uno que no haga más que definir las *interfaces* entre el sistema y el resto del universo, es decir, el *ambiente.* Por razones obvias, este modelo se conoce como el *modelo ambiental.* Modela el exterior del sistema; el modelo del *interior* del sistema, conocido como modelo *del comportamiento*, se discute en los Capítulos 20 y 21.

Además de determinar qué está en el *interior* del sistema y qué en el exterior (lo que se logra definiendo la *frontera* entre el sistema y el ambiente), también es críticamente importante definir las *interfaces* entre el sistema y el ambiente. Se necesita saber qué información entra al sistema desde el ambiente exterior, y qué información produce como salida al ambiente externo.

Desde luego, las entradas y salidas no se producen al azar: ningún sistema de información toma todos los datos disponibles en el universo, ni expulsa cosas al azar al ambiente exterior ningún sistema realista. Los sistemas que construimos son racionales y tienen propósito; específicamente, producen salidas como *respuesta* a algún *acontecimiento,* o *estímulo,* en el ambiente. Así, otro aspecto crítico del modelo ambiental consiste en identificar *los acontecimientos que ocurren en el ambiente al cual debe responder el sistema.* No para todos los acontecimientos; después de todo, el ambiente en su totalidad genera un número infinito de acontecimientos. Sólo nos preocupan aquellos que (1) ocurren en el ambiente exterior y (2) requieren una respuesta del sistema.

Observe que la frontera entre un sistema y su ambiente, como ilustra la Figura 18.1, es *arbitraria.* Puede haberse creado por algún decreto administrativo, como

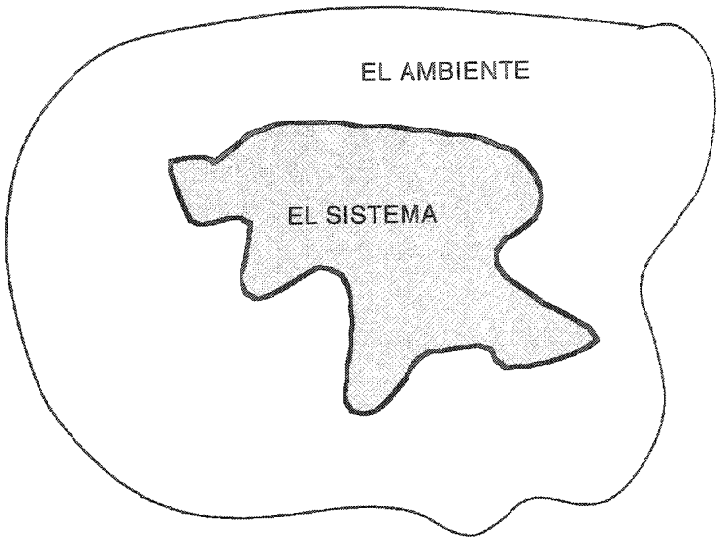


Figura 18.1: **La frontera entre el sistema y el ambiente**

resultado de alguna negociación política, o simplemente por accidente. *Y eso es algo que el analista de sistemas usualmente tiene oportunidad de influenciar.*

Generalmente el usuario tendrá una buena idea de la frontera *general* entre el sistema y el ambiente. Pero, como ilustra la Figura 18.2, a menudo existe un "área gris" que está abierta a negociaciones, es decir, un área sobre la cual el usuario (1) no está seguro, (2) no había pensado, (3) tenía algunas ideas preconcebidas que está dispuesto a reflexionar o, (4) todas las anteriores.

Por ejemplo, el usuario podría pedirle al analista desarrollar un sistema de cuentas por cobrar. Aunque esto pudiera representar una frontera firme y bien definida entre el sistema (conocido como el sistema C/C) y el ambiente, el analista ciertamente debiera considerar el "área gris", como ilustra la Figura 18.3, de cuentas por pagar, control de inventarios, manejo de efectivo, facturación y entrada de pedidos, como perspectiva un tanto más amplia.

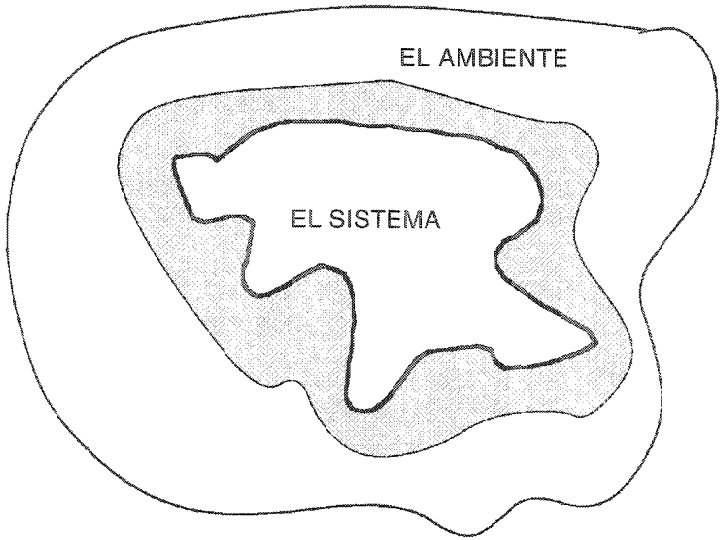


Figura 18.2: **El área gris entre el sistema y el ambiente**

Si el analista escoge una perspectiva demasiado *pequeña* para un proyecto está condenado al fracaso, pues el usuario puede haber identificado sin darse cuenta el *síntoma* del problema (por ejemplo, "nuestras cuentas por cobrar están fuera de control") en lugar de la *causa* del problema. Y si el analista, por exceso de confianza, ingenuidad o exuberancia, escoge una perspectiva demasiado *amplia* para el proyecto, está condenado al fracaso, puesto que estará tratando con una situación

política bastante más compleja, y estará intentando desarrollar un sistema demasiado grande bajo cualquier circunstancia. Además pudiera estar tratando asuntos que no le importan al usuario o que no se pueden cambiar en lo absoluto. Así que es importante dedicar bastante tiempo y tener suficiente participación del usuario en la elección de una frontera apropiada para el sistema.

En un sistema grande se puede tomar en cuenta una cantidad de factores cuando se está escogiendo la perspectiva del proyecto. Entre los más importantes están los siguientes:

- *El deseo del usuario de lograr una cierta participación en el mercado para el producto, o incrementarla a más de su nivel actual.* Esto puede hacerse ofreciendo un nuevo producto o una mayor funcionalidad de uno existente (por ejemplo, la mayor funcionalidad que ofrecen los sistemas de cajero automático y los sistemas bancarios en línea). O el usuario podría

Figura 18.3: **El área gris que rodea a los sistemas de cuentas por cobrar**

tratar de aumentar su mercado ofreciendo un mejor y más rápido servicio (por ejemplo,"todos nuestros pedidos se surten en menos de 24 horas, y tenemos un elaborado sistema de rastreo de pedidos para saber dónde se encuentra en todo momento").

- *La legislación establecida por el gobierno federal, estatal o de la ciudad.* La mayor parte de tales sistemas son de informes, por ejemplo, que reportan el empleo (o desempleo) de trabajadores basándose en edad, sexo, nacionalidad, etc. O podría tenerse que hacer un nuevo sistema para considerar los cambios en las leyes sobre impuestos.

- *Deseo del usuario por minimizar gastos operativos de algún área de su negocio.* Esto era particularmente común en las compañías grandes en los años 60, y sucede con muchos negocios pequeños que están instalando su primera computadora. La mayor parte de las organizaciones que han tenido computadoras instaladas durante 10 años o más ya aprovecharon las oportunidades obvias de reducir el personal de oficina.

- *Deseo del usuario por lograr alguna ventaja estratégica para la línea de productos o área de negocios que opera.* El usuario intenta hacerlo organizando y manejando información sobre el mercado para poder producir artículos de manera más oportuna y económica. Un buen ejemplo de esto son las líneas aéreas (al igual que muchas otras industrias recientemente desreglamentadas) donde mejor información acerca de tendencias del mercado y preferencias de los clientes pueden llevar a costos de pasajes e itinerarios de aerolíneas más eficientes.

El área dentro de la frontera del sistema a veces se conoce como el *dominio de cambios*. Por esto, simplemente queremos decir que todo lo que está dentro de la frontera del sistema está sujeto a cambios (por ejemplo, reorganización y/o automatización), mientras que todo lo que está fuera se queda en su forma actual y no es investigado por el analista.

Para ver dos ejemplos de fronteras de sistemas, examine los casos de estudio de los apéndices F y G. En el caso del Sistema de Información de YOURDON Press (Apéndice F), la frontera es un tanto mayor de lo que se esperaría: incluye la facturación y el manejo de recibos que normalmente son parte del departamento de contabilidad (y por tanto estarían fuera de la frontera); de manera similar, el controlador del elevador del Apéndice G tiene una frontera un tanto menor que lo deseable: se hubiera desarrollado un sistema muy distinto si los paneles de control del elevador se consideraran parte del ambiente. En ambos casos, las elecciones fueron arbitrarias.

18.1 HERRAMIENTAS USADAS PARA DEFINIR EL AMBIENTE

El modelo del ambiente consta de tres componentes:

1. Declaración de propósitos
2. Diagrama de contexto
3. Lista de acontecimientos

Cada uno se discute a continuación.

18.1.1 La declaración de propósitos

El primer componente del modelo ambiental es una declaración textual breve y concisa del *propósito* del sistema, dirigida al nivel administrativo superior, la administración de los usuarios, y otros que no están directamente involucrados con el desarrollo del sistema.

El siguiente es un ejemplo de una declaración de propósitos típica:

El propósito del Sistema de Procesamiento de Libros Ajax es manejar todos los detalles de los pedidos de libros de los clientes, además del envío, facturación y cobro retroactivo a clientes con facturas vencidas. La información acerca de los pedidos de libros debe estar disponible para otros sistemas, tales como mercadeo, ventas y contabilidad.

La declaración de propósitos puede constar de una, dos o varias frases. Sin embargo, jamás debe llegar a más de un párrafo, ya que *la intención no es proporcionar una descripción completa y detallada del sistema.* Tal esfuerzo iría en contra del objetivo: el propósito del resto del modelo ambiental y del modelo de comportamiento es dar todos los detalles.

Como resultado, la declaración de propósitos será deliberadamente vaga en cuanto a muchos detalles. En el ejemplo anterior se podrían hacer preguntas como:

- ¿Exactamente qué *tipo* de información proporciona a contabilidad, ventas y mercadeo el sistema de pedidos de libros?
- ¿Cómo determina el sistema de pedido de libros si un cliente tiene crédito? ¿Lo determina el sistema mismo o por medio del departamento de contabilidad?
- ¿Cómo se entera el sistema sobre nuevos libros que se han publicado y están disponibles para la venta?

Estas preguntas detalladas sólo pueden responderse viendo el modelo del comportamiento que se discute en los Capítulos 19 y 20.

Aunque el documento de declaración de propósitos no responde las preguntas detalladas sobre el comportamiento, generalmente basta con responder a una serie de preguntas de alto nivel:

- ¿Es responsable el sistema de pedido de libros de las actividades de nómina? *No;* prácticamente cualquiera que lea lo anterior estará de acuerdo en que la nómina queda fuera de la perspectiva del sistema y probablemente esté incluida en el sistema de contabilidad.
- ¿Es responsable el sistema de pedido de libros de enviar facturas a los clientes que piden libros? *Sí;* la declaración de propósitos así lo afirma. Podría imaginarse esto como tema de debate entre el departamento de pedidos de libros y el de contabilidad. Por ello es apropiado que se mencione en la declaración de propósitos.
- ¿Es responsable el sistema de pedido de libros del control de inventarios, es decir, de determinar cuándo volver a surtir libros que están a punto de acabarse? *No.* La declaración de propósitos no hace tal afirmación. Es muy probable que el control de inventarios sea uno de muchos otros sistemas (o departamentos) que usen la información sobre pedido de libros que produce el sistema de pedido.

Muchos analistas sienten también que la declaración de propósitos debe resumir los beneficios tangibles y cuantificables que se logren con el nuevo sistema; por ejemplo, "el propósito del sistema es reducir el tiempo que se requiere para procesar un pedido, de tres días a uno". Aunque esto puede ser muy útil en proyectos pequeños y muy definidos, no es fácil de lograr en proyectos más grandes. En su lugar suele requerirse un análisis de costo-beneficio.

18.1.2 El diagrama de contexto

La siguiente parte del modelo ambiental empieza a contestar algunas de las preguntas que surgen a raíz de la declaración de propósitos. El *diagrama de contexto* es un caso especial del diagrama de flujo de datos, en donde una sola burbuja representa todo el sistema. La Figura 18.4 muestra un diagrama de contexto de un sistema de pedido de libros. En los apéndices F y G se proporcionan ejemplos de diagramas de contexto para dos sistemas reales.

El diagrama de contexto enfatiza varias características importantes del sistema:

- Las personas, organizaciones y sistemas con los que se comunica el sistema. Se conocen como *terminadores.*
- Los datos que el sistema recibe del mundo exterior y que deben procesarse de alguna forma.

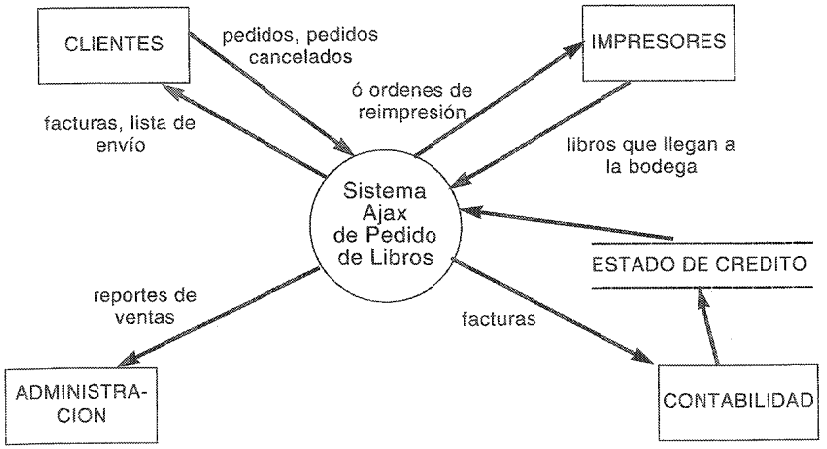


Figura 18.4: **Diagrama de contexto**

- Los datos que el sistema produce y que se envían al mundo exterior.
- Los almacenes de datos que el sistema comparte con los terminadores. Estos almacenes de datos se crean fuera del sistema para su uso, o bien son creados en él y usados fuera.
- La frontera entre el sistema y el resto del mundo.

Las técnicas para la construcción del diagrama de contexto se discuten en la Sección 18.2.

18.1.3 La Lista de acontecimientos

La lista de acontecimientos es una lista narrativa de los "estímulos" que ocurren en el mundo exterior a los cuales el sistema debe responder. En la Figura 18.5 se muestra una lista de acontecimientos para el sistema de pedido de libros.

1. Un cliente hace un pedido. (F)
2. Un cliente cancela un pedido. (F)
3. La administración pide un reporte de ventas. (T)
4. Llega un pedido de reimpresión de un libro a la bodega. (C)

Figura 18.5: **Lista de acontecimientos**

Observe que cada acontecimiento se etiqueta como F, T o C. Con ello se muestra si es de tipo de *flujo, temporal,* o de *control.* El orientado a flujos es el que se asocia con un flujo de datos; es decir, el sistema se da cuenta de que ha ocurrido

el acontecimiento cuando llega algún dato (o posiblemente varios). Como podrá imaginarse, esto corresponderá al flujo de datos en el diagrama de contexto.

Sin embargo, no todos los flujos de datos del diagrama de contexto necesariamente son acontecimientos de tipo flujo. Considere, por ejemplo, el diagrama de contexto parcial que se muestra en la Figura 18.6.

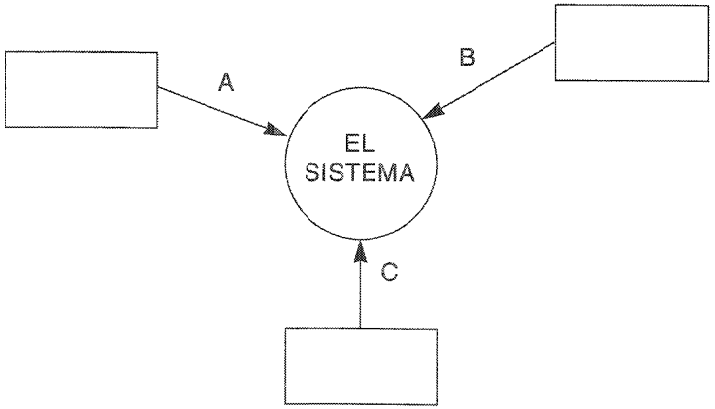


Figura 18.6: **Diagrama parcial de contexto**

A primera vista, uno pudiera verse tentado a concluir que los flujos de datos A, B y C son todos indicadores de acontecimientos separados y discretos. Sin embargo, pudiera resultar que sólo el flujo de datos A esté asociado con un acontecimiento (por ejemplo, al flujo de datos lo inicia el terminador). Para procesar un acontecimiento el sistema *explícitamente* podría pedir entradas a otros terminadores a lo largo de los flujos de datos B y C para obtener alguna respuesta.

Así que no necesariamente existe una correspondencia uno a uno entre los flujos de datos del diagrama de contexto y los acontecimientos de la lista de acontecimientos. En general, cada flujo de datos es un acontecimiento (o, más precisamente, la indicación de que ha ocurrido), o bien es requerido por el sistema para poder procesar un acontecimiento.

Además de los acontecimientos de tipo flujo, un sistema puede también tener acontecimientos *temporales*. Como su nombre implica, los acontecimientos temporales arrancan con la llegada de un momento dado en el tiempo. Algunos ejemplos de acontecimientos temporales pudieran ser:

- A las 9:00 de la mañana se requiere un reporte diario de todos los pedidos de libros.

- Las facturas deben generarse a las 3:00 PM.
- Se deben generar reportes administrativos una vez por hora.

Observe que los acontecimientos temporales no se *inician* con flujos de datos de entrada; podría imaginarse que el sistema tiene un reloj interno con el cual puede determinar el paso del tiempo. Sin embargo, tenga en mente también que un acontecimiento temporal podría requerir que el sistema solicite entradas de uno o más terminadores. Por ello, podrían asociarse uno o más flujos de datos con un acontecimiento temporal, aunque los flujos de datos, en sí, no representan el acontecimiento mismo.

Los *acontecimientos de control* deben considerarse un caso especial del acontecimiento temporal: un estímulo externo que ocurre en algún momento impredecible. A diferencia de un acontecimiento temporal normal, el acontecimiento de control no se asocia con el paso regular del tiempo, por lo que el sistema no puede anticiparlo utilizando un reloj interno. Y a diferencia de un acontecimiento de flujo normal, el de control no indica su presencia con el arribo de datos. Como lo muestra la Figura 18.7, un acontecimiento de control se asocia con un *flujo de control* en el diagrama de contexto.

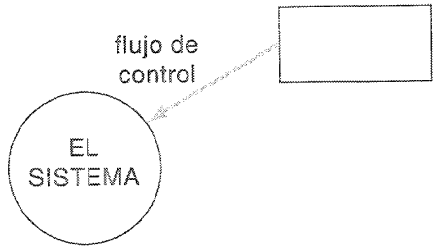


Figura 18.7: **Flujo de control asociado con un acontecimiento de control**

El flujo de control puede considerarse como un flujo de datos binario: está encendido o apagado, y puede cambiar de un estado al otro en cualquier momento, señalando así al sistema que se necesita tomar alguna acción *inmediata*. Los sistemas de información de negocios *no* suelen tener flujos de control en sus diagramas de contexto; el Sistema de Información de YOURDON Press que se describe en el Apéndice F, por ejemplo, no tiene. Pero los flujos de control son bastante comunes en los sistemas de tiempo real; como ejemplo vea el diagrama de contexto del sistema de control del elevador en el Apéndice G.

18.1.4 Componentes adicionales del modelo ambiental

En la mayor parte de los proyectos, la lista de acontecimientos, el diagrama de contexto y la declaración de propósito bastan. Sin embargo, puecen ser útiles dos componentes adicionales, dependiendo de la naturaleza y complejidad del sistema:

- El diccionario de datos inicial, que define todos los flujos y almacenes externos.
- El modelo de entidad-relación de los almacenes externos

Incluso un sistema mediano comúnmente tendrá unas cuantas docenas de flujos de datos de entrada y salida; un sistema grande pudiera tener literalmente cientos. Aunque estos flujos de datos finalmente se definirán con gran detalle en el modelo de comportamiento (que se discute en el Capítulo 20), podría ser útil comenzar la construcción del diccionario de datos *ahora*. Esto puede ser importante si las interfaces entre el sistema y los diversos terminadores están sujetas a cambios y a negociación; entre más pronto se comience a definir formalmente estas interfaces (definiendo la composición y significado de los almacenes), más pronto se podrán finalizar.

De manera similar, se puede construir un diagrama de entidad-relación de los almacenes externos (si hay). Esto puede ayudar a dejar al descubierto las relaciones entre almacenes que de otra manera no serían evidentes hasta que se desarrollara el modelo de comportamiento. Al concentrarse en estas relaciones en esta fase temprana se tiene forma de volver a verificar las interacciones entre los terminadores (que típicamente incluyen a los usuarios finales del sistema) y el sistema mismo.

18.2 CONSTRUCCION DEL MODELO AMBIENTAL

La discusión anterior probablemente hace que el modelo ambiental parezca simple y directo: después de todo, existe sólo un proceso, unos cuantos flujos de datos y terminadores, una descripción narrativa breve del propósito del sistema y una lista de acontecimientos. A pesar de esto, a menudo resulta que el modelo ambiental requiere de una gran cantidad de trabajo; además, usualmente se desarrolla como una serie de refinamientos iterativos, con datos adicionales que se añaden y refinan.

Una razón importante por la cual muchos refinamientos y revisiones suelen ser necesarios es que normalmente *una sola persona no puede comprender la perspectiva completa del sistema como se definió inicialmente.* Si el proyecto involucra un nuevo sistema que reemplazará a uno existente, es posible hablar con los usuarios que actualmente llevan a cabo sus funciones. En cierto sentido, tienen la perspectiva de quienes desde dentro ven las cosas hacia afuera, como ilustra la Figura 18.8. Sin embargo, incluso en este caso, los diversos usuarios individuales internos generalmente sólo están familiarizados con una porción, y sus diversas opiniones a veces

entran en conflicto. Peor aún, las entrevistas iniciales con la comunidad usuaria tal vez omitan uno o más usuarios importantes cuyas interacciones con los terminadores externos se deben modelar.[1]

Figura 18.8: **La visión del usuario del sistema**

Es importante dedicar una buena cantidad de tiempo y energía al modelo ambiental, pues a menudo es el punto focal de juntas y presentaciones importantes al comienzo de la vida de un proyecto de desarrollo de sistemas. De hecho, a veces es la *única* parte del modelo global del sistema que muchos usuarios y administradores de alto nivel llegarán a ver (los únicos con el dinero necesario para continuar financiando el proyecto, y con el poder para cancelarlo). Después de construido y aprobado lo verá en los pizarrones de avisos, así que es importante que esté correcto.

18.2.1 Construcción del diagrama de contexto

El diagrama de contexto, como hemos visto, consiste en terminadores, flujos de datos y flujos de control, almacenes de datos y un solo proceso que representa a todo el sistema. Se analizan uno por uno a continuación.

La parte más fácil del diagrama de contexto es el proceso; como hemos visto, consiste en una sola burbuja. El nombre dentro del proceso suele ser el nombre del sistema completo o un acrónimo convenido. En las Figuras 18.9(a) y (b) se muestran ejemplos.

---

1 Tales usuarios pudieran no ser importantes en términos de jerarquía organizacional; pueden considerarse como humildes oficinistas, secretarias o administradores. No obstante, las *funciones* que realizan pudieran ser vitales, y ser crucial modelar con precisión las entradas que reciben del mundo exterior y las salidas que mandan. La razón de que el analista de sistemas a menudo olvide hablarles a estas personas es muy sencilla: un usuario de nivel superior (es decir, el jefe) le dirá al analista cor quién hablar. "No moleste a mi gente", le dirá, "están todos muy ocupados, por eso requerimos el nuevo sistema. Yo le diré todo lo que necesita saber sobre el sistema". Como se discutió en el Capítulo 3, tal vez no haya forma diplomática de evitar esto, pero es crucial revisar el modelo ambiental con cuidaco para asegurar que no falta nada.

Figura 18.9(a): **Nombre típico de proceso para un diagrama de contexto**

Figura 18.9(b): **Otro nombre típico de proceso**

Observe que, en el caso extremo, el nuevo sistema podría representar una organización completa; en este caso, el nombre del proceso típicamente sería el de la organización misma, como lo muestra la Figura 18.9(c).[2]

Figura 18.9(c): **Nombre de proceso que representa una organización completa**

Los terminadores, como hemos visto, se representan con rectángulos en el diagrama de contexto. Se comunican directamente con el sistema a través de flujos

2 Este es un escenario poco probable para un proyecto de desarrollo de sistemas típico, pero se da más y más a medida que se usan los diagramas de flujo de datos y las otras herramientas descritas en este libro para construir modelos de *empresa*. Esto se puede hacer sin pretender computarizar toda la empresa, simplemente para entender lo que ya existe, sobre todo los datos que la organización requiere para llevar a cabo su propósito. El tema de modelos de empresa se discute en *Information Engineering for the Practitioner*, de William Inmon (Englewood Cliffs, N.J.: Prentice-Hall, 1987), así como en Strategic *Data Base Modeling*, de James Martin (Englewood Cliffs, N.J.: Prentice-Hall, 1985).

de datos o de control, como muestra la Figura 18.10(a), o a través de almacenes externos, como se muestra en la Figura 18.10(b).

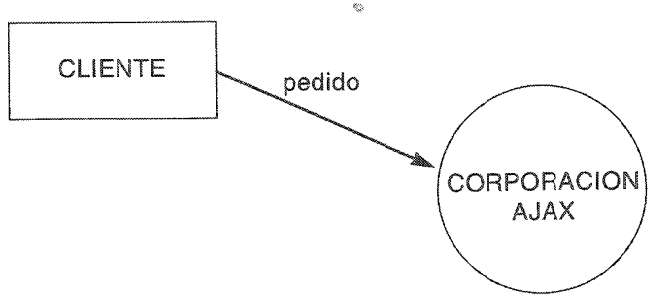


Figura 18.10(a): **Comunicación directa entre terminador y sistema**

Observe que los terminadores no se comunican directamente entre sí, por lo cual el diagrama de contexto de la Figura 18.11 no es correcto. En realidad, los terminadores sí se comunican entre sí pero, dado que por definición son *externos* al sistema, la naturaleza y contenido de las interacciones terminador-terminador son irrelevantes para el sistema. Si durante la discusión con los usuarios encuentra que es esencial saber cuándo, por qué o cómo se comunican entre sí, entonces *los ter-*

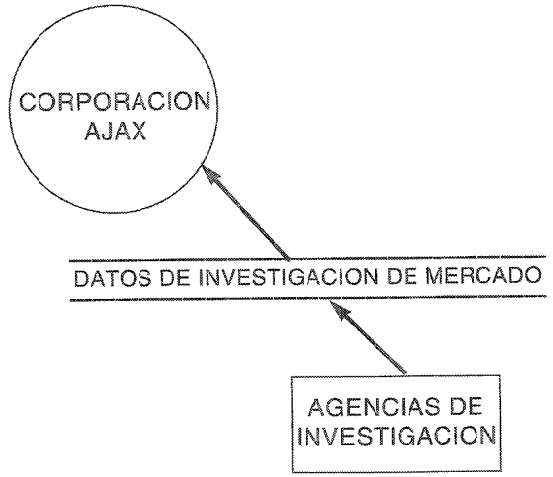


Figura 18.10(b): **Comunicación a través de un almacén externo**

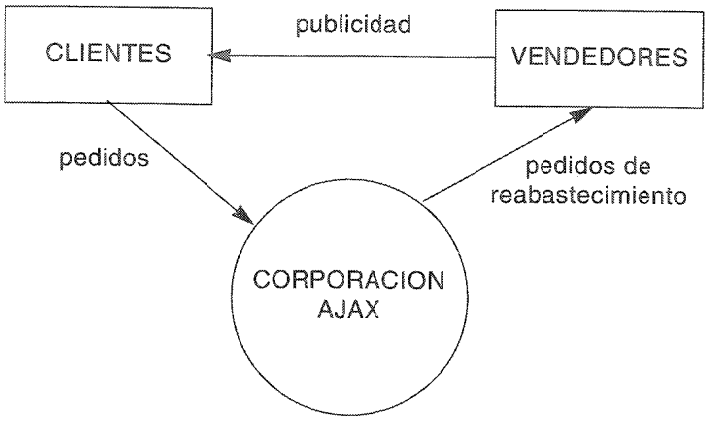


Figura 18.11: **Diagrama de contexto incorrecto**

minadores son parte del sistema, y deben incluirse dentro de la burbuja de proceso del diagrama de contexto.

Tres aclaraciones más acerca de los terminadores:

1. Algunos terminadores tienen un buen número de entradas y salidas. Para evitar un diagrama innecesariamente atiborrado conviene dibujar el terminador más de una vez, como muestra la Figura 18.12(a). Note que los terminadores duplicados se marcan con un asterisco; otra posibilidad es representar los terminadores repetidos con una diagonal, como muestra la Figura 18.12(b).

2. Cuando el terminador es una persona individual, generalmente es preferible indicar el *rol* que desempeña, más que su identidad; así, la Figura 18.13(a) se prefiere a la Figura 18.13(b). Hay dos razones para ello: primero, la persona que desempeña el papel puede cambiar con el tiempo, y es deseable que el diagrama de contexto permanezca estable y preciso incluso si hay cambios de personal. Segundo, una misma persona puede desempeñar varios papeles distintos en el sistema; en lugar de mostrar un terminador etiquetado "Juan Pérez" con varios flujos de entrada y salida no relacionados, es más significativo mostrar los diversos papeles que Juan Pérez desempeña, cada uno como terminador separado.

3. Dado que estamos interesados principalmente en desarrollar un modelo *esencial* del sistema, es importante distinguir entre *fuentes* y *manejadores* cuando se dibujan terminadores en el diagrama de contexto. Un manejador es un mecanismo, dispositivo o medio físico usado para transportar

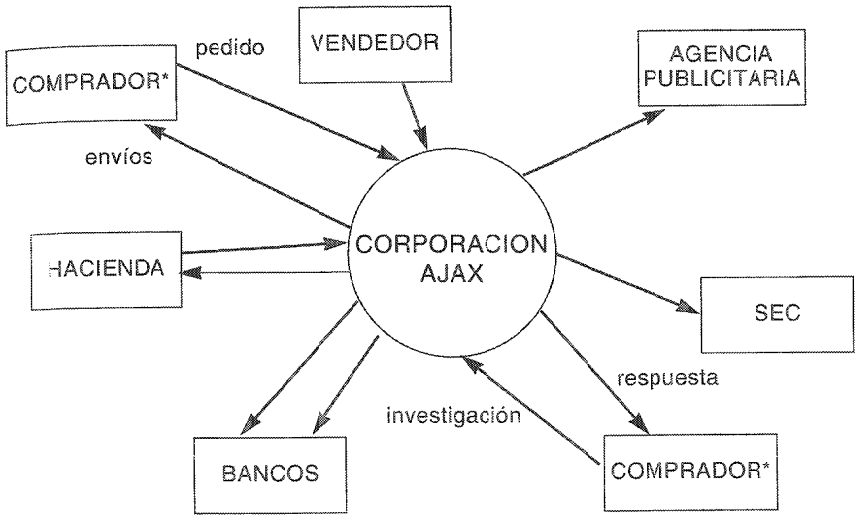


Figura 18.12(a): **Terminadores duplicados en el diagrama de contexto**

datos hacia dentro o fuera del sistema. Dado que a menudo dichos manejadores son familiares y visibles para los usuarios de la implantación actual de un sistema, existe la tendencia a mostrar al manejador, en lugar de la verdadera fuente de los datos. Sin embargo, puesto que el nuevo sistema generalmente tendrá opción de *cambiar la tecnología mediante la cual los datos se introducen y sacan del sistema,* el manejador no debe mostrarse. Así, el diagrama de contexto de la Figura 18.14(a) se prefiere al que se muestra en la Figura 18.14(b).

Como compromiso, sobre todo si el usuario insiste, se puede etiquetar un terminador para mostrar tanto la fuente verdadera como al manejador que introduce o saca los datos del sistema; esto se ilustra en la Figura 18.14(c).

Los *flujos* que aparecen en el diagrama de contexto modelan datos que entran y salen del sistema, además de las señales de control que recibe o genera. Los flujos de datos se incluyen en el diagrama de contexto si se ocupan para detectar un acontecimiento en el ambiente al que deba responder el sistema, o si se ocupan (como datos) para producir una respuesta. Los flujos de datos también pueden aparecer en el diagrama de contexto para ilustrar datos que estén siendo transportados entre terminadores por el sistema. Finalmente, los flujos de datos se muestran en el diagrama de contexto cuando el sistema produce datos para responder a un acontecimiento.

Como ya se ha hecho notar, el diagrama de contexto de un modelo esencial evita (hasta donde sea posible) mostrar los manejadores cercanos a la implantación que introducen y sacan datos de un sistema. Más aún, no se desea mostrar los mensajes y medios específicos de coordinación que el sistema y los terminadores pasan entre sí para indicar que están listos para las entradas o salidas. Se desea evitar diagramas de contexto como el de la Figura 18.15(a) pues incluye suposiciones sobre la implantación que pueden cambiar drásticamente cuando se construye el nuevo sistema.

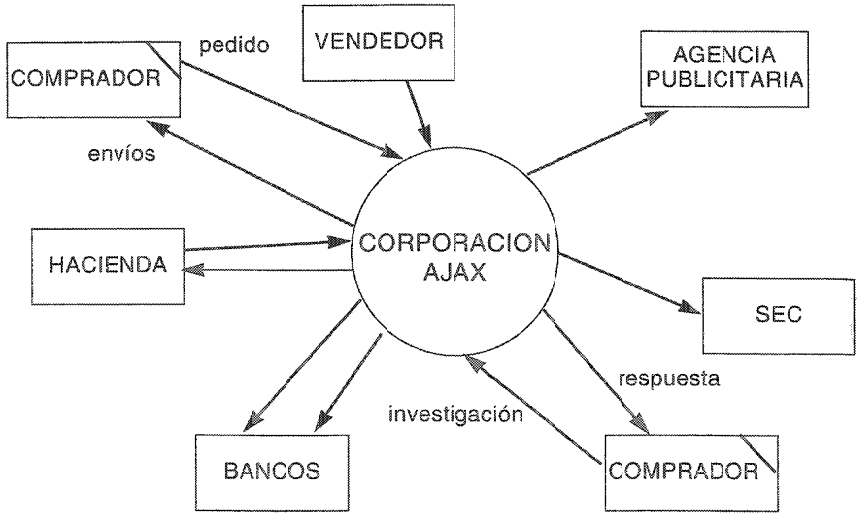


Figura 18.12(b): **Forma alternativa de mostrar terminadores duplicados**

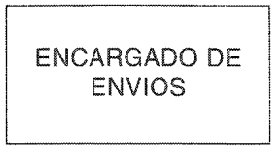


Figura 18.13(a): **Forma preferente de mostrar un terminador**

Figura 18.13(b): **Forma menos deseable de mostrar un terminador**

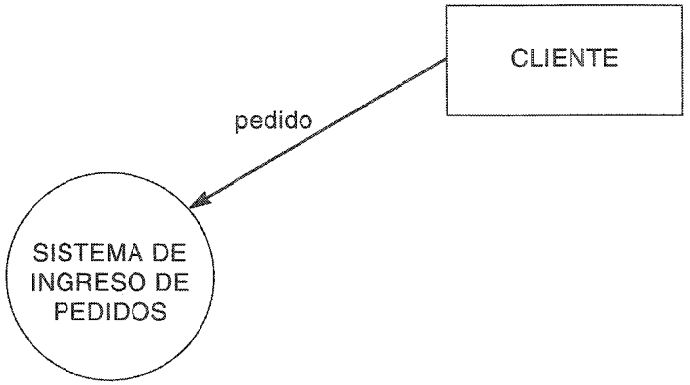


Figura 18.14(a): **Terminador que muestra la verdadera fuente de los datos**

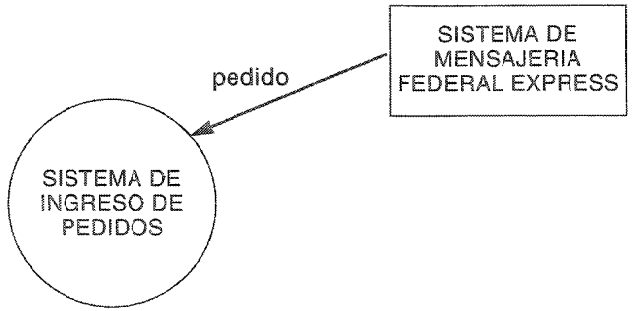


Figura 18.14(b): **Terminador que actúa como manejador**

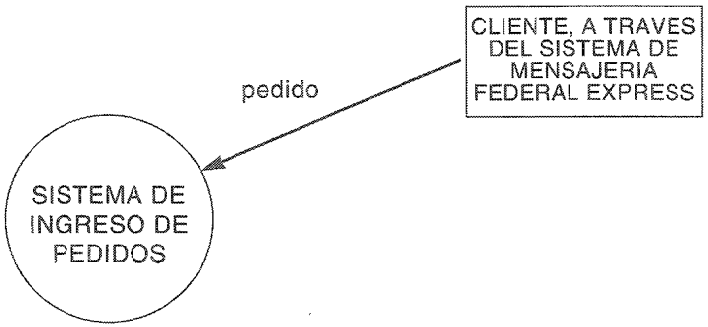


Figura 18.14(c): **Terminador que combina la fuente y el manejador**

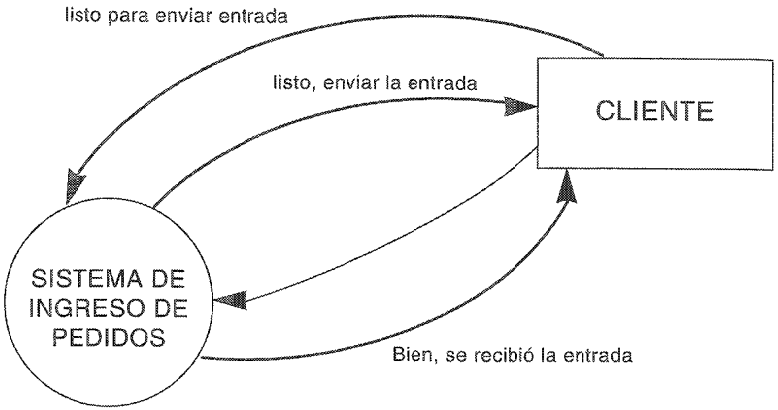


Figura 18.15(a): **Diagrama de contexto con mensajes innecesarios**

En lugar de eso, debe dibujarse el diagrama de contexto bajo el supuesto de que las entradas son causadas *e iniciadas por* los terminadores, y que las salidas son causadas *e iniciadas por* el sistema. Al evitar mensajes extraños y entradas y salidas orientadas a la implantación, se modela sólo el flujo *neto* de datos.

Sin embargo, habrá ocasiones en las que un terminador no inicie las entradas pues, aun con tecnología perfecta, el terminador no sabe que el sistema requiere sus entradas. Similarmente, hay ocasiones en las que el sistema no inicia la generación de salidas, debido a que no sabe que el terminador las necesita o desea. En ambos casos, el mensaje es una parte *esencial* del sistema, y debe mostrarse en el diagrama de contexto; la Figura 18.15(b) tiene un ejemplo. A veces resulta conveniente mostrar el mensaje y el correspondiente flujo de entrada o salida con un flujo de diálogo (una flecha de dos cabezas), como muestra la Figura 18.15(c).[3]

18.2.2 Construcción de la lista de acontecimientos

La lista de acontecimientos, como se vio, es un listado textual sencillo de los acontecimientos del ambiente a los cuales debe responder el sistema. Al crear la lista de acontecimientos se debe asegurar de distinguir entre un acontecimiento y un flujo relacionado con un acontecimiento. Por ejemplo, lo siguiente probablemente no sea un acontecimiento:

3 No es *necesario* usar un flujo de diálogo, pero sí vuelve más legible al diagrama de contexto al empaquetar las entradas y salidas asociadas para hacerlas visibles de inmediato al lector. Además, utilizar una flecha para mostrar el diálogo, en lugar de dos separadas, logra un diagrama de contexto menos atiborrado. Esto es importante en los grandes sistemas, donde pudiera haber incluso cien o más diferentes interacciones con terminadores externos.

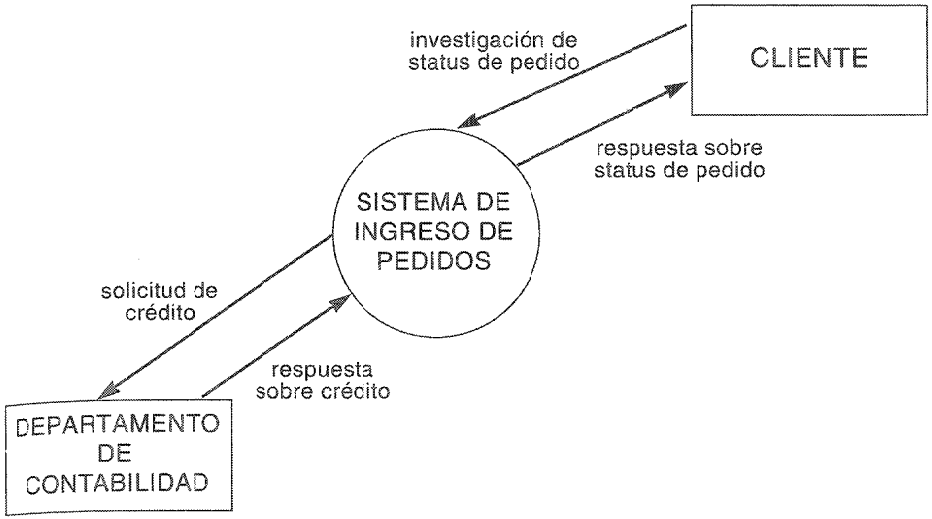


Figura 18.15(b): **Flujos de diálogo para mostrar mensajes esenciales**

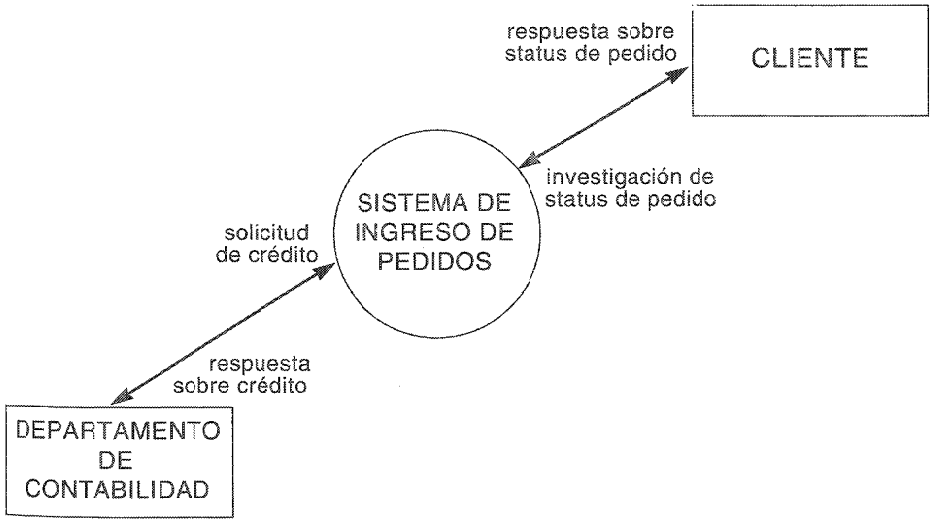


Figura 18.15(c): **Forma alterna de mostrar flujos de diálogo**

"El sistema recibe el pedido del cliente".

Más bien, probablemente sea el flujo de datos de entrada mediante el cual el sistema se da cuenta de que ha ocurrido el acontecimiento. Un nombre más apropiado para el acontecimiento sería:

"El cliente hace un pedido".

Esto pudiera parecer un ejercicio de semántica, pero no lo es. Si describimos el acontecimiento desde el punto de vista del sistema (por ejemplo, desde dentro, viendo hacia fuera), se podrían identificar erróneamente los flujos entrantes que no son acontecimientos por sí mismos, pero que se requieren para poder procesar algún otro acontecimiento. Por ello, siempre se trata de describir los acontecimientos desde el punto de vista del ambiente (es decir, desde fuera, viendo hacia dentro).

En la mayor parte de los casos, la manera más fácil de identificar los acontecimientos relevantes para un sistema es visualizarlo en acción: examinar cada terminador y preguntar qué efecto pueden tener sus acciones sobre el sistema. Esto usualmente se hace en conjunto con los usuarios del sistema desempeñando el papel de terminadores. Sin embargo, debe distinguirse cuidadosamente entre acontecimientos discretos que se han "empaquetado" accidentalmente como si fueran uno solo; esto sucede a menudo con acontecimientos de tipo flujo. Debe examinarse el acontecimiento candidato y preguntar si todas sus instancias involucran los mismos datos; si en algunas instancias están presentes los datos, y ausentes en otras, podría en realidad haber dos acontecimientos distintos. Por ejemplo, si se ve de cerca el acontecimiento "el cliente hace un pedido", se encontraría que algunas instancias incluyen el dato "identificación del vendedor" y otros no; y podría encontrarse que la respuesta del sistema es diferente cuando participa un vendedor que cuando no. Por ello, podría ser más apropiado tener dos acontecimientos separados: "el cliente hace un pedido" y "el vendedor tramita el pedido del cliente".

Además, tenga en mente que la lista de acontecimientos debe incluir no sólo las interacciones normales entre el sistema y sus terminadores sino también situaciones de falla. Dado que se está creando un modelo esencial, no hay que preocuparse por fallas del sistema; pero se deben tomar en cuenta posibles fallas o errores causadas por los terminadores. Como señalan Paul Ward y Stephen Mellor en *Structured Development for Real-Time Systems* (Nueva York: YOURDON Press, 1985),

> Puesto que los terminadores están por definición fuera de las fronteras del intento de construcción de sistema representado por el modelo, los realizadores no pueden modificar la tecnología de los terminadores para mejorar su confiabilidad. En lugar de ello, deben construir respuestas para los problemas de los terminadores en el modelo esencial del sistema. Un enfoque útil para modelar respuestas a los problemas de terminadores es construir una lista de acontecimientos "normales" y luego preguntar, para cada acontecimiento, "¿Necesita el sistema responder si este acontecimiento deja de ocurrir como se espera?"

Por ejemplo, nuestra lista de acontecimientos para el Sistema de Pedido de Libros Ajax (Figura 18.5) incluía un acontecimiento llamado "el pedido de reimpresión de libro llega a la bodega". Pero ¿qué tal si no llega a tiempo (por ejemplo, una semana después de la fecha prometida por el impresor)? ¿Qué debería hacer el sistema? Probablemente se necesitaría un acontecimiento adicional iniciado por el sistema para hacer que se comunique con el impresor y localice el origen del retraso.

18.2.3 ¿Qué se hace primero, el diagrama de contexto o la lista de acontecimientos?

Puede empezarse con la lista de acontecimientos *o* con el diagrama de contexto. En realidad no importa mientras finalmente se produzcan ambos componentes del modelo ambiental y *se revisen para asegurar que sean consistentes.*

Podría encontrarse también hablando con personas enteradas de todo lo que entra y sale del sistema; algunos usuarios pueden proporcionar esta información, o tal vez los programadores encargados del mantenimiento de la versión actual del sistema puedan saber algo al respecto. Esto ofrecerá las piezas del diagrama de contexto como punto de partida. Pueden discutirse entonces las transacciones que los usuarios mandan al sistema y la respuesta que esperan que dé. Con ello se puede crear una lista de acontecimientos a partir del diagrama de contexto.

Sin embargo, podría haber una situación en la que el diagrama de contexto no esté disponible. Esto es muy común, sobre todo al comienzo de algunos proyectos de desarrollo de sistemas: tal vez no sea fácil identificar los terminadores de los diversos flujos que entran y salen del sistema. *En este caso suele ser más práctico empezar con un DER que muestre los objetos y relaciones.* Los acontecimientos candidatos pueden encontrarse a continuación buscando las actividades u operaciones que ocasionan que se creen o eliminen relaciones. La creación de la lista de acontecimientos puede llevar entonces al desarrollo del diagrama de contexto; esto se ilustra en la Figura 18.16.

Por ejemplo, suponga que se han identificado los objetos CLIENTE y LIBRO en un sistema de publicaciones; el usuario podría también decir que existe una relación "pedidos" entre CLIENTE y LIBRO. Un acontecimiento probable, entonces, sería una acción que crea una instancia de la relación pedidos; otro acontecimiento sería una acción que elimine una instancia de una relación. Esto llevaría a identificar "El cliente pide un libro" y "El cliente cancela su pedido del libro" como acontecimientos en la lista de acontecimientos. No hace falta mucha investigación para darse cuenta de que "cliente" es un terminador para el sistema (mientras que "libro" no lo es); se podría entonces comenzar por dibujar el diagrama de contexto.

Cuando se termine con ambos componentes del modelo ambiental será posible confirmar lo siguiente:

- El sistema necesita cada flujo de entrada del diagrama de contexto para reconocer que ha ocurrido un acontecimiento; debe necesitarlo para producir una respuesta a un acontecimiento, o ambas cosas.
- Cada flujo de salida debe ser respuesta a un acontecimiento.
- Cada acontecimiento no temporal de la lista de acontecimientos debe tener entradas a partir de las cuales el sistema pueda detectarlo.
- Cada acontecimiento debe producir salidas inmediatas como respuesta o bien almacenar los datos que luego serán salidas (como respuesta o parte de una respuesta a algún otro acontecimiento), o debiera ocasionar un

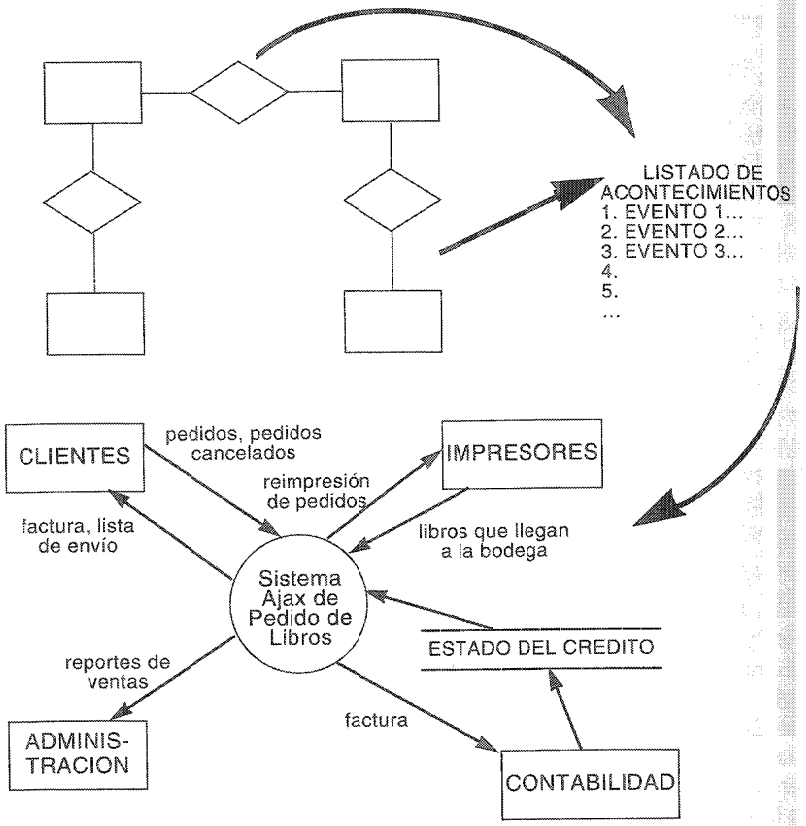


Figura 18.16: **Creación del diagrama de contexto a partir de un DER**

cambio en el estado del sistema (como indica el diagrama de transición de estados).

18.3 RESUMEN

La construcción de un modelo ambiental es lo primero y más importante en la construcción de un modelo completo de los requerimientos del usuario para un nuevo sistema. Hasta aquí parecería una tarea fácil; después de todo, el diagrama de contexto consiste sólo en una sola burbuja, y la lista de acontecimientos parece una simple lista de transacciones. Pero en un proyecto grande esto puede involucrar una gran cantidad de trabajo: la burbuja única del diagrama de contexto puede interactuar con docenas de terminadores externos y tener más de cien flujos de datos de entrada y salida. La lista de acontecimientos constituye un gran esfuerzo en los grandes sistemas; puede haber fácilmente cerca de cien acontecimientos que el sistema tiene que manejar, y todos necesitan ser identificados. Además, puede ser difícil encontrar una declaración sencilla de por qué debe existir el sistema.

Una vez construido el modelo ambiental debe ser revisado cuidadosamente por todos los representantes clave de los usuarios, además del equipo del proyecto. Entonces se estará preparado para comenzar a construir el modelo del comportamiento, es decir, el modelo del interior del sistema. Esto se discute en los Capítulos 19 y 20.

PREGUNTAS Y EJERCICIOS

1. ¿Cuáles son las tres cosas que define el modelo esencial?
2. ¿Qué tipo de acontecimientos debe modelar el modelo esencial?
3. ¿Cómo determina el analista la frontera entre el sistema y el ambiente?
4. ¿Cuál es la consecuencia probable de que el analista escoja un alcance demasiado pequeño para el proyecto?
5. ¿Cuál es la consecuencia probable de que el analista escoja un alcance demasiado amplio para el proyecto?
6. ¿Qué factores deben tomarse en cuenta cuando se escoge el alcance de un proyecto?
7. ¿Cómo afecta al alcance del proyecto el deseo del usuario de obtener una mayor participación en el mercado?
8. ¿Cómo afecta al alcance del proyecto la legislación impuesta por los diversos cuerpos gubernamentales?
9. Cómo afecta al alcance del proyecto el deseo del usuario de minimizar (o reducir) sus gastos operativos?

10. ¿Cómo afecta al alcance del proyecto el deseo del usuario de obtener ventajas estratégicas sobre la competencia?

11. Proyecto de investigación: Sobre un proyecto de su propia organización, ¿qué factor cree que afecte más en la elección del alcance? ¿Cree que el usuario, el analista y el equipo del proyecto estén conscientes y de acuerdo con ello?

12. En general, ¿cuáles cree que sean los posibles factores clave para los sistemas que se desarrollen en los años 90? Por ejemplo, ¿será más importante minimizar los gastos operativos que los cambios causados por la legislación gubernamental?

13. ¿Cuáles son los tres principales componentes del modelo ambiental?

14. ¿Aproximadamente de qué longitud debe ser un documento de declaración de propósitos?

15. ¿Cuáles cinco características de un sistema muestra un diagrama de contexto?

16. ¿Cuáles son los componentes de un diagrama de contexto?

17. ¿Qué es una lista de acontecimientos?

18. ¿Cuáles son los tres tipos de acontecimientos que debe modelar un diagrama de contexto?

19. ¿Qué relación hay entre flujos y acontecimientos en el diagrama de contexto?

20. ¿Qué es un acontecimiento temporal?

21. ¿Qué componentes adicionales pueden encontrarse en un modelo ambiental además del diagrama de contexto, la lista de acontecimientos y la declaración de propósitos?

22. ¿Por qué suele ser necesario hacer muchas revisiones y refinamientos del modelo ambiental?

23. ¿Por qué es importante asegurar que el modelo ambiental esté correcto?

24. ¿Qué tipo de nombre debiera ponerse dentro de una burbuja en un diagrama de contexto?

25. ¿Qué es un modelo de empresa?

26. ¿Cómo se comunican los terminadores con el sistema?

27. ¿Se comunican los terminadores entre sí en un modelo de sistema? ¿Por qué?

28. ¿Bajo qué condiciones se dibujaría un terminador más de una vez en un diagrama de contexto? ¿Cómo se mostraría?

29. Si el terminador es un individuo, ¿cómo se mostraría en el diagrama de contexto?

30. ¿Cómo puede estar seguro el analista de que ha identificado todos los terminadores en el diagrama de contexto?

31. ¿Qué es un manejador? ¿Qué diferencia hay entre una fuente y un manejador?

32. ¿Por qué deberían las fuentes, y no tanto los manejadores, aparecer en un diagrama de contexto?

33. ¿Qué debe hacer el analista si el usuario insiste en mostrar los manejadores en un diagrama de contexto?

34. ¿Bajo qué condiciones se muestran los flujos en un DFD?

35. ¿Por qué en general no deben mostrarse los mensajes y la sincronización en un diagrama de contexto?

36. ¿Qué significa el término flujo neto de datos?

37. ¿Bajo qué condiciones *no* inicia las entradas un terminador en un sistema?

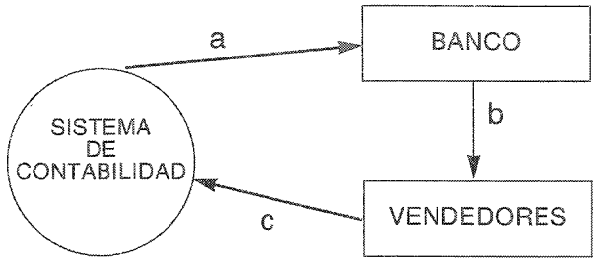
38. ¿Bajo qué condiciones el sistema *no* inicia las salidas al terminador?

39. ¿Qué debe desarrollarse primero, el diagrama de contexto o la lista de acontecimientos? ¿Por qué?

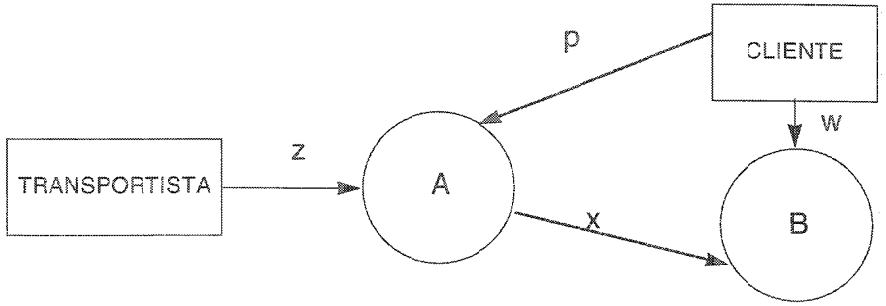
40. ¿Cuáles son las cuatro cosas que deben revisarse para asegurar que el modelo ambiental es correcto?

41. ¿Qué tiene mal el siguiente diagrama de contexto?

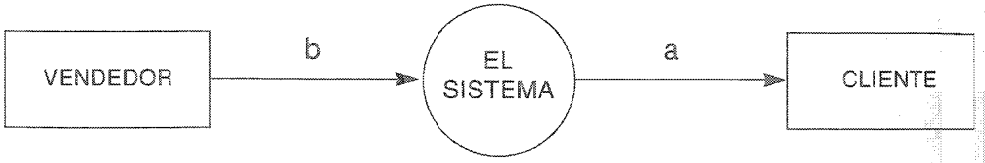
42. ¿Qué tiene mal el diagrama de contexto siguiente?



43. ¿Qué tiene mal el diagrama de contexto siguiente?



44. ¿Qué tiene mal el diagrama de contexto siguiente?



LISTA DE ACONTECIMIENTOS

1. El cliente necesita una "a"
2. El vendedor necesita la factura
3. El vendedor hace un envío
4. El cliente hace un pedido

19 CONSTRUCCION DE UN MODELO PRELIMINAR DE COMPORTAMIENTO

Las cosas siempre son mejores en sus comienzos.
Blaise Pascal, *Lettres Provinciales*
1656-1657, no. 4

En este capítulo se aprenderá:

1. Por qué es difícil un enfoque puramente descendente del modelo de comportamiento.
2. Cómo desarrollar un modelo preliminar de comportamiento usando la partición por acontecimientos.
3. Cómo desarrollar el DER inicial del modelo de datos.

En el capítulo anterior vimos cómo desarrollar el modelo ambiental para un sistema. Si se estuviera trabajando en un proyecto real ahora, se habría terminado el diagrama de contexto, la lista de acontecimientos y una declaración de propósitos. Además, se debe haber comenzado a construir el diccionario de datos, con por lo menos la definición de los datos que representan interfaces entre los terminadores externos y el sistema.

Nuestra labor ahora es comenzar a construir el modelo de *comportamiento* del sistema, es decir, el modelo del comportamiento final que el sistema debe tener para manejar con éxito el ambiente. Esto involucrará el desarrollo de un diagrama de flujo de datos y un diagrama de entidad-relación preliminares, además de la elaboración de las entradas iniciales del diccionario.

Básicamente, este enfoque implica dibujar el borrador del diagrama de flujo de datos, con un proceso (burbuja) para la respuesta del sistema ante cada acontecimiento que se identificó en la lista de acontecimientos. A continuación se dibujan almacenes en el borrador del DFD para modelar los datos que deben recordarse entre acontecimientos no sincronizados. Finalmente, se conectan los flujos de entrada y salida apropiados a las burbujas y se compara el conjunto de diagramas de flujo de datos contra el diagrama de contexto para asegurar la consistencia.

Una vez hecho esto se procede a un proceso de limpieza, descrito en el Capítulo 20, para producir un modelo bien organizado del proceso y un modelo de datos para presentarlo al usuario final. Este enfoque se llamó partición por acontecimientos en [McMenamin y Palmer, 1984].

Comenzamos por comparar este enfoque con el enfoque descendente clásico.

19.1 EL ENFOQUE CLASICO

El enfoque sugerido en este capítulo es sustancialmente diferente del enfoque descendente que se describe en textos clásicos tales como el de DeMarco [DeMarco, 1979], el de Gane [Gane y Sarson, 1979], y otros. El enfoque clásico supone que ya se dibujó el diagrama de contexto, pero supone que se procederá *directamente* de la burbuja única del diagrama de contexto a un DFD de nivel superior (conocido como figura 0), en donde cada burbuja representa un subsistema principal. Cada burbuja de la figura 0 se parte a continuación en figuras de nivel inferior, y cada burbuja de las figuras de nivel inferior se parte aún más, etc., hasta haber alcanzado el nivel de una burbuja "atómica" que no requiera de mayor descomposición. Esto lo ilustra la figura 19.1.

Aunque este enfoque descendente difiere del que se presenta en este libro, no me opongo a él ...*si funciona*. Sin embargo, tome en cuenta que muchos analistas encuentran los siguientes problemas cuando imtentan seguir un enfoque descendente:

- *Parálisis del análisis.* En muchos sistemas grandes y complejos, simplemente no existe pista alguna que guíe al analista para dibujar una figura 0 apropiada desde el diagrama de contexto. Por ello, el analista se queda sentado, viendo el diagrama de contexto, esperando la inspiración divina, o a alguien que le diga que ya no queda tiempo para el análisis, y que ya hace falta empezar a codificar.

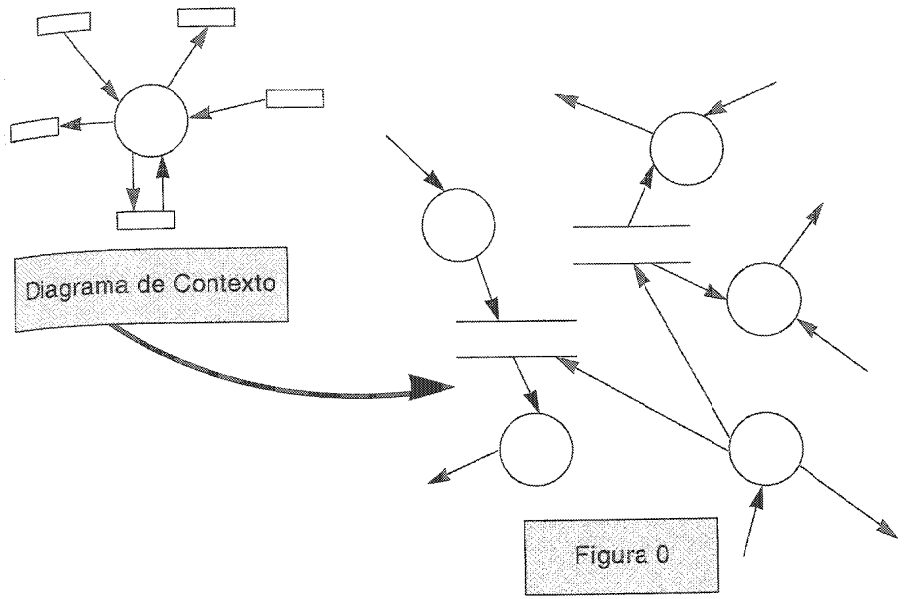


Figura 19.1: **El desarrollo descendente del modelo de comportamiento**

- *El fenómeno de los seis analistas.* En un sistema grande y complejo suele haber más de un analista viendo el diagrama de contexto. Para dividir el trabajo en forma equitativa y no estorbarse unos a otros, arbitrariamente crean una figura 0 con una burbuja para cada analista. Así, si hay seis analistas, la figura 0 constará de seis burbujas. Desde luego, ésta pudiera no ser la partición óptima para el sistema. ¿Qué ocurre, por ejemplo, si al mismo sistema lo especifican tres analistas? ¿O nueve? ¿O uno?

- *Una partición física arbitraria.* En muchos casos, un sistema nuevo se basa en uno existente, o representa la computarización de una organización existente. La partición de alto nivel del sistema actual (por ejemplo, las unidades organizacionales actuales o los sistemas de cómputo existentes) suele utilizarse como parámetro para la partición del nuevo. Así, si el sistema existente se representa con un Departamento de Compras y un Departamento de Control de Calidad, el nuevo sistema también tendrá un Subsistema de Compras y un Subsistema de Control de Calidad aun cuando tal vez no sean (y muchas veces no son) la mejor manera de partir el sistema (desde un punto de vista funcional).

El enfoque que se describe en este capítulo no es puramente descendente ni puramente ascendente. Es, en cierto sentido, un enfoque "medio"; después de desarrollar el DFD inicial, se requiere alguna nivelación *ascendente*, y luego podría necesitarse alguna partición *descendente*.

19.2 IDENTIFICACION DE RESPUESTAS A ACONTECIMIENTOS

El enfoque de partición por acontecimientos incluye los siguientes cuatro pasos:

1. Se dibuja una burbuja, o proceso, para cada acontecimiento de la lista.
2. La burbuja se nombra describiendo la respuesta que el sistema debe dar al acontecimiento asociado.
3. Se dibujan las entradas y salidas apropiadas de tal forma que la burbuja pueda dar la respuesta requerida, y se dibujan los almacenes, como sea apropiado, para la comunicación entre burbujas.
4. El borrador de DFD que resulta se compara con el diagrama de contexto y la lista de acontecimientos para asegurar que esté completo y sea consistente.

El primer paso es directo, de hecho casi mecánico en naturaleza. Si existen 25 acontecimientos en la lista, se deben dibujar 25 burbujas. Para tener una referencia conveniente, numere cada burbuja para hacer juego con el acontecimiento asociado: el acontecimiento 13 corresponde a la burbuja 13. (Más adelante, como veremos en el Capítulo 20, se renumeran apropiadamente las burbujas).

El segundo paso también es directo y mecánico: a cada burbuja se le da un nombre apropiado, basado en la respuesta requerida. Esto significa que se debe examinar el acontecimiento y preguntar "¿qué respuesta debe dar el sistema a este acontecimiento?" Recuerde, sin embargo, escoger nombres tan específicos como sea posible. Así, si el acontecimiento es CLIENTE REALIZA PAGO, un nombre de burbuja apropiado pudiera ser ACTUALIZAR CUENTAS POR COBRAR (si ésa es la única respuesta del sistema), en lugar de PROCESAR PAGO DE CLIENTE (que nada dice acerca de la naturaleza de la respuesta).

El tercer paso definitivamente no es mecánico, pero usualmente es bastante directo. Para cada burbuja dibujada, identifique las entradas que requiere para efectuar su trabajo. Identifique las salidas (si hay) que cada una produce e identifique los almacenes a los que que cada burbuja debe tener acceso. Esto normalmente se hace entrevistando al usuario o usuarios apropiados y concentrándose en cada acontecimiento y su burbuja asociada. Las preguntas son: "¿Qué necesita esta burbuja para hacer su trabajo?" y "¿Qué salidas genera?".

En muchos casos el acontecimiento está determinado por el flujo; esto significa que el sistema detecta la ocurrencia de un acontecimiento por la llegada de algún dato de un terminador externo. Obviamente, esto significa que el flujo de datos apropiado debe estar conectado al proceso requerido para responder a tal acontecimiento. Pero, como muestran las figuras 19.2(a) y (b), se pueden requerir entradas adicionales (de otros terminadores, y posiblemente de almacenes de datos) para que un proceso produzca su salida requerida.

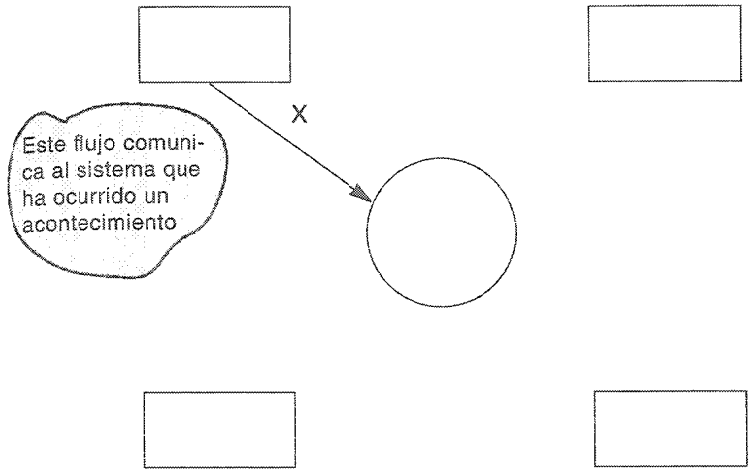


Figura 19.2(a): **Flujo de datos que señala la ocurrencia de un acontecimiento**

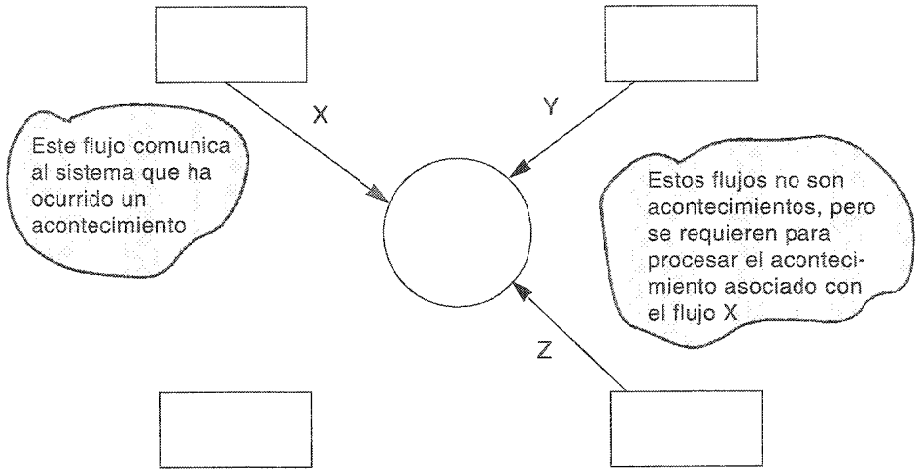


Figura 19.2(b): **Entradas adicionales requeridas para producir una respuesta**

De manera similar, como parte de la respuesta dibuje las salidas adecuadas producidas por el proceso. En muchos casos, esto implicará devolver salidas a los terminadores fuera del sistema; sin embargo, puede también involucrar salidas que se envían a los almacenes de datos, para ser usadas como entradas de otros procesos. Esto se ilustra en las figuras 19.3(a) y (b).

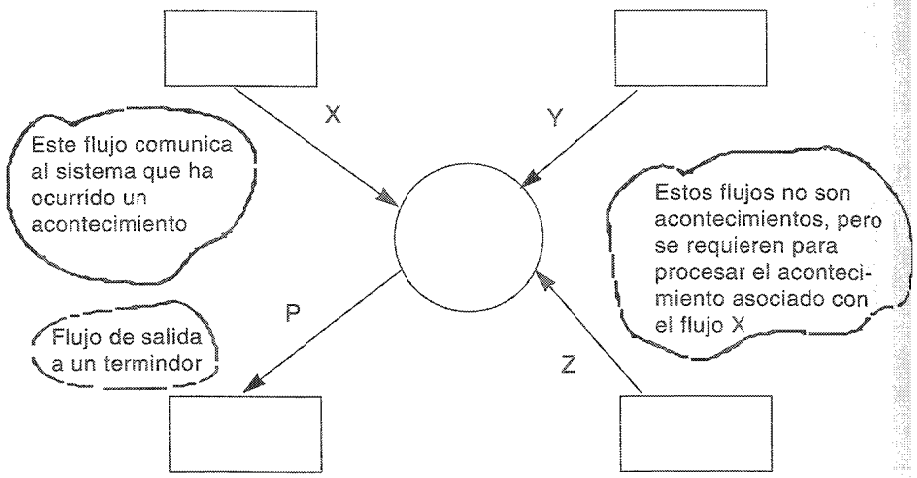


Figura 19.3(a): **Salida enviada desde un proceso a un terminador**

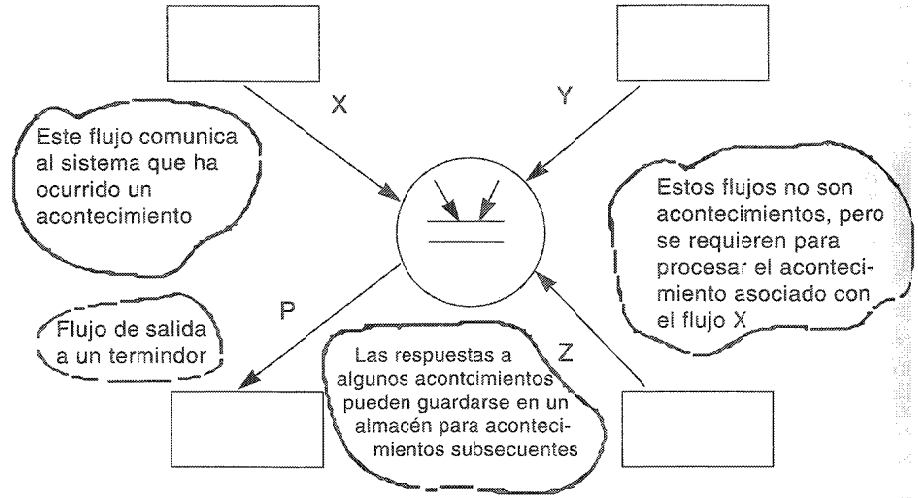


Figura 19.3(b): **Salida enviada de un proceso a un almacén**

Finalmente, el cuarto paso es una actividad de verificación de consistencia, similar a los pasos de balanceo del Capítulo 14. Verifique cada entrada del diagrama de contexto para ver si está asociada con alguna entrada de alguno de los procesos del DFD preliminar; y verifique también que cada salida producida por algún proceso en el DFD preliminar se envíe a un almacén o sea una salida externa incluida en el diagrama de contexto.

Existen dos casos especiales: (1) acontecimientos únicos que causan respuestas múltiples y, (2) acontecimientos múltiples que causan la misma respuesta. En el primer caso, un solo acontecimiento puede causar múltiples respuestas, *cada una* de las cuales se modela con su propia burbuja en el DFD preliminar. Esto se ilustra en la figura 19.4. Sólo es apropiado si todas las respuestas usan el mismo flujo de datos de entrada, y sólo si todas las respuestas son independientes entre sí. Ninguna salida de alguna parte de la respuesta global debe ser requerida como entrada por alguna otra parte de la respuesta global.

De manera inversa, habrá situaciones ocasionales en las que un proceso se asocia con más de un acontecimiento; esto se ilustra en la figura 19.5. Es válido y apropiado sólo si la respuesta de la burbuja es *idéntica* para los diversos acontecimientos, y sólo si los datos de entrada y salida son idénticos para las diversas respuestas a acontecimientos.

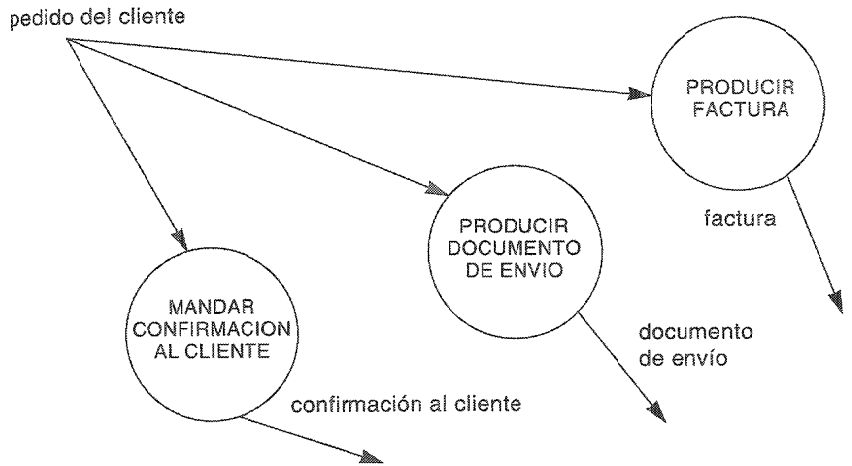


Figura 19.4: **Múltiples respuestas del mismo acontecimiento**

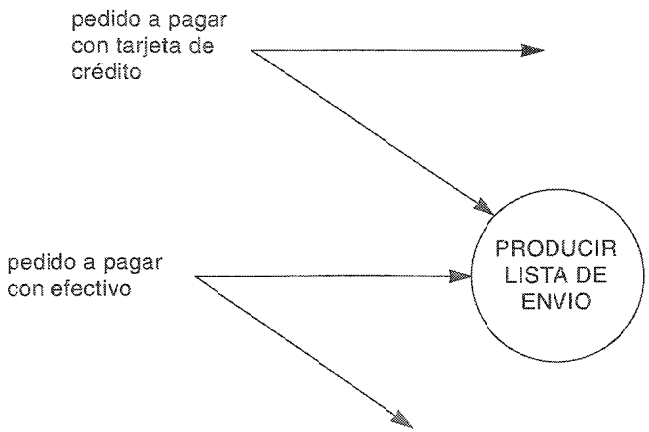


Figura 19.5: **Múltiples acontecimientos con la misma respuesta**

19.3 CONEXION DE LAS RESPUESTAS A ACONTECIMIENTOS

Observe que en los ejemplos anteriores ninguno de los procesos en el diagrama de flujo de datos preliminar está conectado con otro: *las burbujas no se comunican directamente con otras.* En vez de eso se comunican entre sí a través de otros almacenes de datos.

¿Por qué? Simplemente porque las burbujas del DFD preliminar representan *respuestas a un acontecimiento,* y los acontecimientos que ocurren en el ambiente externo son, en el caso general, no sincronizados. Es decir, no hay forma de garantizar que dos acontecimientos ocurrirán en el mismo instante, o con segundos de diferencia, o en algún otro período especificado de tiempo. Los acontecimientos ocurren en el ambiente externo cuando éste desea que sucedan.

Y, dado que:

- La respuesta a un acontecimiento puede requerir datos producidos por algún otro, y
- No hay forma de saber cuándo ocurrirán los acontecimientos, y
- Debe suponerse, en un modelo esencial, que cada proceso realizará su labor de manera infinitamente rápida, y
- Cada flujo de datos actúa como conducto que puede transmitir datos con rapidez infinita,

se sigue que la única forma de sincronizar múltiples acontecimientos interdependientes es mediante un almacén. Nótese que éste es un almacén *esencial;* que se necesita, no por retrasos asociados con la tecnología imperfecta, sino por consideraciones de tiempo en el ambiente.

Así que no se tendrá un diagrama de flujo de datos preliminar como el de la figura 19.6(a); ya cue los acontecimientos asociados no están sincronizados, sólo podría funcionar la figura 19.6(a) si hubiera escondido un almacén de datos diferido en el tiempo en alguno de los procesos o en el flujo de datos mismo. Por ello, es la forma correcta de mostrar la comunicación entre procesos es la figura 19.6(b).

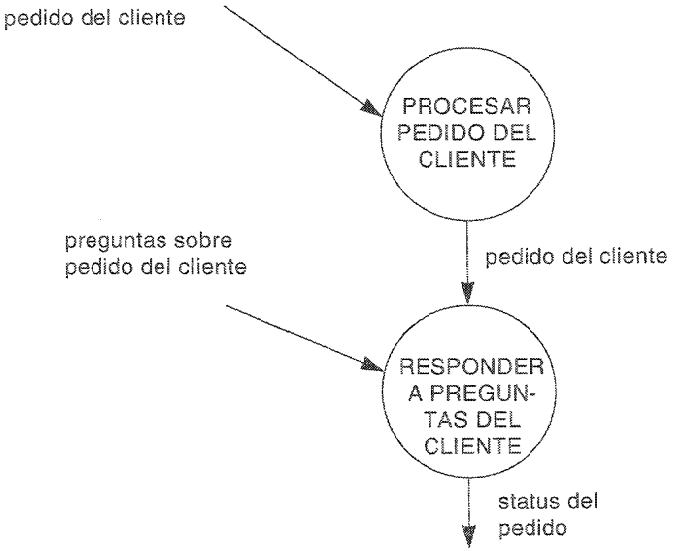


Figura 19.6(a): **Modelo no apropiado de la comunicación retardada entre procesos**

19.4 DESARROLLO DEL MODELO INICIAL DE DATOS

Como hemos visto, el procedimiento de dibujar el DFD inicial implica el dibujo de almacenes de datos entre procesos no sincronizados. En muchos casos su naturaleza será obvia, y los nombres pueden escogerse de la comprensión del tema del proyecto.

Mientras tanto, sin embargo, alguien debería haber comenzado a trabajar en la versión inicial del diagrama de entidad-relación *como actividad independiente, en paralelo con el desarrollo del DFD inicial.* Esto debe hacerse usando las técnicas del Capítulo 12.

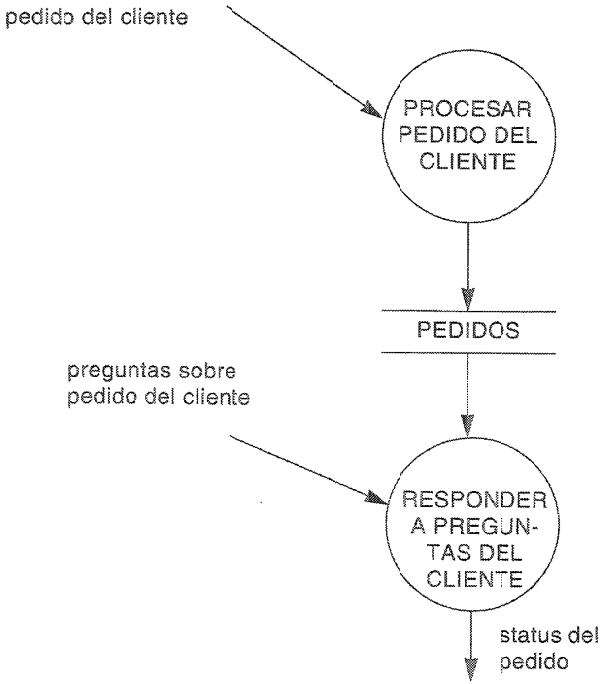


Figura 19.6(b): **Modelo apropiado de la comunicación retardada entre procesos**

Como el DER y el DFD se están desarrollando en paralelo, pueden usarse para revisarse entre sí. Los *almacenes* que se definieron tentativamente en el DFD preliminar pueden usarse para sugerir *objetos* en el DER preliminar; y los *objetos* que se identificaron tentativamente en el DER preliminar pueden usarse para ayudar a escoger *almacenes* apropiados en el DFD preliminar. Ningún modelo debe considerarse el dominante que controla al otro; cada uno es equivalente y puede proporcionar asistencia invaluable al otro.

Podría también encontrar que la *lista de acontecimientos* es tan útil para crear el DER inicial como para crear el DFD inicial. Sucederá que los *sustantivos* de la lista de acontecimientos sean objetos del DER; por ejemplo, si un acontecimiento es "Cliente hace pedido", inmediatamente se identificaría **cliente** y **pedido** como objetos tentativos. Similarmente, se puede usar una lista de acontecimientos para verificar el DER inicial: todos los tipos de objetos del DER deben corresponder con sustantivos de la lista de acontecimientos.

19.5 RESUMEN

Lo más importante a tomar en cuenta de este capítulo es que aún no se producirá un modelo de comportamiento listo para mostrarse al usuario. No está terminado; no es bonito; no es lo suficientemente sencillo ni bien organizado como para poderse entender en su totalidad. Puede ver un ejemplo de esto en el caso de estudio del Apéndice F.

¿Entonces qué es? ¿Cuál es el objeto de realizar los pasos descritos en la Sección 19.3? Simplemente es un comienzo, un *marco de referencia* sobre el cual basar el desarrollo de la versión final del modelo esencial.

No se preocupe en este momento por la organización del modelo de comportamiento, o su complejidad o claridad. Resista la tentación de reorganizar, empaquetar, descomponer o "recomponer" cualquiera de las burbujas del DFD preliminar. Lo único que debe importarle en este momento es *lo correcto* del modelo: ¿Tiene un proceso para cada acontecimiento? ¿Muestra las salidas y entradas necesarias para cada acontecimiento? ¿Muestra las conexiones necesarias entre acontecimientos?

Una vez establecido esto se puede comenzar a trabajar en la reorganización del modelo. Esto se discute con mayor detalle en el Capítulo 20.

REFERENCIAS

1. Tom DeMarco, *Structured Analysis and Systems Specification*. Englewood Cliffs, N.J.: Prentice-Hall, 1979.

2. Chris Gane y Trish Sarson, *Structured Systems Analysis: Tools and Techniques*. Englewood Cliffs, N.J.: Prentice-Hall, 1979.

3. Steve McMenamin y John Palmer, *Essential Systems Analysis*. Nueva York: YOURDON Press, 1984.

PREGUNTAS Y EJERCICIOS

1. ¿Qué es el modelo del comportamiento de un sistema? ¿Cuál es su propósito?

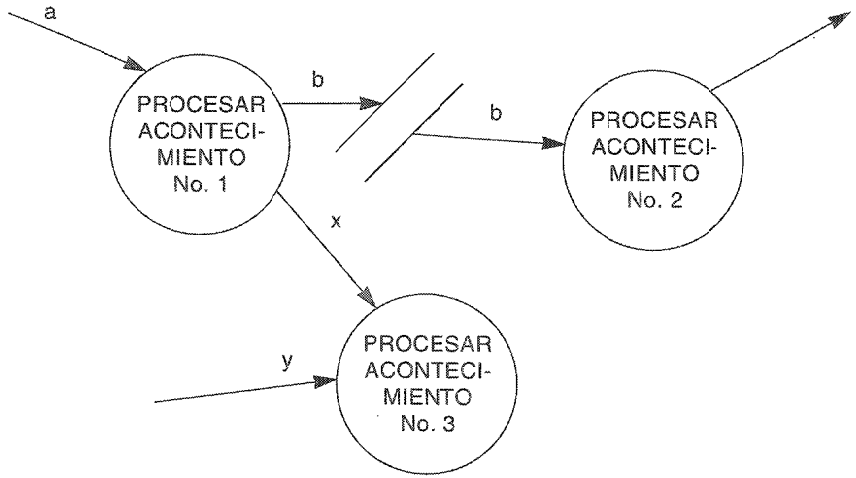
2. ¿Cuáles son los tres principales componentes del modelo *preliminar* de comportamiento?

3. ¿Cuál es el enfoque clásico para la construcción de un modelo del comportamiento? ¿Por qué se caracteriza como un enfoque descendente?

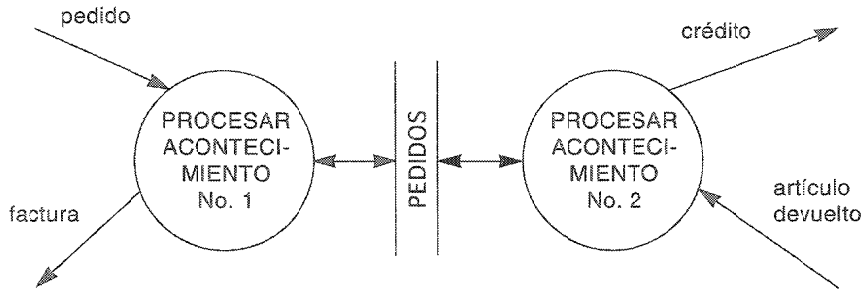
4. ¿Cuáles son los tres principales problemas que normalmente enfrentan los analistas cuando tratan de seguir el enfoque descendente clásico al construir un modelo del comportamiento de un sistema de información?

5. ¿Por qué cree que algunos analistas sufran parálisis o "bloqueo para escribir" cuando tratan de desarrollar el DFD de la figura 0 del diagrama de contexto?
6. ¿Por qué razón el modelo de comportamiento en algunos proyectos exhibe una partición física arbitraria?
7. ¿Qué significa el término partición por acontecimientos?
8. ¿Cuáles son los cuatro pasos de la partición por acontecimientos?
9. Si el analista ha descubierto 13 acontecimientos en un modelo ambiental, cuántos procesos (burbujas) debe haber en el primer borrador del modelo de comportamiento?
10. ¿Qué tipo de numeración se utiliza para las burbujas en el borrador del DFD del modelo de comportamiento?
11. ¿Qué parámetros se usan para nombrar las burbujas en el borrador del DFD del modelo de comportamiento?
12. ¿Cómo debe determinar el analista las entradas, salidas y almacenes que cada burbuja requiere en el borrador del DFD?
13. ¿Si un acontecimiento está determinado por el flujo, cuántos flujos de datos de entrada debe recibir la burbuja que lo procesa?
14. ¿Cuáles son las reglas de consistencia que el analista debe seguir cuando dibuja el borrador del DFD del modelo de comportamiento?
15. ¿Cómo debe dibujarse el borrador del DFD para el caso de un acontecimiento que produce múltiples respuestas?
16. ¿Bajo qué condiciones puede asociarse una sola burbuja del borrador del DFD con más de un acontecimiento?
17. En el borrador del DFD, ¿cómo se comunican las burbujas entre sí? Es decir, cómo se convierten las salidas producidas por una burbuja en entradas para otra?
18. ¿Cómo muestra el borrador del DFD la sincronización de acontecimientos múltiples, interdependientes y no sincronizados?
19. ¿Qué debe desarrollarse primero: el borrador del DFD o el borrador del modelo de datos (DER)? ¿Por qué?
20. Qué debe hacer el analista con el borrador del DFD y del DER después de terminarlos?

21. ¿Debe revisarse con el usuario el borrador del DFD cuando se ha terminado?
22. ¿Qué tiene mal el siguiente borrador de DFD?



23. ¿Qué tiene mal el siguiente borrador de DFD?



LISTA DE ACONTECIMIENTOS (del modelo ambiental)

1. El cliente pide un artículo
2. El cliente regresa un artículo
3. El cliente efectúa un pago

20 TERMINADO DEL MODELO DE COMPORTAMIENTO

Dennos las herramientas y terminaremos el trabajo.
Winston Churchill, transmisión de radio, 1941

En este capítulo se aprenderá:

1. Cómo nivelar *hacia arriba* un DFD inicial.
2. Cómo esconder los almacenes locales de datos.
3. Cuándo y cómo partir las burbujas iniciales del DFD *hacia abajo*.
4. Cómo completar el diccionario de datos inicial.
5. Cómo completar las especificaciones del proceso.
6. Cómo completar el modelo de datos.
7. Cómo completar el diagrama de transición de estados.

En el capítulo anterior presenté la estrategia para el desarrollo de una versión inicial del modelo de comportamiento. Sin embargo, debe ser evidente que este modelo no puede presentarse al usuario para su verificación. ¿Por qué no? Principalmente, porque es demasiado complicado. Como se vio en el Capítulo 19, el DFD preliminar tendrá un proceso para cada acontecimiento que se identificó en el modelo ambiental; de aquí que pudiera tener hasta 40 o 50 burbujas, o posiblemente más. De manera similar, la versión inicial del DER probablemente sea demasiado tosca como para revisarla con los usuarios; como se discutió en el Capítulo 12, se necesita un refinamiento para eliminar los objetos innecesarios y/o añadir nuevos.

Existe un segundo problema con el modelo: consiste principalmente en gráficos, con poco o ningún apoyo textual. Aunque el diagrama de flujo de datos y el diagrama de entidad-relación son excelentes vehículos para presentar una visión global del sistema al usuario, necesitan el apoyo de un diccionario de datos completo y un juego completo de especificaciones de proceso.

20.1 TERMINADO DEL MODELO DEL PROCESO

20.1.1 Nivelación del DFD

Lo primero es reorganizar el DFD que se desarrolló en el Capítulo 19. Como vimos, consiste en un solo nivel, con demasiadas burbujas. Por ello, se necesita una nivelación *ascendente* del DFD preliminar. Esto significa que se desea agrupar procesos *relacionados* en agregados con significado, cada uno de los cuales representará una burbuja de un diagrama de nivel superior. Esto se ilustra en la Figura 20.1.

Existen tres reglas que se debe tener en mente al hacer esto:

1. Cada agrupación de procesos deben involucrar respuestas relacionadas cercanamente (recuerde que cada burbuja del DFD preliminar se nombra acorde con la respuesta a un acontecimiento en la lista). Esto usualmente significa que los procesos manejan *datos* relacionados cercanamente.

2. Busque la oportunidad de esconder o "enterrar" datos almacenados que aparecen en el nivel inferior. Si ve un grupo de procesos en los DFD preliminares que se refieren a un almacén común, *y no hay otros procesos en el DFD preliminar que se refieran a este almacén,* entonces puede crear una burbuja de nivel superior para esconderlo. Esto se ilustra en la Figura 20.2 .

3. Tenga en mente que la persona que ve sus DFD, sea un usuario u otro analista, no querrá ver demasiado a la vez. Por ello, cree agregados o grupos del DFD preliminar que consistan en aproximadamente 7 más o menos 2 bloques de información, donde un proceso (y sus flujos relacionados) se consideren como un bloque.[1]

1 Este número aparentemente arbitrario (de siete más/menos dos) es una regla para controlar la complejidad en una variedad de situaciones de solución de problemas. Se basa en la obra de George Miller, quien por primera vez observó la dificultad de manejar trozos múltiples de información, en un trabajo clásico titulado: "The Magical Number Seven, Plus Minus Two: Some Limits on Our Capacity for Processing Information", publicado en *Psychological Review*, volumen 63 (1956) pp. 81-97.

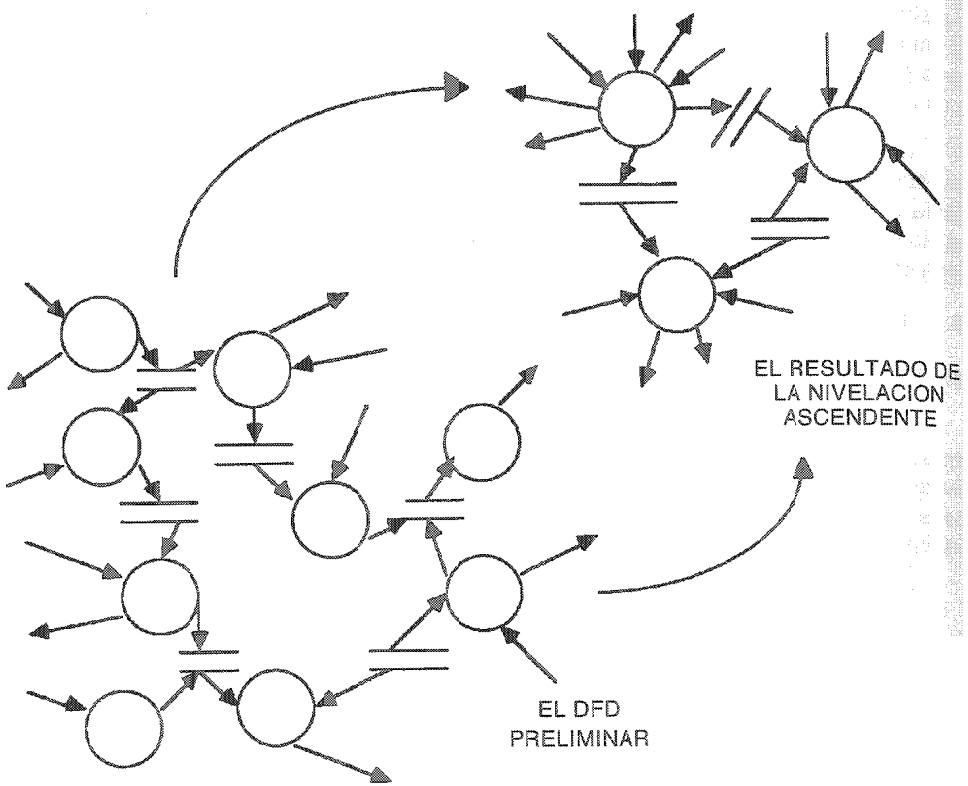


Figura 20.1: **Nivelación ascendente del DFD**

Desde luego, esto significa que tal vez se necesiten varios intentos de nivelación ascendente. Por ejemplo, si se empezara con un DFD preliminar que tuviera (por decir) 98 procesos y se organizara el diagrama en grupos de 7 burbujas (ignorando, por simplicidad, los almacenes), entonces se crearía un diagrama de nivel superior con 14 burbujas, cada una de las cuales representa una "abstracción" de siete de las de nivel inferior. Pero 14 burbujas son demasiadas para manejar y mostrar al usuario a la vez; así que probablemente se creará (como muestra la Figura 20.3) un diagrama de nivel superior con sólo dos burbujas.

Observe que este ejemplo tiene números completamente artificiales. No conduzca la actividad de nivelación procurando asegurar que cada diagrama tenga exactamente siete burbujas. De hecho, las dos primeras reglas mencionadas anteriormente: la agrupación de datos en torno a datos comunes y la búsqueda de oportunidades para esconder almacenes locales, deben ser el parámetro principal para la nivelación ascendente, y no alguna regla aritmética.

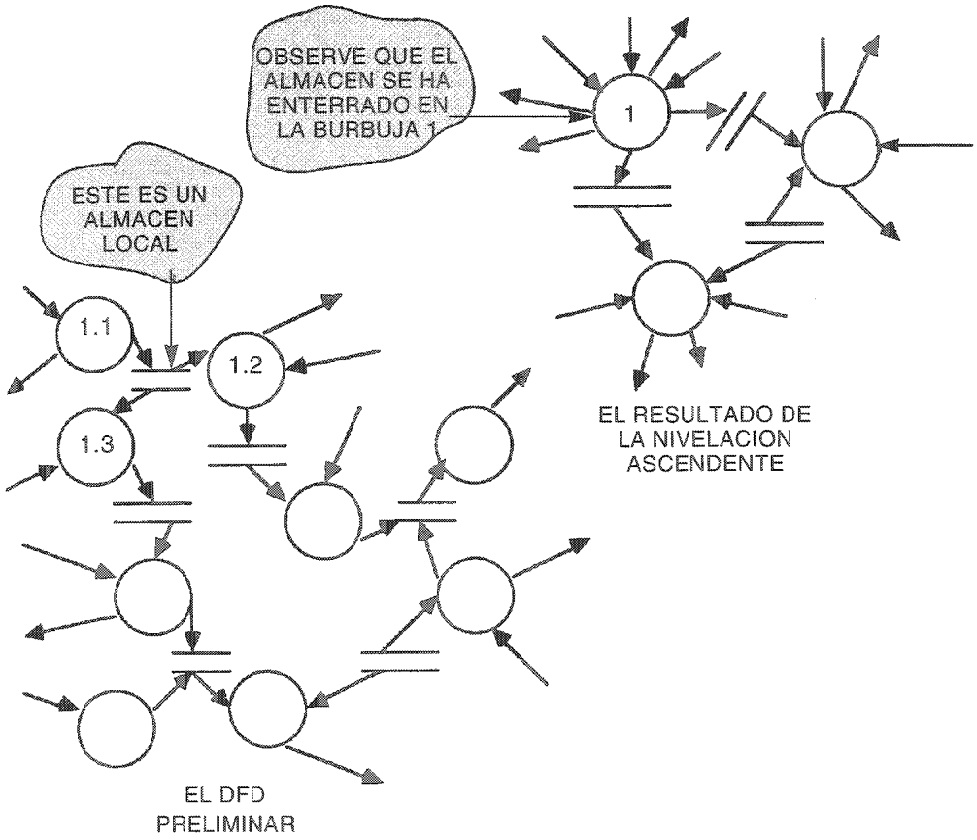


Figura 20.2: **Cómo se esconde un almacén local en el nivel superior**

Note también que podría requerirse nivelación *descendente*. Es decir, posiblemente los procesos identificados en el DFD resulten no ser procesos primitivos y requieran particiones descendentes en DFD de nivel inferior. Esto sólo significa que los procesos iniciales, cada uno de los cuales es responsable de producir la respuesta a un acontecimiento, pudieran resultar demasiado complejos para ser descritos adecuadamente en una especificación de proceso de una página. Muchas veces esto se volverá evidente tan pronto como vea el proceso con cuidado, o cuando pida al usuario una explicación de lo que la burbuja debe hacer. Si el usuario se pone a pensar un momento, respira hondo y dice: "Pues, es una larga historia, pero es algo así como..." ya tiene un fuerte indicio de que es muy probable que requiera dividir su burbuja preliminar.

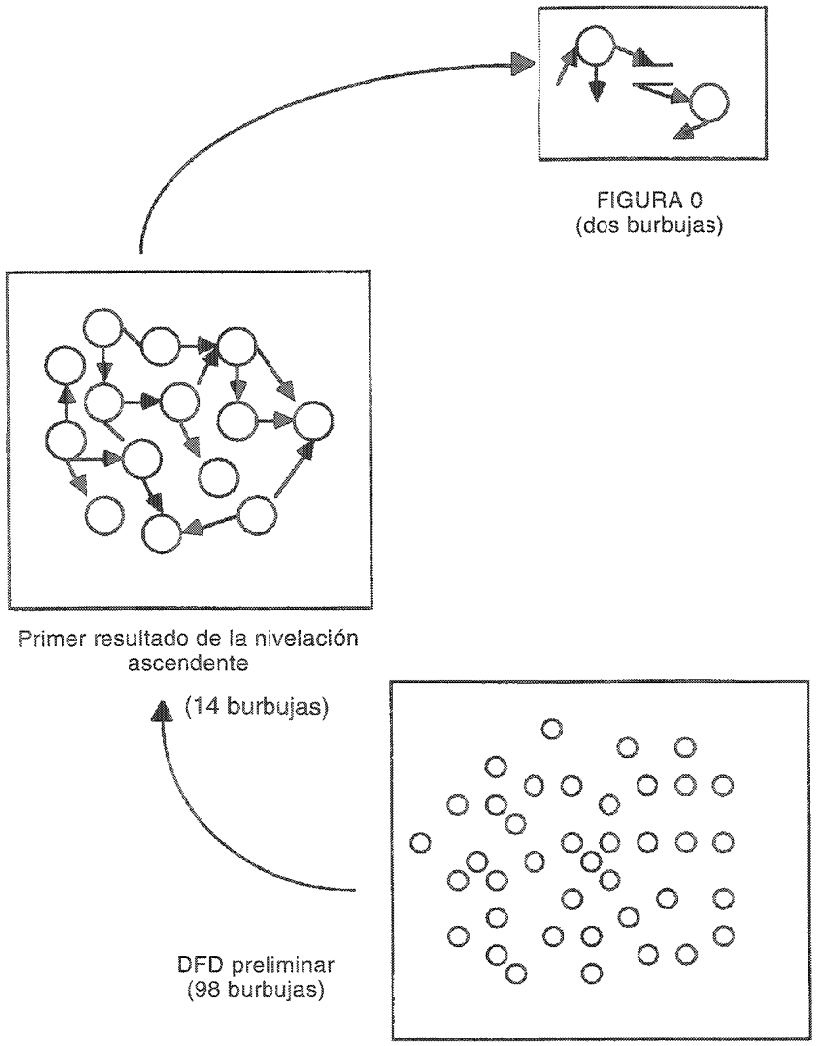


Figura 20.3: **Nivelación ascendente múltiple de un DFD**

En otros casos pudiera no ser evidente que la nivelación descendente se requiera hasta que de hecho se intente escribir la especificación del proceso; si encuentra que lleva tres páginas sobre la burbuja preliminar y que hay mucho más qué decir, de nuevo tiene un buen indicio de que se necesita la partición descendente.

He aquí algunas reglas para llevar a cabo la nivelación descendente:

- En algunos casos es apropiado un enfoque de descomposición funcional pura. Es decir, si encuentra una burbuja de proceso que realiza una función compleja, trate de identificar subfunciones, cada una de las cuales pueda ser hecha por una burbuja de nivel inferior. Por ejemplo, suponga que hubo un proceso llamado "Ajustar trayectoria del misil"; podría ser la burbuja responsable de manejar un acontecimiento temporal en un proyecto de guía de misiles en tiempo real. La función global de ajustar su trayectoria puede descomponerse en varias subfunciones:

  - Calcular variación de la coordenada x
  - Calcular variación de la coordenada y
  - Calcular variación de la coordenada z
  - Calcular nuevo factor de "retardo" atmosférico.
  - Calcular nueva velocidad del viento
  - Calcular impulso en la coordenada x
  - Calcular el impulso en la coordenada y
  - etc.

- En otros casos, los flujos de datos de entrada y salida proporcionarán la mejor guía para la nivelación descendente. Por ejemplo, suponga que se tuviera una burbuja como la de la Figura 20.4 . Es probable que se pudiera crear un DFD de nivel inferior con la forma general que se muestra en la Figura 20.5. Obviamente, podría requerirse más de una burbuja para

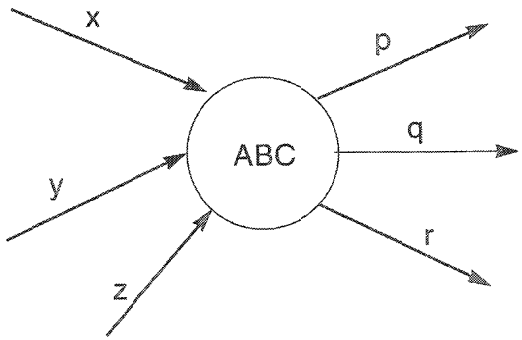


Figura 20.4: **Nivelación descendente de una burbuja compleja**

la combinación o agregado de datos individuales, pero la idea es la misma: *que los datos sean la guía.*

Tenga en mente al realizar esta actividad de nivelación ascendente y descendente, que el balanceo es importante. Es decir (como se discutió en el Capítulo 14), debe asegurar que las entradas y salidas netas que se muestran para la burbuja de alto nivel correspondan a las entradas y salidas netas que se muestran para el diagrama de nivel inferior. Para un ejemplo de esta actividad de nivelación ascendente, vea el caso de estudio del Sistema de Información de YOURDON Press en el Apéndice F. En este caso se comenzó con un DFD preliminar que contenía 40 burbujas; se requirió un nivel de nivelación ascendente, lo cual llevó a un DFD de Figura 0 con nueve burbujas.

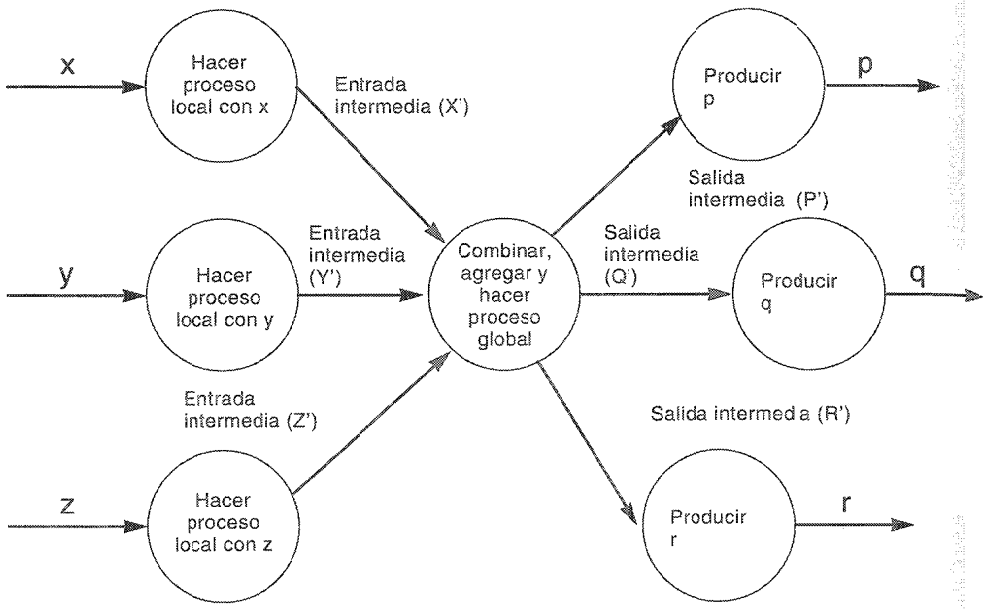


Figura 20.5: **El DFD de nivel inferior**

20.1.2 Cómo completar el diccionario de datos

Al desarrollar el DFD preliminar en el Capítulo 19, debió haberse comenzado a desarrollar el diccioanrio de datos; de hecho, es bastante común empezar el diccionario de datos cuando se está desarrollando el diagrama de contexto. Sin embargo, de ninguna manera estará completo aún. Comúnmente será necesario llenar la descripción del *significado* de cada dato; también sería apropiado dividir los datos complejos en elementos menores por claridad.

Al irse completando el diccionario de datos, también verifique que esté completo y sea consistente. Revise que el diccionario sea consistente internamente (es decir, que ninguna parte contradiga a otra), que esté balanceado con el diagrama de flujo de datos por niveles, el diagrama de entidad-relación y las especificaciones del proceso.

20.1.3 Cómo completar las especificaciones de proceso

Para cuando desarrolle el DFD preliminar, utilizando el enfoque de partición por niveles del Capítulo 19, es probable que no haya escrito especificaciones de proceso. Puede haber algunos cuantos casos en los que haya una especificación de proceso individual por algún interés en particular de parte suya o del usuario, pero su principal preocupación será simplemente organizar el DFD mismo.

De hecho, suele ser mala idea dedicar tiempo a la escritura de las especificaciones del proceso antes de terminar el DFD preliminar, porque el desarrollo inicial del DFD se ve sujeto a muchos cambios, correcciones y revisiones. Las burbujas pueden aparecer, desaparecer, reacomodarse y cambiar de nombre. Cuando el DFD preliminar empieza a estabilizarse, y cuando ha pasado la prueba de la nivelación ascendente (es decir, si la actividad no descubre fallas importantes en el modelo), *entonces* puede comenzar a escribir las especificaciones del proceso.

Esto a menudo será un esfuerzo largo y consumirá mucho tiempo, porque cada burbuja de nivel inferior en el conjunto de DFD requiere una especificación de proceso. Es posible que un grupo de dos o tres analistas dibujen unas cuantas docenas de DFD, pero puede ser necesario un mayor grupo de analistas para completar todas las especificaciones de proceso a tiempo.

Al irse completando las especificaciones de proceso debem balancearse y compararse con el diccionario de datos y el DER, usando las reglas que se presentaron en el Capítulo 14.

20.2 TERMINADO DEL MODELO DE DATOS

Como señalamos en el Capítulo 12, el DER se desarrolla de una manera similar a la descrita para el DFD: se desarrolla un DER tosco, y luego se refina y se mejora. Muchas de estas mejorías se pueden hacer simplemente asignando o atribuyendo datos a los tipos de objetos apropiados; esto usualmente ayudará a identificar nuevos tipos de objetos o tipos innecesarios.

Sin embargo, tenga en mente que muchas veces el DER se desarrolla casi al mismo tiempo que el DFD. Es muy común encontrar a alguien (o un pequeño grupo) dentro del mismo equipo que trabaja en el DER, mientras que otro (u otro grupo) tra-

baja en el DFD. O el equipo del proyecto desarrolla el DFD, mientras que el DER lo desarrolla un grupo centralizado de administración de datos de la organización de proceso electrónico de datos. En todo caso, si el DER y el DFD se desarrollan aproximadamente al mismo tiempo, entonces los conocimientos que se obtienen del DFD (por ejemplo, la existencia de almacenes, flujos de datos, etc.) puede usarse para refinar y revisar el DER.[2]

20.3 TERMINADO DEL DTE

Si su sistema tiene características de tiempo real, estará desarrollando un diagrama de transición de estados además del DFD y del diagrama de entidad-relación. El conocimiento detallado del comportamiento del sistema le ayudará a refinar este modelo. Como señalamos en el Capítulo 13, examine el diagrama de transición de estados inicial para encontrar los siguientes tipos comunes de errores:

- ¿Se han definido todos los estados?
- ¿Se puede llegar a todos los estados?
- ¿Se puede salir de todos los estados?
- En cada estado, ¿responde el sistema adecuadamente a todas las condiciones posibles?

20.4 RESUMEN

Hasta aquí, hemos llegado al final del modelo esencial. Si ha seguido todos los pasos de los capítulos 18, 19 y éste, debe tener un modelo completo, detallado, formal y riguroso de todo lo que el sistema debe hacer para llenar los requisitos del usuario. Contendrá lo siguiente:

- Diagrama de contexto
- Lista de acontecimientos
- Declaración de propósitos
- Conjunto completo de diagramas de flujo de datos por niveles
- Diagrama de entidad-relación completo y terminado
- Conjunto completo de diagramas de transición de estados
- Diccionario de datos completo (para la fase de análisis del proyecto)

2 Idealmente el mismo grupo debiera desarrollar los DER y DFD, trabajando en conjunto. Esto impide problemas de comunicación y tiende a asegurar que se les dé igual énfasis a ambos modelos. Desafortunadamente, rara vez sucede en la realidad.

- Conjunto completo de especificaciones, con una para cada proceso de nivel inferior

Suponiendo que ha revisado los componentes de las especificaciones para asegurarse que estén completos y sean consistentes, y suponiendo que el usuario ha revisado y aprobado el documento, ya debe haber terminado. Puede ponerle un bello listón rojo al paquete y entregarlo al equipo de diseño/programación cuya labor será *construir* el sistema. Luego, puede retirarse a la comodidad de su oficina hasta que surja el siguiente proyecto.

Pero, espere. Falta un paso. Todo lo que se ha desarrollado en este modelo esencial supone la existencia de tecnología perfecta, pero también supone que el usuario no tendrá restricciones de implantación que imponerle al sistema. La tecnología perfecta es producto de nuestra imaginación, pero podemos dejarle al equipo de implantación la decisión de cómo llegar a un compromiso razonable con la tecnología existente.

Suponer que el usuario ignorará todas las restricciones de implantación es también producto de nuestra imaginación, y es algo que debemos tratar antes de entregar al equipo de implantación la última versión de la especificación. Esta actividad final, que debe hacerse con la colaboración de usuarios, analistas *y algunos miembros del equipo de implantación, es el desarrollo del modelo de implantación* del usuario. Se discute en el siguiente capítulo.

PREGUNTAS Y EJERCICIOS

1. ¿Por qué no se puede presentar al usuario el primer borrador del modelo de comportamiento?
2. ¿Está completo el borrador del modelo de comportamiento? Si no, ¿qué elementos le faltan?
3. ¿Qué significa la nivelación ascendente en el contexto de este capítulo?
4. ¿Qué criterios debe seguir el analista para agrupar burbujas en un DFD?
5. ¿Cuáles son las tres reglas que el analista debe tener en mente al ir nivelando hacia arriba?
6. ¿Qué significado tiene el concepto de esconder datos almacenados en el contexto de este capítulo?
7. Cuántos niveles de DFD de mayor nivel deben crearse del borrador de un DFD? ¿Existe alguna fórmula matemática para dar una aproximación del número de niveles que se requieren?
8. ¿Bajo qué condiciones será necesaria una nivelación descendente en un DFD?

9. ¿Es posible que el analista realice nivelaciones ascendente y descendente en el DFD? ¿Por qué?

10. ¿Por qué tiene que completarse el diccionario de datos normalmente durante esta etapa del desarrollo del modelo de comportamiento?

11. ¿Qué tipo de verificación debe hacerse en el diccionario de datos durante este período del proyecto?

12. Por qué resulta ser mala idea que el analista invierta tiempo escribiendo especificaciones de proceso antes de completar el DFD preliminar? ¿Bajo qué condiciones pudiera tener sentido escribir por lo menos algunas especificaciones de proceso?

13. ¿Cuáles son los ocho componentes principales del modelo final de los requerimientos del usuario?

21 EL MODELO DE IMPLANTACION DEL USUARIO

Se debe observar una simplicidad espartana. Nada se hará meramente porque contribuye a la belleza, conveniencia, comodidad o prestigio.
De la Oficina del Oficial de Señales en Jefe, Ejército de los EUA, 29 de mayo de 1945.

En este capítulo se aprenderá:

1. Cómo escoger la frontera de automatización del sistema.
2. Cómo seleccionar dispositivos de entrada y salida para el usuario.
3. Cómo desarrollar formatos de entrada y salida.
4. Cómo diseñar las formas para el sistema.
5. Cómo desarrollar codificación para entradas del sistema.
6. Cómo identificar las actividades de apoyo manual.
7. Cómo describir las restricciones operacionales del sistema.

Al final del último capítulo habíamos terminado el desarrollo del modelo esencial de un sistema de información. Este modelo contiene una descripción completa de lo que el sistema *debe hacer* para satisfacer al usuario. Específicamente, el modelo esencial describe:

- La política, o lógica, esencial de las funciones que se requiere realizar.
- El contenido esencial de los datos que almacena el sistema, y que se mueven a través de él.
- El comportamiento esencial dependiente del tiempo que el sistema debe exhibir para manejar señales e interrupciones del ambiente exterior.

En el mejor caso (desde el punto de vista del analista y el equipo de implantación), con esta información sería suficiente para los diseñadores y programadores; simplemente se les daría el modelo esencial y se les permitiría escoger el mejor hardware, sistema operativo, sistema de administración de bases de datos y lenguaje de programación, dentro de las restricciones globales del proyecto en tiempo, dinero y recursos humanos. Sin embargo, no es tan sencillo: en prácticamente todos los proyectos de desarrollo de sistemas, el usuario insistirá en proporcionar alguna información adicional.

Esta información adicional involucra cuestiones de *implantación* que son suficientemente importantes, es decir, tienen el suficiente impacto sobre la capacidad del usuario para usar el sistema, que deben especificarse ya. El asunto de implantación más obvio de interés para el usuario es la frontera de automatización, es decir, cuáles partes del modelo esencial se van a implantar con la computadora y cuáles se van a realizar manualmente por personal de la organización. Tal vez el analista tenga su opinión al respecto, y el equipo diseñador/programador la suya, pero obviamente es el usuario quien tiene la última palabra.

Similarmente, el *formato* de las entradas y salidas del sistema (a veces conocido como interfaz humana) es de enorme interés para el usuario. A menudo, incluso, parece interesarle más que las funciones del sistema. Desde que los sistemas de cómputo empezaron a generar reportes en papel, los usuarios empezaron a preocuparse por la organización y distribución de la información en el reporte. ¿Dónde deben estar los encabezados? ¿Cómo organizar cada renglón para una lectura más fácil? ¿Debe haber un resumen o subtotal al final de cada página o sólo al final de cada reporte? Etc.

Con el advenimiento de los sistemas en línea en los años 70, esta cuestión se extendió para incluir el interés del usuario por el formato de las pantallas de entrada en la terminal de video. ¿Dónde deben aparecer los mensajes del sistema en la pantalla? ¿Qué tipo de mensajes de error deben aparecer? ¿De qué color deben ser? ¿De qué manera se podrá mover el usuario de una pantalla a otra?

Más recientemente, varias opciones y posibilidades más han aumentado la importancia de estas cuestiones de implantación:

- Los usuarios finales a menudo tienen la oportunidad de usar computadoras personales (PC) como parte de una red distribuida de computadoras (por ejemplo, se les da una PC que se conecta con la computadora principal de la organización). Esto lleva a una serie de preguntas: ¿Qué partes del modelo esencial se asignarán a la PC (bajo el control del usuario) y qué partes a la computadora principal? ¿Qué parte de los datos se asignará a la PC y cuál a la principal? ¿Qué formato tendrán las entradas que el usuario proporciona a la PC? ¿Qué actividades adicionales de apoyo hay que proporcionar para asegurar que el usuario no dañe inadvertidamente los datos almacenados en la PC o en la computadora principal?
- Los usuarios finales tienen cada vez más oportunidades hoy en día de escribir sus propios programas en lenguajes de cuarta generación tales como FOCUS, NOMAD e IDEAL, en la computadora principales o en dBASE-IV y Rbase-5000 en una PC. En la medida en que llegan a involucrarse con tales cuestiones de implantación, necesitan especificar los formatos de entradas y salidas del sistema. O más importante, decidir qué partes del sistema se implantarán utilizando lenguajes de cuarta generación y cuáles usando lenguajes convencionales de tercera generación.[1]
- En muchas situaciones actuales, el usuario y el analista podrían hacer un prototipo de porciones del sistema utilizando un lenguaje de cuarta generación o un paquete de generación de aplicaciones. El prototipo se haría porque el usuario no está seguro respecto a la política detallada que tarde o temprano tendrá que convertirse en especificaciones del proceso en el modelo esencial; pero más a menudo la actividad de hacer prototipos se dedica a la exploración y experimentación con formatos de entrada, diálogos en línea y diálogos de salida para pantallas o reportes.
- En muchas aplicaciones de negocios, otra opción para el usuario es la selección y compra de un paquete de software, es decir, un producto ya existente que puede comprarse o usarse bajo licencia. En este caso, las mismas cuestiones de implantación son de importancia para el usuario: ¿Qué parte de las funciones esenciales las implantará el paquete y cuáles tendrá que hacer el usuario (o tendrán que ser implantadas por el de-

---

1 Esto ilustra la necesidad de una buena comunicación entre los usuarios y el equipo de implantación, además de con los analistas de sistemas. Aunque los analistas pudieran estar bastante interesados en la utilización de lenguajes de cuarta generación, el equipo de implantación puede requerir investigar su desempeño. Los sistemas con grandes volúmenes de entradas y salidas pueden encontrar que los lenguajes de cuarta generación son demasiado ineficientes. Discutiremos esto más a fondo en el capítulo 23.

partamento de sistemas de información como sistema aparte)? ¿A cuáles datos esenciales se les dará mantenimiento con el paquete y a cuáles por el usuario? ¿Cuál será la forma y secuencia de las entradas que requiere el sistema comprado, y si esto será aceptable?[2]

Estas cuestiones deben tratarse como parte del *modelo de implantación del usuario,* que se crea aumentando, revisando o haciéndole anotaciones al modelo esencial, como veremos en las secciones siguientes de este capítulo. Sin embargo, recomiendo siempre conservar una copia del modelo esencial original intacto; esto permitirá explorar otros modelos de implantación en el futuro.

De manera general, el modelo de implantación del usuario cubre los siguientes cuatro puntos:

1. Distribución del modelo esencial entre personas y máquinas.
2. Detalles de la interacción humano-máquina.
3. Actividades manuales que se podrían requerir.
4. Restricciones operativas que el usuario desea imponer al sistema.

Cada uno se discute con más detalle a continuación.

21.1 DETERMINACION DE LA FRONTERA DE AUTOMATIZACION

Recuerde que el modelo del sistema con el que estamos trabajando identifica todas las actividades (funciones) y todos los datos esenciales. La cuestión ahora es: ¿Qué funciones y qué datos se manejarán manualmente, y cuáles se automatizarán? Aunque haya habido una elección tentativa preliminar de la frontera de automatización durante el estudio de factibilidad, no se debería considerar como congelada. De hecho, la frontera de automatización es casi irrelevante en el modelo esencial pues, aunque el usuario obviamente quiere que se desarrolle un sistema automatizado, también necesita una declaración bien hecha de los requerimientos de las funciones y datos que queden justo fuera de la frontera de automatización.

Hay tres casos extremos que mencionaremos brevemente:

- *Al usuario pudiera no importarle dónde esté la frontera de automatización.* Es poco probable que esto suceda, pero es una posibilidad teórica. Efectivamente, el usuario le está diciendo al equipo de implantación, "Ustedes díganme si tiene sentido que ciertas porciones del sistema sean manuales o automatizadas". Aparte del hecho de que normalmente el usuario se muestra preocupado, se espera que el analista produzca (como resultado adicional de su trabajo) un análisis revisado de costo-beneficio del proyecto completo. Esto usualmente requerirá de por lo menos una decisión preliminar acerca de cuáles partes del modelo esencial se automatizarán y cuáles serán manuales.

- *El usuario podría escoger un sistema totalmente automatizado.* Esta es una situación más común, sobre todo si el sistema que se desarrolla es reemplazo de uno actual y no se cambia la frontera de *automatización.* Así que las actividades manuales que el usuario realiza pudieran estar ya fuera de la frontera del sistema, que se representan en el diagrama de contexto por los terminadores con los que el sistema se comunica.

- *El usuario podría optar por un sistema completamente manual.* Esta es una opción fuera de lo común, sobre todo en esta era de automatización, porque el analista usualmente tiene interés en computarizar lo más posible. Sin embargo, puede suceder en situaciones donde las intenciones expresas del usuario sean *no* computarizar nada, sino simplemente reorganizar la forma en la que se desempeñan actualmente las actividades en una organización.

Normalmente estas opciones extremas no ocurren; basándose en interacciones entre el usuario, el analista y el equipo de implantación, se llegará a algún compromiso. Es decir, se automatizará parte de las actividades del modelo esencial y otras se identificarán como funciones manuales; de manera similar, algunos de los datos esenciales se identifican como candidatos obvios para computarización (y de esta forma quedan bajo el control de la administración de sistemas de información), y algunos se dejarán bajo el control del usuario. A menos de que éste tome una decisión inmediata y arbitraria al respecto, es bueno que las tres partes (el usuario, el analista y el equipo de implantación) exploren diversas opciones. Como ilustran las figuras 21.1(a), (b) y (c), pudiera haber diversas alternativas razonables para dibujar la frontera de automatización. Cada una tendrá diferente costo (que el equipo de implantación debe estimar, puesto que tiene conocimiento sobre las posibilidades de la tecnología de implantación) y diferentes ramificaciones organizacionales en el área del usuario.

No es labor ni del analista ni del equipo de implantación escoger la frontera de automatización, sino responsabilidad del usuario, y este libro no proporciona reglas para determinar qué tipo de elección es la mejor. Pero nótese que el modelo esencial sirve como herramienta útil para que el usuario y el equipo de implantación exploren diversas opciones. Una vez elegida la frontera de automatización, el analista pudiera darse el lujo de pensar en eliminar procesos y datos manuales (es decir, aquellas burbujas y almacenes que no se automatizarán). Pero esto generalmente no es verdad. En el caso más sencillo, se puede requerir regresar las actividades y los datos manuales a los terminadores que rodean al sistema, como muestran las figuras 21.2(a) y (b).

2 Aquí se está haciendo una suposición muy importante: que el modelo esencial debe desarrollarse primero, antes de evaluar el paquete comercial. Muchas organizaciones hacen justo lo contrario: primero evalúan el paquete y luego tratan de derivar un modelo de los requerimientos esenciales usando las características que ofrece.

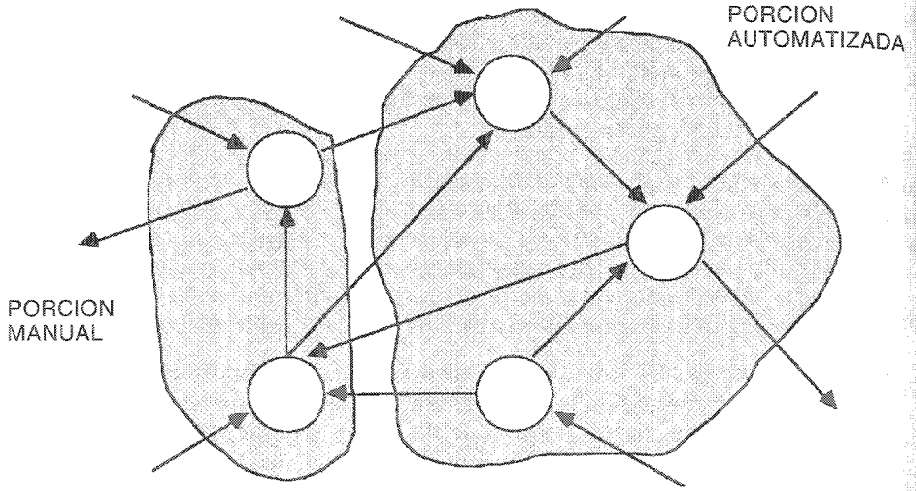


Figura 21.1(a): **Una posible elección de frontera automatizada**

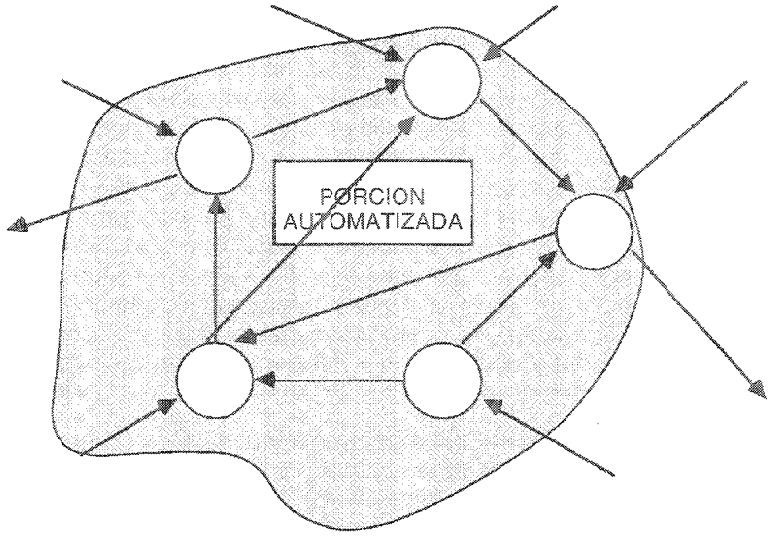


Figura 21.1(b): **Otra elección de frontera de automatización**

Pero, en el caso general, el analista debe reconocer que incluso las actividades manuales *son parte del nuevo sistema*. Por ello puede tener que escribir procedimientos para los usuarios de modo que sepan cómo llevar a cabo las funciones

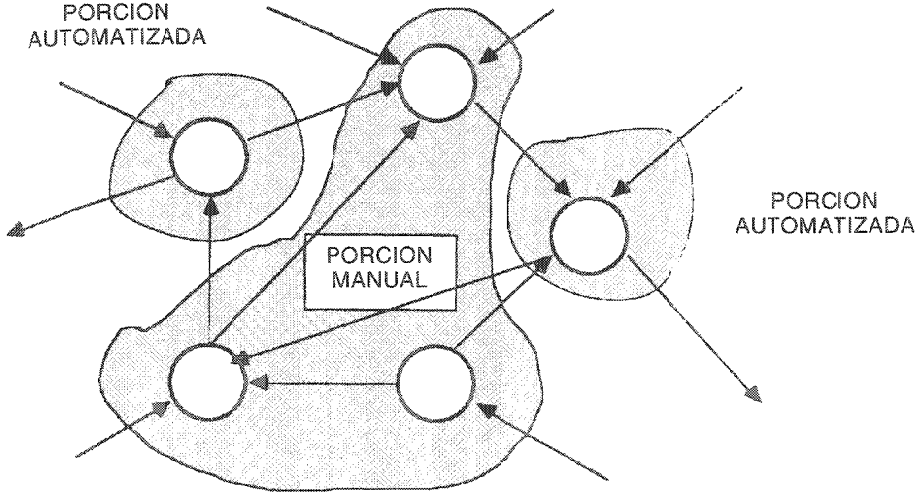


Figura 21.1(c): **Una tercera opción de frontera automatizada**

requeridas, además de proporcionar alguna guía sobre la organización de los almacenes que no se van a automatizar.[3] Nótese que este aspecto del modelo de implantación del usuario meramente requiere de anotaciones en el DFD y el DER para indicar las actividades manuales y las automatizadas.

Observe que una vez escogida la frontera de automatización pudiera ser importante considerar algunas cuestiones ambientales: nivel de ruido, radiación, iluminación, comodidad de la terminal de video y espacio de trabajo, etc. A menudo,el nuevo sistema perturbará el ambiente normal de trabajo del usuario (por ejemplo, ocasionará que se coloque una terminal en el escritorio de un usuario, donde antes jamás hubo).[4] O podría traer actividades de proceso de información a un ambiente donde nunca las hubo antes (por ejemplo, el área de producción de una fábrica).

---

3 Los procedimientos del usuario para los procesos manuales pueden basarse en la especificación del proceso. De hecho, en el caso más sencillo la especificación del proceso es el procedimiento del usuario; sin embargo, dado que las especificaciones del proceso se escribieron cuidadosamente para evitar cualquier prejuicio de implantación, tal vez sea necesario expandirlas o reescribirlas para servir de guía para los usuarios.

4 Piense un momento en el tipo de problemas que pueden surgir al simplemente poner una terminal en el escritorio de un usuario. Primero, tal vez no quepa: posiblemente el usuario necesita todo el espacio para otras cosas que esté haciendo. Segundo, puede ser que no haya suficientes tomas de corriente para la terminal de video, la impresora, el modem y otros periféricos. Tercero, el escritorio podría tener la altura adecuada para leer y escribir más no para teclear. Cuarto, la luz de la oficina puede molestar a tal grado que sea difícil leer la información de la pantalla. Quinto, el ruido del tecleo del usuario en la terminal puede resultar molesto a otros usuarios en la misma área.

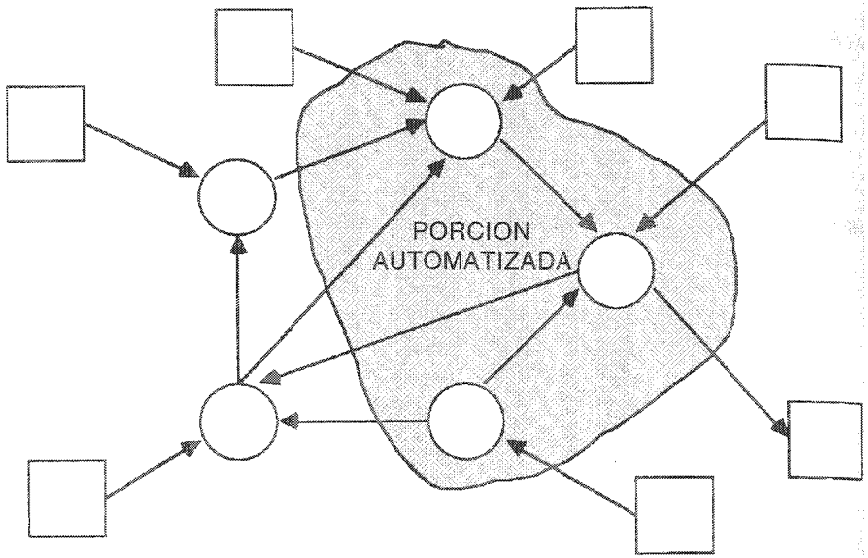


Figura 21.2(a): **Modelo esencial con frontera de automatización**

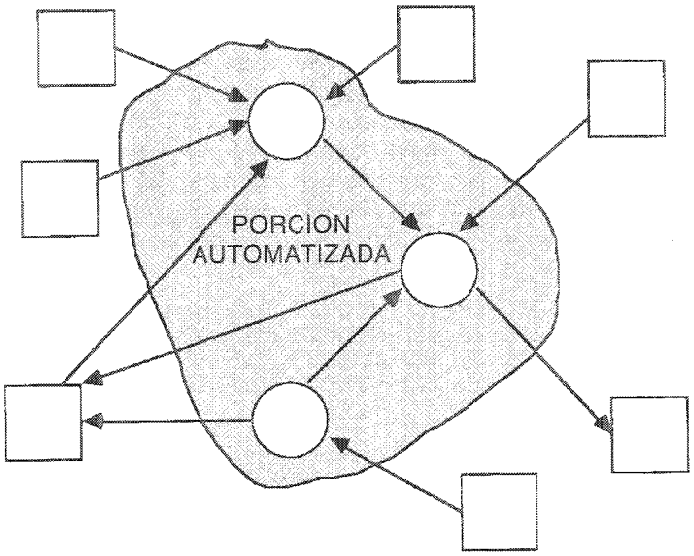


Figura 21.2(b): **Las actividades manuales se han integrado a los terminadores**

Por ello, es importante asegurarse haber estudiado estos factores humanos a fondo antes de tomar la determinación final respecto a la frontera. A veces el usuario tendrá bastante qué decir al respecto, pero si no ha tenido experiencia previa con sistemas de información quizá no pueda predecir el tipo de problemas que surgirán. Pida consejo a otros profesionales de sistemas que hayan desarrollado e instalado sistemas similares en condiciones ambientales similares.

Finalmente, note que una vez que se ha escogido la frontera de automatización, podría ser necesario aumentar el modelo esencial para mostrar cómo se enciende y apaga el sistema; el modelo esencial muestra sólo el comportamiento de estado estable del sistema, y supone que éste ha estado trabajando desde siempre, y que continuará trabajando para siempre. Así, podría requerirse incluir procesos adicionales (burbujas en el DFD), flujos de datos y almacenes que muestren las actividades de inicio y apagado del sistema, lo cual incluiría reportes a la administración (o a los usuarios o al departamento de operaciones) sobre las características operacionales del sistema.

21.2 DETERMINACION DE LA INTERFAZ HUMANA

Probablemente la actividad que consume más tiempo, y que más interese al usuario, es la especificación de la interfaz humana. Esto involucra cuatro asuntos relacionados:

1. La elección de los dispositivos de entrada y salida (por ejemplo, terminales de video, dispositivos de reconocimiento óptico de caracteres, tarjetas perforadas, etc., para las entradas, y reportes en papel, despliegues en pantalla o luces como salidas).
2. El formato de todas las entradas que fluyen desde los terminadores hasta el sistema.
3. El formato de todas las salidas que fluyen desde el sistema hacia los terminadores.
4. La *secuencia* y los tiempos de entradas y salidas en un sistema en línea.

21.2.1 Dispositivos de entrada y salida

La elección de los dispositivos de entrada y salida puede estar determinada por los terminadores fuera del sistema; por ejemplo, si el sistema produce reportes de salida para el gobierno, tal vez no haya otra opción que producirlos en papel. Los dispositivos que se usan típicamente para proporcionar entradas al sistema[5] incluyen los siguientes:

---

5 Note que estamos discutiendo las entradas proporcionadas por el usuario. Muchos sistemas (sobre todo los de tiempo real) deben manejar dispositivos que proporcionan entradas independientes de los humanos (por ejemplo, unidades de radar, registradores de datos y señales de satélite).

- *Tarjetas perforadas.* Solían ser la forma más común de entradas, pero ya rara vez se usan, excepto en algunos sistemas muy grandes de computación por lotes. Una ventaja de la tarjeta perforada es que puede ser utilizada como documento fuente por el usuario externo (por ejemplo, piense en los cheques que producen algunos sistemas de gobierno; son documentos negociables, pero también se usan como entradas directas a un sistema de cómputo). Las principales desventajas de las tarjetas son su tamaño estorboso, su limitada capacidad de almacenamiento de datos, el hecho que sólo pueden usarse una vez y la susceptibilidad a errores del operador.

- *Cinta magnética.* Pudiera ser una forma apropiada de tener entradas de otros sistemas; también puede ser un medio apropiado de entrada si el usuario dispone de un dispositivo de captura de datos de teclado a cinta. La principal ventaja de este enfoque, desde luego, es que se puede almacenar un volumen mucho mayor de datos en una cinta que en una tarjeta; la desventaja es que los datos no se pueden manipular fácilmente una vez que se graban en la cinta. La tarjeta, por otro lado, es más primitiva, pero sí permite al usuario la flexibilidad de reacomodarlas o de eliminar algunas (tirándolas a la basura) antes de ingresarlas al sistema.

- *Discos flexibles.* Con el advenimiento de las computadoras personales a comienzos de los años 80, los discos flexibles se volvieron una forma popular de medio de entradas. Los datos normalmente se registran en el disco flexible por medio de una interacción fuera de línea con una computadora personal (por fuera de línea se entiende que la actividad no tiene conexión con el sistema de información en desarrollo). Un disco flexible típico puede almacenar entre 360,000 y 1.2 millones de caracteres; esto no es tanto como lo que almacena una cinta magnética, pero es adecuado para muchas aplicaciones de mediano volumen.

- *Terminales y computadoras personales.* Las terminales de video se han vuelto una de las vías de entradas más comunes durante los diez últimos años, al bajar su costo de $3,000 dólares estadounidenses (o más) a $300 (o menos). Es importante distinguir entre terminales simples, que no proporcionan más que un teclado y pantalla; terminales inteligentes, que ofrecen una variedad de facilidades de edición local y capacidad de almacenamiento local; y las computadoras personales, que tienen una capacidad de almacenamiento local mucho mayor y todas las capacidades computacionales de una computadora de tipo general. Las terminales inteligentes y las PC vuelven posible que el usuario haga cambios y corrija errores triviales de manera instantánea, en lugar del retraso de mandar las entradas mediante líneas de telecomunicaciones a una computadora principal; la capacidad de almacenamiento local hace posible que el usuario ingrese una gran cantidad de entradas aun cuando el sistema computacional no esté operacional todo el tiempo.

- *Lectores ópticos y lectores de código de barras.* Leen información impresa o codificada en varios tipos de documentos; particularmente ventajoso para aplicaciones tales como cajas de supermercados, donde el usuario proporciona el código del producto y otros datos relevantes sobre la compra. En la medida en que estos dispositivos leen los datos *directamente,* se elimina la necesidad de teclearlos manualmente en una terminal. Algunos lectores ópticos pueden leer documentos ordinarios mecanografiados, y algunos pueden leer documentos manuscritos. La principal desventaja de este tipo de medio de entradas es su costo; otra desventaja es su tendencia a los errores.

- *Teléfono.* Para algunas aplicaciones, el teléfono por tonos puede ser un medio apropiado de ingreso de entradas.[6] Esto es particularmente ventajoso para los sistemas que tratan con el público en general; pocos tienen una terminal o una PC en su casa, pero aproximadamente el 98% de los ciudadanos norteamericanos tienen al menos un teléfono en casa. Dado que el teléfono sólo proporciona entradas numéricas, sus aplicaciones son un tanto limitadas. Es conveniente para situaciones donde el usuario sólo necesita proporcionar cosas tales como un número de cuenta, pero no sería práctico para situaciones que requieran entradas textuales.

- *Voz.* Finalmente, algunos sistemas pueden usar la voz humana como medio de entrada. La tecnología de entrada de voz de fines de los años 80 es capaz de reconocer un vocabulario de unos cientos de palabras para un usuario individual, y debe reprogramarse para cada usuario nuevo. Las ventajas son obvias; las desventajas, por el momento, son: 1) el costo del dispositivo, 2) su limitado vocabulario, 3) su tiempo de respuesta lento y 4) verdaderos problemas si la voz del usuario cambia de manera significativa debido a un resfriado o alguna otra causa.

Así como el usuario tiene opción de medios distintos de ingreso de entradas para el sistema, también hay varias posibilidades de medios para las salidas. Los más comúnmente usados son los siguientes:

- *Salidas impresas.* Es definitivamente la forma más común de salida para los sistemas de cómputo actuales. Se pueden producir con una variedad de dispositivos: impresoras de matriz de puntos (a menudo conectadas a la terminal que se usa para la entrada), impresoras de línea de alta velo-

6 Nótese que estamos haciendo distinción entre el uso del instrumento telefónico (el teléfono) y la línea de telecomunicaciones: muchas terminales están conectadas por modem a una línea telefónica; pero aquí se habla del teléfono mismo como medio de entrada.

cidad, impresoras laser de alta velocidad, impresoras laser personales, impresoras personales de margarita, etc. La principal ventaja de las salidas impresas es que constituye un documento permanente, y puede usarse en una variedad de aplicaciones fuera del sistema. Las desventajas de los reportes impresos son su tamaño estorboso, la probabilidad de que se impriman más copias (o más copias de la información) de lo que realmente se ocupa, y la velocidad relativamente baja a la que se produce la información.

- *Tarjetas perforadas.* Así como pueden servir de medio de entrada, también pueden servir para las salidas. Como se señaló anteriormente, las tarjetas perforadas pueden usarse como documentos legales; como lo señala Marjorie Leeson en [Leeson, 1981], las tarjetas perforadas pueden servir como "documento que circula" (es decir, que sale del sistema hacia un terminador externo y posteriormente se convierte en entrada para el sistema desde el mismo terminador). Pero, en general, son de tamaño estorboso y no pueden almacenar mucha información; por ello la mayoría de los sistemas de información actuales no las utilizan.

- *Terminal.* Los sistemas en línea que usan terminales como medio de entrada típicamente las usan también para las salidas. La ventaja de la terminal es que puede mostrar una gran cantidad de información a gran velocidad; con las terminales modernas se puede desplegar de manera conveniente combinaciones de textos y gráficos. La principal desventaja de la terminal es que no representa una salida material; la salida es transitoria y se pierde cuando se muestra el siguiente desplegado.

- *Salida de voz.* Para algunas aplicaciones, la voz es adecuada como medio de salida. Sucede donde el teléfono se usa como medio de entrada (véase arriba); el mismo teléfono puede usarse para llevar salida de voz al usuario. Algunas terminales también están equipadas con dispositivos de salida de voz, pero no es muy común. La principal ventaja del medio de salida de voz es que se puede usar para comunicar mensajes relativamente breves en un medio (por ejemplo, una fábrica) conde posiblemente el usuario no tenga oportunidad de leer salidas impresas.

- *Graficador.* El graficador se usa normalmente para producir diagramas y dibujos grandes y complejos (por ejemplo, dibujos y planos arquitectónicos). Los dibujos del tamaño de una hoja normal de papel pueden producirse actualmente con impresoras laser o de matriz de puntos, pero los graficadores pueden producir salidas de un metro de ancho por varios de largo. La desventaja de este medio de salida es su costo, su tamaño y la cantidad de tiempo que se requiere para producir las salidas.

- *Cinta magnética o disco.* Obviamente, así como se pueden usar para las entradas, se pueden usar para las salidas. Esto normalmente es práctico sólo en casos en los que las salidas se van a mandar a otros sistemas de cómputo (es decir, donde el terminador del sistema no es una persona, sino una computadora).

- *COM.* Acrónimo de las siglas en inglés de Microforma para Salidas de Computadora. Las salidas COM normalmente se reservan para archivos (por ejemplo, material de referencia voluminoso que sería demasiado caro y estorboso producir como reportes impresos normales). Los rollos de microfilm (que los bancos utilizan, por ejemplo, para guardar copias de cheques cancelados) o las tarjetas de microfichas son ejemplos de esto.

21.2.2 Formatos de entrada y salida

Una vez escogidos los dispositivos de entrada y salida, el siguiente paso es determinar los *formatos* de las entradas y salidas del sistema. En algunos casos, los formatos pudieran no ser cuestión de negociación, sino simplemente cuestión de que el usuario informe al analista de "la manera en la que las cosas tienen que ser". Es así sobre todo si el nuevo sistema debe comunicarse con otros sistemas o con personas (o grupos) *externos* a la organización que construye el nuevo sistema. Las organizaciones externas o los otros sistemas de cómputo externos pueden proporcionar datos al sistema nuevo en un formato físico prescrito que no se puede cambiar. Igualmente, pueden requerir salidas del sistema con un formato también rígidamente prescrito.

Si el diálogo humano-máquina no se ha definido por completo, ¿qué hay aún por negociar? No la representación interna de los datos dentro del sistema de cómputo pues al usuario no le preocupa, ni debiera percatarse de esta información. Tampoco cosas tales como los valores y límites legales de los datos de entrada, pues deberían haberse especificado como parte del modelo esencial. Sin embargo, si es momento de negociar restricciones razonables de la *implantación* sobre aspectos varios de los datos. A continuación hay unos ejemplos:

- El modelo esencial podría haber identificado el dato **APELLIDO-DEL-CLIENTE.** Como cuestión de política *esencial,* podría no haber límite para la longitud (número de caracteres) de este dato. Después de todo, ¿qué tiene de malo un apellido de 357 letras? Aunque no sucede frecuentemente, algunos miembros de la nobleza europea podrían desear demostrar su linaje incluyendo los nombres de todos sus antepasados ancestrales en su apellido. Esto es interesante y pudiera tener algún significado histórico, pero el usuario y el analista podrían no obstante estar de acuerdo en restringir **APELLIDO-DEL-CLIENTE** a 25 caracteres. Note, por cierto, que esto requerirá de un cambio en las especificaciones apropiadas del proceso que maneja la entrada de **APELLIDO-DEL-CLIENTE** para asegurar que sea válido de acuerdo con esta restricción de la implantación.

- En un sistema de ingreso de pedidos, un **PEDIDO-DE-CLIENTE** podría definirse como ***PEDIDO-DE-CLIENTE* = NOMBRE + DOMICILIO + ARTICULO PEDIDO.** En el modelo esencial, podría no haber razón para limitar el número de artículos diferentes que un cliente compra en un solo pedido. Desde la perspectiva de la implantación del usuario, sin embargo, existen varias razones: 1) como actividad de apoyo manual (que se discute más adelante, en la Sección 21.3), el usuario puede desear que el empleado de ventas tome el pedido en una forma pre-impresa donde sólo caben, por ejemplo, 8 artículos distintos; 2) al usuario podría preocuparle que el empleado de ventas cometa algún error si trata de manejar más de cierto número limitado de artículos en cada pedido; 3) al usuario podría preocuparle que el prestar servicio a uno de sus clientes con 597 artículos distintos moleste a otros clientes que esperan servicio, etc. Por tanto, hay un buen número de razones válidas para imponerle a este dato alguna restricción definida por el usuario.

Observe que las cuestiones de implantación del usuario de este tipo involucran sobre todo anotaciones extra en el diccionario de datos, además de lógica adicional (si se necesita) en las especificaciones del proceso que tratan con la validación de datos de entrada. Pero hay otro aspecto del diálogo humano-máquina que requiere algo más que anotaciones del diccionario de datos: la *secuencia*, sobre todo en un sistema en línea, que se puede modelar utilizando un diagrama de transición de estados. La figura 21.3 muestra un ejemplo típico de diagrama que se usa para modelar la secuencias en pantalla que el usuario final utiliza para comunicarse con el sistema.

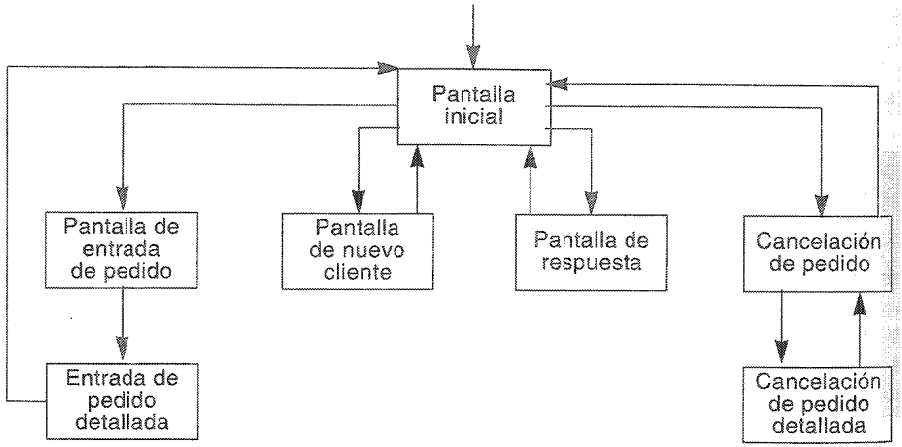


Figura 21.3: **Diagrama de transición de estados para modelar pantallas de video**

Esto es útil sobre todo para manejar preguntas como: "¿Puedo regresar al menú principal desde la imagen de pantalla 12?" Otras cuestiones de entrada/salida importantes son el acomodo físico de los datos en la pantalla de video, la naturaleza de los mensajes si se comete un error de entrada; y el acomodo físico de los datos de salida en la pantalla y los reportes. Con la gran gama de lenguajes de cuarta generación, de herramientas para hacer prototipos y de computadoras personales que existe actualmente, recomiendo dar al usuario oportunidad de jugar con diversas variantes de pantallas de entrada, despiegados de salida, etc.[7] Una vez que el usuario esté de acuerdo, anexe al modelo esencial una copia de los despiegados en pantalla, formatos de reportes y diagramas de transición de estados apropiados, con referencias apropiadas a los datos esenciales del diccionario de datos.

Desde luego, en muchos casos el usuario no tendrá mayores sugerencias, pues no tiene experiencia previa trabajando con un sistema de cómputo; esto es un tanto análogo a alguien que ha vivido en un departamento toda su vida, pero ahora quiere especificar los detalles de su primera casa de varios niveles, de diseño exclusivo. En el caso de sistemas de información computarizados, las siguientes reglas ayudarán a desarrollar una interfaz amable con el usuario en la mayor parte de los casos:

1. *El sistema debe pedir entradas y producir salidas en forma consistente.* Esto es particularmente cierto para los sistemas en línea donde el usuario puede ingresar una o varias transacciones y/o recibir uno o más despliegues distintos. Por ejemplo, suponga que el sistema pide al usuario la fecha cuando se ingresa una transacción; sería un desastre si un tipo de transacción requiere la fecha en la forma 12/23/87 mientras que otra la requiere en la forma 87/12/23. Y sería un desastre si una parte del sistema requiere que el usuario especifique el estado como una clave de dos caracteres (por ejemplo, NY para Nueva York), mientras que otra pide que se deletree el estado. De manera similar, si una parte del sistema requiere que el usuario proporcione una clave de larga distancia cuando se ingresa un número telefónico, entonces *todas* las demás deben solicitarla cuando se ingresa el número de teléfono.

2. *Pida información con una secuencia lógica.* En muchos casos esto depende de la naturaleza de la aplicación y requerirá discusiones cuidadosas con el usuario. Por ejemplo, suponga que se está desarrollando un sistema de ingreso de pedidos, y se quiere que el usuario especifique el nombre del cliente. Sea que la información se ingrese en tarjetas perforadas o (lo que es más probable) desde una terminal de video, tendrá que especificar la secuencia en la cual desea que se ingresen los componen-

7 De hecho, puede requerir herramientas de inteligencia artificial de quinta generación para experimentar con entradas en lenguaje natural, entradas de voz, salidas gráficas, etc.

tes de "nombre de cliente". Una secuencia lógica sería pedirle al usuario proporcionar la información en la siguiente así:

título (Sr./Srita./etc.)

nombre

inicial del segundo nombre

apellido

Pero el usuario podría encontrar esto muy difícil. Suponga que se ha obtenido el nombre del cliente de alguna fuente externa, como un directorio telefónico. En este caso, sería mucho más conveniente teclear:

apellido

nombre

inicial del segundo nombre

título

De lo que sí se puede estar seguros es que la siguiente secuencia sería bastante impopular con el usuario:

inicial del segundo nombre

nombre

título

apellido

3. *Haga obvio al usuario el tipo de error que ha cometido, y dónde.* En muchos sistemas, el usuario proporciona al sistema mucha información por cada evento o transacción. Por ejemplo, en un sistema de ingreso de pedidos, el usuario puede especificar el nombre del cliente, su domicilio y número telefónico además de información acerca de los artículos pedidos, como descuentos, instrucciones de envío e impuestos sobre la venta. Toda esta información puede colocarse en una sola pantalla de información de video antes de enviarla al sistema. Desde luego, el sistema podría entonces determinar que parte de, o todas, las entradas son erróneas. Lo importante es asegurar que el usuario entienda el tipo de error que se está cometiendo y dónde se localiza; no es aceptable que el sistema simplemente suene una señal y despliegue el mensaje ENTRADA INVALIDA. *Cada error* (suponiendo que haya más de uno) debe identificarse, con un mensaje descriptivo, o bien (en el caso de un sistema en línea) enfatizando o mostrando el dato erróneo de acuerdo con algún código de colores. Dependiendo de la naturaleza del sistema y el nivel del usuario, podría también ser importante desplegar un mensaje de explicación; esto se discute con mayor detalle en el inciso 5.

4. *Distinga entre edición de campos y ediciones de pantalla.* En muchos casos el sistema será capaz de determinar si el dato proporcionado por el usuario es o no correcto sin hacer referencia a otros datos. Por ejemplo, si se teclea un código de dos caracteres para un estado, se debería determinar inmediatamente si esos dos caracteres representan un estado válido, una provincia, etc. Pero si el usuario fuera a ingresar un código postal como dato individual, hay sólo una cantidad limitada de edición en el campo que se puede realizar sin necesidad de entradas adicionales; se requiere *tanto* el código postal como el código del estado para determinar si el código postal debe ser de cinco dígitos, de nueve, o si debe ser un código postal de longitud seis. El sistema podría comparar el código estatal y el postal para determinar si el postal está dentro del rango adecuado (por ejemplo, si el código estatal es NY entonces el postal debe empezar con el dígito 1). La relación entre los datos pudiera ser obvia para el analista; sin embargo, requiere conocimientos detallados e íntimos de la aplicación que sólo se pueden obtener del usuario.

5. *Haga la edición y la revisión de errores dependientes del usuario.* Como se indicó anteriormente, suele ser buena idea que el sistema despliegue un mensaje de explicación cuando se detecta un error, pero sólo si el usuario no puede determinar por sí mismo lo que hizo mal. Si tecleó, por ejemplo, un código de tres dígitos, no es necesario que el sistema despliegue un mensaje elaborado sobre de la longitud de los códigos; sin embargo, sí lo sería si el usuario teclea un código de cinco dígitos y el sistema espera uno de nueve. Note también que a) algunos usuarios son más conocedores que otros y pueden molestarse aún más rápido si tienen que ver largos y pomposos mensajes de error y, b) tras el uso repetitivo, incluso un novato se convierte en experto en algunas partes del sistema. Por ello, es importante hacer que los mensajes de error sean flexibles y, tal vez, que el usuario los pueda cambiar; lo más fácil es combinar mensajes cortos (que pueden consistir sólo en enfatizar las entradas erróneas y sonar la alarma para atraer la atención del usuario) y mensajes largos (con texto explicativo y una referencia a alguna parte apropiada del manual para el usuario). Véase también el inciso 7.

6. *Permita que el usuario pueda (a) cancelar parte de la transacción y, (b) cancelarla toda.* No es aconsejable suponer que el usuario siempre terminará de ingresar toda la transacción sin que se interumpa. Con un sistema de cómputo por lotes esto no sucede: normalmente el sistema no ve nada proveniente del usuario sino hasta haber manejado varias transac-

ciones individuales.[8] Pero para los sistemas en línea es un asunto importante: el usuario puede haber ingresado el nombre y domicilio de un cliente antes de darse cuenta de que está trabajando con uno equivocado, por lo que desea borrar todo y volver a empezar. O pudiera haber terminado de ingresar la mayor parte de la información del cliente y luego se da cuenta de que escribió mal el nombre por lo que desea regresar a ese dato y corregirlo sin perder todo el resto de la información que tecleó.

7. *Proporcione un mecanismo de "ayuda" conveniente.* Para los sistemas en línea es cada vez más importante proporcionar al usuario un mecanismo conveniente para obtener información sobre cómo usar el sistema. En algunos casos, el mecanismo de "ayuda" simplemente proporciona una explicación si el usuario comete algún error; en otros puede usarse para explicar los detalles de diversas órdenes o transacciones disponibles. Existen muchas formas de realizar esto, y tanto el analista como el usuario deben investigar diversos ejemplos típicos antes de tomar una decisión (la mayoría de los paquetes de software disponibles para la IBM PC y la Apple Macintosh tienen mecanismos de "ayuda"; ése es un buen inicio).

8. *Distinga entre sistemas guaidos por menús y sistemas dirigidos por órdenes; si es apropiado, déle a escoger al usuario.* Un sistema dirigido por menús presenta una lista de opciones (o funciones, o transacciones, etc.) alternativas; una vez que se escoge una, puede aparecer otro sub-menú, que llevaría a sub-menús de nivel inferior antes de que finalmente el sistema entre en acción. Un sistema dirigido por órdenes, por otro lado, espera que el usuario proporcione una directiva detallada (y a menudo larga) indicando lo que quiere que el sistema haga por él. Los sistemas dirigidos por menús se consideran más amables con el usuario, porque muestran todas las opciones disponibles; por tanto, un sistema dirigido por menús se considera preferible para usuarios nuevos, o si debe interactuar con una gran variedad de usuarios de diferente preparación y nivel de habilidad. Pero el usuario con experiencia lo considera tedioso, porque muchas veces se requieren dos o tres interacciones diferentes (y cada una toma un tiempo) antes de que el sistema finalmente se dé por enterado de lo que el usuario desea. Un usuario con experiencia generalmente prefiere un sistema dirigido por órdenes, para lograr lo que desea lo más rápidamente posible.

8 Pero incluso con un sistema por lotes, el usuario puede darse cuenta que ha ingresado datos por error al sistema. Casi siempre será necesario darle la posibilidad de deshacer o regresar lo que metió. Pero esto debió descubrirse durante las entrevistas con el usuario, y debiera ser evidente ya en el DFD y las especificaciones de proceso para el sistema.

9. *Si el sistema está realizando un proceso largo, despliegue un mensaje al usuario para que no crea que se detuvo.* Si el sistema tiene que llevar a cabo cálculos muy extensos, o si es probable que se retrase periódicamente por el volumen de las entradas, es importante desplegar un mensaje apropiado al usuario; de otro modo, puede pensar que su terminal se "congeló" o se "cayó"; que el sistema de cómputo falló, o que hubo una interrupción en la corriente. Por lo menos, el sistema debe mostrar un mensaje (por ejemplo, FAVOR DE ESPERAR - SE ESTA PROCESANDO) a intervalos regulares. Sería mejor una serie de mensajes informando al usuario qué cantidad del trabajo se ha concluido y aproximadamente cuánto se tardará en completar todo.

10. *Proporcione alternativas por omisión para las entradas estándar.* En muchos casos, el sistema puede adivinar bastante certeramente el valor probable de alguna entrada dada por el usuario; al proporcionar alternativas por omisión se ahorra tiempo y actividad de tecleo al usuario. Este enfoque es válido para todo tipo de sistemas, pero sobre todo para los en línea, donde el usuario tal vez no sea un mecanógrafo profesional. Un ejemplo es la fecha: en muchas aplicaciones la fecha más probable que el usuario ingresará es la de hoy. O suponga que se está construyendo un sistema de ingreso de pedidos, y el usuario le dice que el 95% de los clientes viven en el área local; entonces, cuando pida al usuario proporcionar el número telefónico, el sistema debe suponer que la clave es la local, a menos de que se especifique lo contrario.

11. *Aproveche el color y el sonido, pero no abuse.* Las terminales modernas tienen una variedad de colores y efectos de sonido que pueden ser útiles para enfatizar diferentes tipos de entradas o para atraer la atención del usuario hacia aspectos importantes de la interfaz humana. Se podría utilizar el verde para todo el material que el sistema despliega; azul para todas las entradas proporcionadas por el usuario, y rojo para todos los mensajes de error. Sin embargo, no abuse: muchos usuarios se molestan si su pantalla parece árbol de Navidad. Lo mismo se aplica a los efectos de sonido; una campana o alarma ocasional puede ser útil, pero el usuario no quiere que la terminal produzca todos los efectos sonoros de la película de La Guerra de las Galaxias.

21.2.3 Diseño de las formas

El diseño de los formatos de entrada y salida de un sistema tradicionalmente se conoce como diseño de formas, pues la mayor parte de los sistemas de los años 60 y 70 requerían que el usuario codificara las entradas en formas de papel que luego se transcribían a tarjetas perforadas antes de ingresar al sistema de cómputo por lotes. Pero incluso en los sistemas en línea actuales se requiere un poco de diseño de formas; considere por ejemplo la situación común en la que las entradas del sis-

tema se originan en un cliente externo. El proporciona las entradas requeridas llenando la forma y enviándola por correo a los usuarios que interactúan con el sistema; la figura 21.4 muestra un ejemplo de una forma real. Se debe prestar atención al diseño de estas formas.

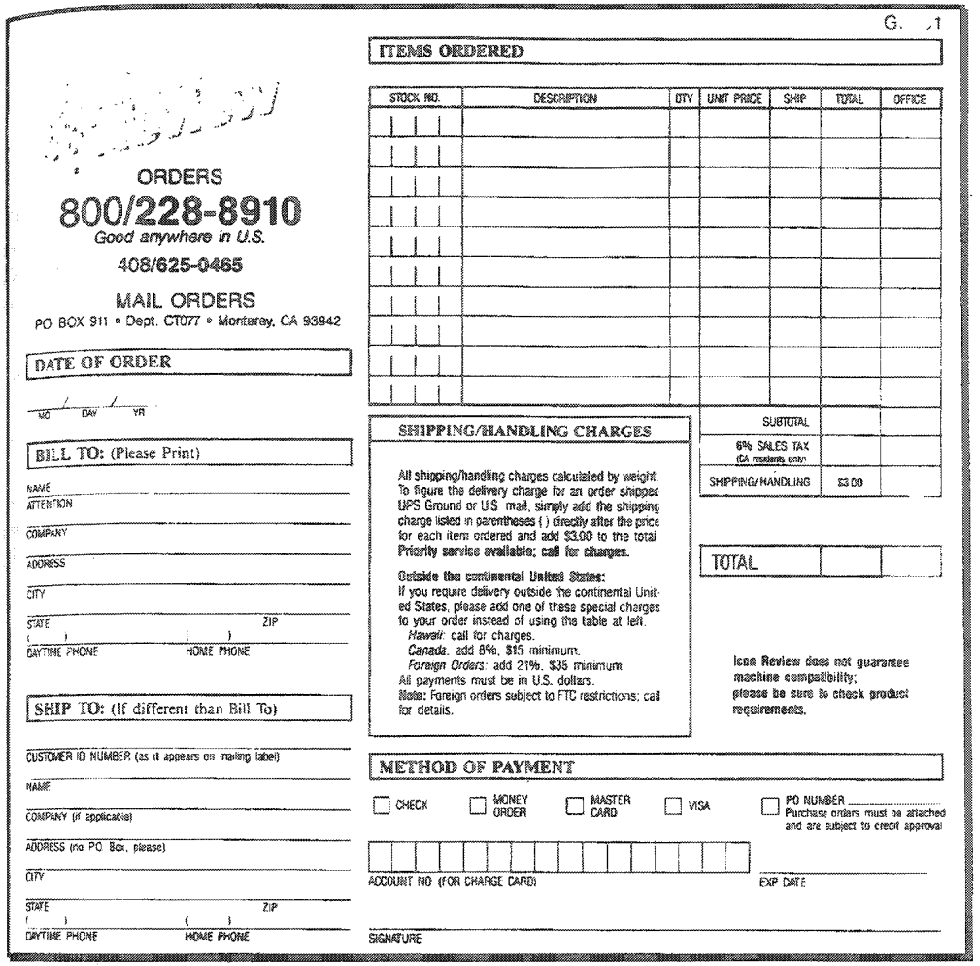
En ciertos casos, el analista puede recurrir a algún departamento interno de diseño gráfico o a un productor de formas externo para obtener ayuda; alternativamente, pudiera tener formas asociadas con el sistema existente que desee continuar usando en el nuevo. También habrá muchas situaciones en las que el analista y el usuario diseñen formas nuevas para un sistema nuevo. Aunque hay muchos estilos diferentes para las formas, todas deben contener la siguiente información básica:

- *Título,* para distinguirla de cualquier otra. El título usualmente se imprime con letras grandes y resaltadas en la parte superior de la forma para que el usuario sepa que es una "forma de pedido", o bien una "forma de reporte de fallas".
- *Instrucciones,* para decir al usuario cómo poner la información necesaria en la forma. Las instrucciones generales usualmente se colocan al principio de la forma, cerca de la parte superior. Suelen colocarse las instrucciones específicas cerca, o bajo cada dato que se necesite escribir.
- *Cuerpo,* es decir, la parte principal de la forma, donde se ingresan los datos. Puede organizarse con un estilo abierto, o encajonado, o una combinación de ambos. El estilo encajonado típicamente se usa para información binaria (por ejemplo,"marque esta caja si desea que se añada su nombre a nuestra lista de correo para anuncios de productos futuros") o de formato fijo (por ejemplo, un número telefónico o código postal). El estilo abierto típicamente se usa para información de longitud variable tal como nombre o domicilio.

El decidir exactamente cómo debe distribuirse la forma es un arte en sí mismo y lo realizan mejor personas con experiencia en el diseño de formas. Uno de los errores más comunes del diseñador novato es, por ejemplo, no dejar espacio suficiente para la información requerida. Esto sucede sobre todo con formas que requieren información manuscrita.[9]

Dependiendo de la aplicación, el analista puede diseñar formas individuales o de especialidad. Las primeras suelen imprimirse en hojas sencillas y son adecuadas para la gran mayoría de las situaciones; con la disponibilidad de los sistemas de edi-

9 Deje por lo menos un centímetro de espacio vertical para cada renglón de información manuscrita en una forma. Deje casi un centímetro y algún múltiplo de medio centímetro por cada línea de información mecanografiada. Asegúrese de que el mecanógrafo no tenga que realinear la máquina cada vez que pasa a otro renglón.



G. 21

ITEMS ORDERED

STOCK NO.	DESCRIPTION	QTY	UNIT PRICE	SHIP	TOTAL	OFFICE

ORDERS
800/228-8910
Good anywhere in U.S.
408/625-0465
MAIL ORDERS
PO BOX 911 • Dept. CT077 • Monterey, CA 93942

DATE OF ORDER
MO / DAY / YR

BILL TO: (Please Print)
NAME
ATTENTION
COMPANY
ADDRESS
CITY
STATE ZIP
( ) ( )
DAYTIME PHONE HOME PHONE

SHIP TO: (If different than Bill To)
CUSTOMER ID NUMBER (as it appears on mailing label)
NAME
COMPANY (if applicable)
ADDRESS (no PO Box, please)
CITY
STATE ZIP
( ) ( )
DAYTIME PHONE HOME PHONE

SHIPPING/HANDLING CHARGES

All shipping/handling charges calculated by weight. To figure the delivery charge for an order shipped UPS Ground or US mail, simply add the shipping charge listed in parentheses ( ) directly after the price for each item ordered and add $3.00 to the total. Priority service available; call for charges.

Outside the continental United States:
If you require delivery outside the continental United States, please add one of these special charges to your order instead of using the table at left.
Hawaii: call for charges.
Canada, add 8%, $15 minimum.
Foreign Orders: add 21%, $35 minimum.
All payments must be in U.S. dollars.
Note: Foreign orders subject to FTC restrictions; call for details.

SUBTOTAL	
6% SALES TAX (CA residents only)	
SHIPPING/HANDLING	$3.00
TOTAL	

Icon Review does not guarantee machine compatibility; please be sure to check product requirements.

METHOD OF PAYMENT

☐ CHECK ☐ MONEY ORDER ☐ MASTER CARD ☐ VISA ☐ PO NUMBER
Purchase orders must be attached and are subject to credit approval

ACCOUNT NO (FOR CHARGE CARD) EXP DATE

SIGNATURE

Figura 21.4: **Ejemplo de una forma típica**

ción de escritorio y los programas de diseño de formas, el usuario y el analista pueden diseñar fácilmente sus propias formas.

Las formas de especialidad son más complejas y se crean con la ayuda de un diseñador de formas con experiencia (que normalmente se asocia con un productor de formas). El ejemplo más común es una forma múltiple, que usa hojas de papel carbón o papel especial de copia sin carbón. Los tipos de formas de especialidad son:

- Formas empastadas en libros (por ejemplo, libros de pedidos de ventas)
- Formas múltiples desprendibles, con un original y varias copias que se pueden separar (por ejemplo, formas de cobro de tarjeta de crédito).
- Formas continuas, que se llenan manualmente o por computadora.
- Para correo: formas preimpresas insertas en un sobre, unidas como en una forma continua. La computadora puede entonces imprimir información estándar, tal como nombre y domicilio del cliente, que (por medio del papel carbón) se imprime tanto en el sobre como en la carta que lleva dentro.

Las formas de especialidad son, como se espera, mucho más caras que las sencillas; por lo que se debe tener cuidado de que no representen un costo importante del sistema. Las formas de especialidad deben producirse en cantidad razonable para mantener bajo el costo unitario: el costo de imprimir 10,000 copias de una forma de especialidad suele ser un 10% o 20% más que el costo de imprimir 5000. Se deben usar formas de tamaño estándar para que la compañía impresora no tenga que hacer recortes caros; la mayor parte de las formas estándar son de 8.5" por 11" o bien de 5.5" por 8.5".

21.2.4 Códigos de entrada y salida

Como parte de la labor de especificar formatos de entrada y salida, el analista muchas veces debe especificar *códigos*, es decir, abreviaciones de la información que sería difícil y tardado describir con detalle. Ejemplos de éstos son los números del Seguro Social, códigos postales, números de ISBN para libros publicados y números de identificación de empleados que se asignan a las compañías en sus declaraciones de impuestos.

Los ejemplos anteriores representan *códigos externos* para la mayoría de nosotros; es decir, sin importar el tipo de sistema, tenemos que usar códigos desarrollados por el gobierno, Correos y el Seguro Social. Pero a menudo existen situaciones en las que se necesita designar códigos nuevos asociados con el sistema mismo (por ejemplo, números de cuenta de clientes, números de refacciones, números de formas, códigos de productos, códigos de colores y números de vuelos de aerolíneas). Así como el diseño de formas es un arte, las técnicas de codificación son un área especializada. Como se señala en [Gore y Stubbe, 1983], un método de codificación debe ser:

"• Expandible. Debe proporcionar espacio para entradas adicionales que pudieran requerirse.

- Preciso. Debe identificar al artículo específico.
- Conciso. Debe ser breve pero describir adecuadamente al artículo.

- Conveniente. Debe ser fácil de codificar y decodificar.
- Con significado. Debe ser útil para quienes lo manejan. De ser posible debe indicar algunas de las características del artículo.
- Operable. Debe ser compatible con los métodos presentes y anticipados de proceso de datos, manual o a máquina."

En algunos casos tal vez no sea necesario, deseable o práctico que el código tenga una relación obvia con el artículo que describe. Un buen ejemplo es el número de cliente, de cuenta o de empleado en muchos sistemas: el código es simplemente un número escogido en secuencia. Sin embargo, también es común que la técnica de codificación reserve bloques de números (o letras) para artículos dentro de una categoría común; por ejemplo, un sistema de ingreso de pedidos puede usar cuatro dígitos como número de producto, reservando los números del 1 al 500 para artículos estándar y del 501 al 999 para artículos especiales.

Es más común el *código de clasificación*, que usa grupos de dígitos (o letras) dentro del código para identificar clasificaciones mayores, intermedias o menores dentro del artículo que se describe. Por ejemplo, para llamar a mi oficina en Londres, marco los dígitos siguientes:

011-44-1-637-2182

Los primeros tres dígitos identifican el número telefónico como internacional (comparado con los correspondientes a llamadas dentro de los EUA.). Los siguientes dos son el código de país, siendo el 44 el código del Reino Unido. El siguiente dígito es el código de Londres, análogo al código de zona de tres dígitos que se usa en los EUA. Los siguientes tres representan un conmutador telefónico y a menudo dan al usuario astuto una buena idea del barrio de Londres en el que se localiza el teléfono. Y, finalmente, los últimos cuatro identifican un teléfono específico.

Los códigos alfabéticos también se usan comúnmente en sistemas de información. Muchos códigos alfabéticos son intentos mnemónicos, o auxilios de memoria, que el usuario podrá recordar fácilmente.[10] Que el intento tenga éxito o no depende de la brevedad del código (por ejemplo, dos dígitos en contraposición con diez), de la diversidad y disparidad de los datos mismos, y de la familiaridad del usuario con

---

10 Algunos códigos alfabéticos parecen ser todo lo contrario a mnemónicos; éstos son los que se derivan de uno o más atributos del dato. Un ejemplo es el código que se encuentra en la etiqueta postal de algunas revistas; el código del subscriptor usualmente consta de una porción del apellido, su domicilio, código postal, fecha de vencimiento de la subscripción y otros detalles. Como tal, ciertamente puede no ser mnemónico: no hay manera de que alguien recuerde un código de 20 o 30 caracteres. Sin embargo, una vez que se da el código al sistema de cómputo se puede recuperar el registro del subscriptor bastante rápido, lo cual puede ser muy importante para una base de datos de subscriptores de varios millones de registros. Para obtener más información acerca de estos códigos derivados, consulte la forma GF20- 8093 de IBM, titulada Data Processing Techniques: Coding Methods.

éstos. Por ejemplo, considere los códigos de dos letras que se usan para identificar diferentes aerolíneas; la mayoría de los ciudadanos de los EUA inmediatamente reconocerían que AA representa a American Airlines, y que UA representa a United Airlines. ¿Pero cuántos sabrían que HW representa a Havasu Airlines y que AQ representa a Aloha Airlines? Con los códigos de tres letras hay mejor oportunidad de escoger códigos mnemónicos, como lo ilustran los códigos que se usan para identificar a aeropuertos. Casi todo mundo sabría que JFK quiere decir Aeropuerto John F. Kennedy de Nueva York, y que SFO es el aeropuerto de San Francisco. Pero aquí hay problemas a menos de que el usuario haya memorizado muchos códigos que son todo menos mnemónicos (por ejemplo, ORD para O'Hare, en Chicago, y YYZ para el aeropuerto de Toronto).

Finalmente, algunos códigos se autoverifican; es decir, contienen información adicional (redundante) que puede usarse para verificar que se haya ingresado correctamente. Un ejemplo común de código autoverificador es el que contiene un dígito de verificación, que usualmente se anexa al final del código numérico. Puede calcularse en una variedad de formas, una de las cuales se da a continuación:

dígito-de-verificación = 0
PARA cada dígito en el código numérico
    suma = dígito multiplicado por su número de posición
    dígitoverificador = dígitoverificador + suma
FIN

HACER MIENTRAS haya más de un dígito en **dígito-de-verificación**
    dígitoverificador = la suma de todos los dígitos en **dígito-de-verificación**
FIN

Por ejemplo, si se tiene el código numérico 9876, el dígito de verificación se calcularía como (9*1) + (8*2) + (7*3) + (6*4), que resulta en 70. Sumando los dígitos 7 y 0 se obtiene un resultado final de 7 como dígito de verificación. El objetivo no es que el usuario haga todo el cálculo, sino usar un código que incluya un dígito de verificación (por ejemplo, 9876-7). Luego, cuando el usuario ingresa el código al sistema, la computadora recalcula automáticamente el dígito de verificación esperado (usando el algoritmo descrito anteriormente) y lo compara con el dígito de verificación dado. Un error, usualmente significa que uno de los dígitos se transpuso cuando el usuario lo ingresó.

21.3 IDENTIFICACION DE LAS ACTIVIDADES DE APOYO MANUAL ADICIONAL

En el modelo esencial se supone la existencia de una tecnología perfecta, que significa, entre otras cosas, suponer que la tecnología de implantación nunca se descompondrá y nunca cometerá un error. Los usuarios podrían no estar dispuestos a aceptar esto, ¿y quién los culparía? Además, el usuario podría decidir que ciertas porciones del sistema automatizado esten bajo su propio control operacional (por

ejemplo, una PC o una minicomputadora para su área de trabajo) y preocuparse por posibles errores operacionales que su propio personal pudiera cometer. Adicionalmente, tal vez trabaje con datos financieros, en cuyo caso podrían existir requisitos legales (o requisitos impuestos por los auditores) para asegurar la integridad de las entradas, salidas y archivos del sistema. En la mayor parte de los casos, estas actividades de apoyo adicional se representarán por medio de nuevos procesos en el DFD del modelo de comportamiento. En la figura 21.5 se muestra un ejemplo.

En general, tenemos que preocuparnos por la posibilidad de la tecnología defectuosa en cuatro áreas principales, como lo ilustra la figura 21.6.

- *Ingreso de datos al sistema.* Si los datos de entrada se proporcionan por medio de terminales de video conectadas con las computadoras principales usando líneas de telecomunicaciones, entonces es posible que algunas o todas las transacciones se pierdan o revuelvan. Lo mismo puede suceder con casi cualquier tipo de entrada; por ejemplo, el operador de la computadora podría dejar caer una o dos tarjetas, o uno de varios discos flexibles podría no ser leído por el sistema.

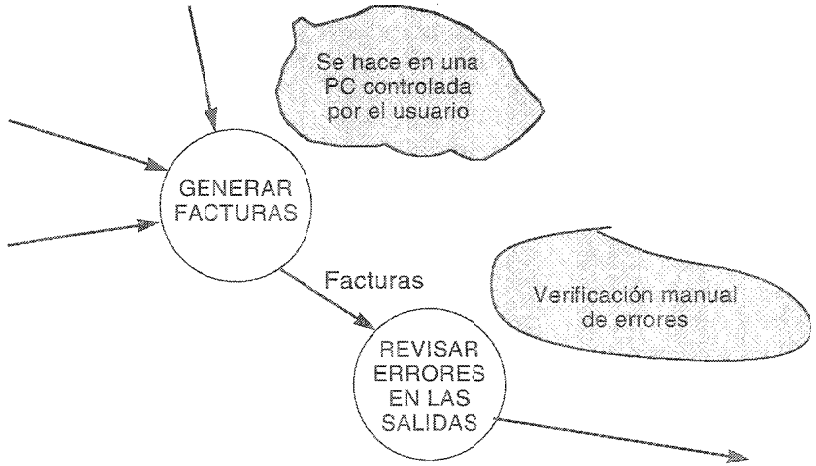


Figura 21.5: **Actividad de apoyo manual de revisión de errores**

- *Realización de los cálculos.* Una vez ingresados los datos al sistema, existe la remota posibilidad de que la computadora misma pudiera funcionar mal, y la posibilidad aún mayor de que un error en los programas pudiera producir un resultado incorrecto. Los errores de hardware pueden ocurrir en un procesador individual o en la interfaz entre varios en la configuración del sistema.

- *Almacenamiento a largo plazo de los datos.* La mayor parte de los sistemas guardan información durante largos periodos en discos magnéticos, cintas magnéticas, discos flexibles, etc. Es posible que algunos (o todos) estos datos se pierdan o destruyan debido a errores de hardware y/o software. Los errores de hardware son problemas del dispositivo mismo de almacenamiento, o problemas de la conexión entre el procesador (CPU) y el dispositivo. Los errores de software pueden ocurrir en los programas de aplicación desarrollados por el equipo del proyecto, o bien en el software de administración de bases de datos comprado.

- *Salida de datos del sistema.* Los problemas potenciales que se pueden dar aquí son análogos a los que se tienen con las entradas al sistema. En algunos casos, las salidas deben transmitirse por líneas de telecomunicaciones, a menudo hacia la misma unidad de video que se usó para las entradas. En otros casos, las salidas se producen en cinta magnética o en reportes en papel. En todos los casos es posible que se pierda información de salida o se altere incorrectamente como resultado de errores de hardware y/o software.

¿Qué debe hacerse al respecto de estas áreas posibles de tecnología defectuosa? Obviamente, depende en gran medida de 1) el nivel estimado de confiabilidad del hardware y software que se usa; 2) la naturaleza de la aplicación del usuario y, 3) los costos y cargos asociados con entradas o salidas defectuosas. Como es obvio, esto amerita una discusión detallada entre el usuario, los analistas y el equipo de implantación; podría decidirse añadir cualquiera de lo siguiente al modelo esencial para vérselas con la tecnología defectuosa:

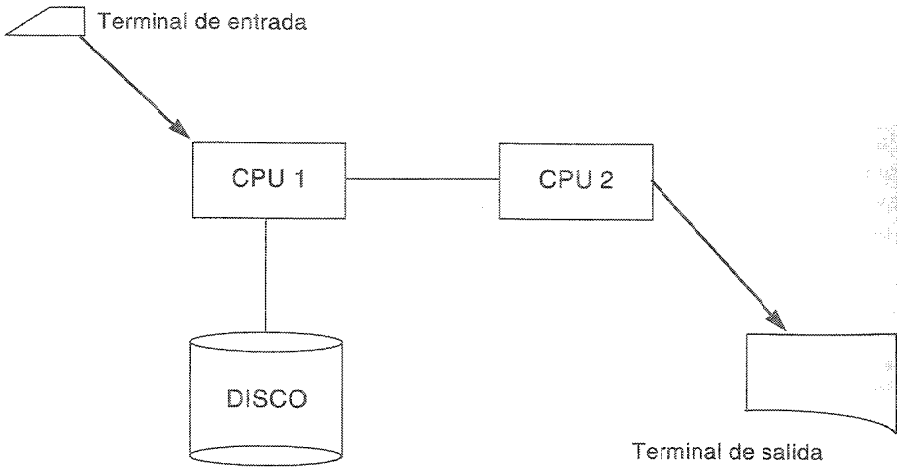


Figura 21.6: **Posibles áreas de tecnología defectuosa**

- *Entradas o salidas redundantes.* En el caso más extremo habría duplicados de las entradas desde dos fuentes distintas (por ejemplo, de dos usuarios distintos ante dos terminales distintas). Además, se pueden producir salidas duplicadas (por ejemplo, dos copias distintas de un reporte de salida hechas con distintas impresoras). Este es un enfoque poco usual, pero pudiera considerarse para aplicaciones extremadamente delicadas.

- *Tecnología de procesador redundante.* Los datos que el sistema mantiene pueden almacenarse, por duplicado, en dos discos o cintas diferentes. Los cálculos que se realizan podrían hacerse, por duplicado, en dos procesadores distintos. De hecho, podría requerirse incluso mayor redundancia: aunque un segundo procesador o disco permite que el sistema continúe si la primera unidad se descompone completamente, no protege contra errores sutiles. ¿Qué tal si el CPU 1 y el CPU 2 realizan el mismo cálculo (por ejemplo, el cálculo de los intereses de una cuenta de ahorros, o de la trayectoria de un misil para un viaje a la luna) y producen distintos resultados? ¿Cuál está en lo correcto? En el caso extremo, entonces, debe insistirse en procesadores por triplicado y almacenamiento por triplicado, con una lógica de voto de mayoría para determinar cuáles dos están en lo correcto. ¿Pero qué tal si las tres están en desacuerdo?

- *Redundancia interna.* Una manera más común de manejar la tecnología defectuosa es mediante redundancia parcial. Por ejemplo, podría pedírsele al empleado bancario que proporciona datos sobre depósitos en un sistema bancario en línea que dé tanto el número de cuenta como el nombre de quien deposita. Aunque normalmente baste con identificar el registro del cliente en el sistema, siempre cabe la posibilidad de que el empleado lo ingrese incorrectamente, o que el número de cuenta se altere debido a algún error de telecomunicaciones, o a un error en la terminal de video, o un error en el procesador. Alternativamente, el sistema podría requerir que el empleado ingresara sólo el número de cuenta y verificara si está correcto desplegando el nombre del cliente después de recuperar el registro.

- *Controles por lote (batch).* Este enfoque se introdujo primeramente en sistemas de cómputo por lotes en la segunda generación, pero todavía es relevante en muchos de los sistemas actuales. Requiere que el usuario mantenga cuenta de las transacciones que ingresa al sistema, además del total acumulativo de los datos importantes en dichas transacciones. En un sistema financiero, lo obvio sería contar la cantidad de cada transacción. Mientras tanto, el sistema mantiene su propia cuenta al recibir las transacciones; periódicamente, el sistema y el usuario comparan sus cuentas para asegurar que no se ha perdido o alterado nada. Se puede

usar el mismo enfoque con las salidas: el sistema puede mantener su propia cuenta del número de transacciones que ha dado como salidas, para luego compararlas con una cuenta manual proporcionada por el usuario.

- *Verificaciones secuenciales.* Este enfoque se relaciona con el concepto de controles por lote. El usuario asigna un número de secuencia o de transacción a cada entrada, y el sistema verifica, al procesarlas, que estén en secuencia y que no se pierda ninguna transacción. De manera similar, el sistema puede añadir un número de secuencia a cada salida que produce, y el usuario verifica manualmente que no se hayan perdido.

21.4 ESPECIFICACION DE RESTRICCIONES OPERACIONALES

Finalmente, el equipo de implantación tendrá que decidir la combinación de hardware, sistema operativo, equipo de telecomunicaciones, lenguaje de programación y estrategia de diseño para implantar mejor los requerimientos. Pero esto será difícil de lograr sin alguna declaración de restricciones operativas, que el modelo esencial deliberadamente evita. Las cuestiones típicas son:

- *Volumen de los datos.* El usuario necesita especificar los volúmenes de transacciones de entrada y el tamaño requerido de los almacenes de datos. Esto es importante sobre todo si hay variaciones importantes en los volúmenes de transacciones (por ejemplo, durante horas pico del día, o épocas pico del año). Así, el usuario podría decir: "normalmente procesamos 2,000 pedidos diarios, pero el volumen salta a 4,000 pedidos diarios durante el mes de diciembre". Además, el usuario necesita estimar el incremento de tasas de transacción y requerimientos de almacenaje a lo largo de la vida útil estimada del sistema. Podría decir: "el almacén de INVENTARIO debe poder manejar balances de 4,000 partes distintas ahora, y esperamos estar manejando alrededor de 6,000 dentro de los próximos 5 años". En general, puede esperarse que la cantidad de datos almacenados en un sistema de información aumente en aproximadamente un 10 por ciento anual.[11]

- *Tiempo de respuesta a las diversas entradas.* Esto se puede plantear en términos absolutos, pero es más realista hacerlo en términos de probabilidades: "el 90 por ciento de todas las transacciones debe tener un tiempo de respuesta menor a 2 segundos". En algunos casos, esto puede indicarse en términos de límites de tiempo: "el reporte XYZ debe producirse a más tardar a las 8:00 cada mañana", o "todas las transacciones de depósitos deben procesarse antes de la media noche, diario, de modo que los clientes puedan determinar su saldo desde sus sistemas bancarios caseros".

11 Esta estimación se basa en una encuesta de aproximadamente 500 instalaciones de cómputo de los EUA hecha por Lientz y Swanson en Software Maintenance Management (Reading, Mass.: Addison-Wesley, 1980).

- *Restricciones políticas sobre modalidades de implantación.* El usuario pudiera, por motivos racionales o irracionales, especificar la marca de hardware que se usará (o que se evitará), el lenguaje de programación ("tiene que estar programado en Ada"), los proveedores de telecomunicaciones que se usarán, etc. Esto debe evitarse si fuera posible, pero espere por lo menos algunas presiones de este tipo. Note que es el departamento de operaciones de la organización el que puede imponer las restricciones de implantación; es decir, es probable que oír algo por el estilo de, "bueno, este nuevo sistema parece bien, pero desde luego que tiene que operar en la computadora principal de la corporación, así que asegúrese de que no ocupe más de 8 megabytes, y le asignaremos unos discos."

- *Restricciones ambientales.* El usuario que trabaja en el equipo de implantación pudiera imponer restricciones de temperatura, humedad, interferencia eléctrica (RFI), consumo de energía, limitaciones de tamaño, peso o emisiones eléctricas, vibración, contaminación, ruido, radiación, y otras restricciones ambientales. A veces no las mencionará explícitamente, sólo supondrá que el nuevo sistema operará de manera satisfactoria dentro de su ambiente normal (por ejemplo, una refinería, una fábrica o una oficina). Por tanto, podría ser necesario documentar las características relevantes del ambiente del usuario para beneficio del equipo de implantación. O tal vez simplemente indique en el modelo de implantación del usuario que el sistema debe operar en el ambiente del usuario, y dejar que el equipo de implantación deduzca por sí mismo lo que eso pueda significar.

- *Restricciones de seguridad y confiabilidad.* El usuario pudiera especificar un tiempo promedio entre fallas (MTBF, por sus siglas en inglés), y tiempo promedio necesario para la reparación (MTTR) para el sistema. La confiabilidad requerida también puede expresarse en términos de disponibilidad; por ejemplo, el usuario podría decir: "No es costeable algo inferior a un 99.5 por ciento de tiempo activo para este sistema".

- *Restricciones de seguridad.* El usuario puede especificar una variedad de restricciones enfocadas a minimizar el uso no autorizado del sistema. Esto puede incluir la consideración de números de cuenta y claves de acceso (para que los usuarios individuales tengan que identificarse). También puede incluir mecanismos para evitar el acceso no autorizado a datos confidenciales; algunos usuarios pueden tener permiso de leer registros de varios almacenes, mientras que otros sólo de modificar (o eliminar) datos existentes, y otros más pudieran sólo tener permiso de anexar registros nuevos. El usuario podría solicitar mecanismos para evitar que usuarios no autorizados realicen ciertas funciones en el sistema (por ejemplo, no todo mundo debiera poder operar el sistema de nómina). Se

podrían imponer diversas medidas de seguridad a los datos que entran o salen del sistema; esto incluye, por ejemplo, la codificación de datos que se transmiten por medio de líneas de telecomunicaciones.[12] Y, por motivos de seguridad, se pudiera requerir que el sistema produzca un rastreo de auditoría: listado completo de todas las transacciones que ingresan al sistema, las salidas que se producen, y tal vez incluso un registro de todas las modificaciones que se le hacen a los archivos.

21.5 RESUMEN

El modelo de implantación del usuario a veces se describe como la "zona fantasma" entre el análisis y el diseño estructurados. No lo puede hacer el analista solo, y es peligroso que él y el usuario lo desarrollen sin la participación de los diseñadores y programadores que finalmente construirán el sistema. Aunque las funciones, datos y comportamiento dependiente del tiempo sean finalmente las características más importantes de un sistema de información, la interfaz humana a menudo es el área que causa la mayor parte de las reacciones emocionales del usuario. Formatos de entrada difíciles, mensajes de error confusos y un tiempo de respuesta lento pueden volver inaceptables para el usuario incluso las funciones más elegantes del sistema. Y también recuérdese que las reatricciones de implantación que el usuario impone (a menudo de manera inocente) pueden torpedear incluso al proyecto mejor administrado: tal vez simplemente no sea posible implantar el sistema dentro de las restricciones señaladas por el usuario.

La labor de análisis del sistema queda concluida una vez construido y revisado el modelo de implantación del usuario por usuarios, analistas y el grupo de diseñadores y programadores. Al llegar a este punto, suele ser necesario presentar los resultados de la fase completa de análisis del proyecto a la administración, para obtener la aprobación de continuar con el diseño y la implantación. La presentación debe incluir la información siguiente:

1. El status actual del sistema existente (suponiendo que lo haya).
2. Los problemas (debilidades, funciones faltantes, etc.) que se identificaron en el sistema actual durante la encuesta inicial o el estudio de factibilidad.
3. Las soluciones alternativas que se identificaron.
4. Una vista global del modelo esencial y el de implantación del usuario, con tanto detalle como la administración requiera. Típicamente, se presenta el modelo de DFD de alto nivel y se proporcionan los componentes detallados para ser leídos con detenimiento posteriormente.

12 La seguridad en las computadoras es un tema primordial por sí mismo y no se discute con detalle en este libro. Para más información, consulte textos sobre seguridad y delitos computacionales.

5. Los costos y beneficios proyectados del nuevo sistema.[13]
6. Las estimaciones de costo, programación y recursos (horas de trabajo) para las fases restantes del proyecto.
7. Las recomendaciones del equipo que realiza el proyecto o del analista.

Suponiendo que se dé la aprobación de la administración, el proyecto mismo sólo está comenzando: todavía falta una gran cantidad de diseño, programación y prueba antes de que el usuario reciba finalmente el sistema que quería. Estas áreas se discuten en los siguientes capítulos.

REFERENCIAS

1. Marjorie Leeson, *Systems Analysis and Design.* Chicago: Science Research Associates, 1981.
2. Tom Gilb and Gerald Weinberg, *Humanized Input.* Cambridge, Mass.: Winthrop Publishers, 1977.
3. James Martin, *Design of Man-Machine Dialogues.* Englewood Cliffs, N.J.: Prentice-Hall, 1973.
4. Marvin Gore and John Stubbe, *Elements of Systems Analysis,* 3ª edición Dubuque, Iowa: william C. Brown Co., 1983.
5. *Data Processing Techniques: Coding Methods,* Forma GF20- 8093. White Plains, N.Y.: IBM Technical Publications Department.

PREGUNTAS Y EJERCICIOS

1. ¿Cuáles son las tres cosas principales que describe el modelo esencial de un sistema?
2. ¿Por qué no suele ser suficiente la información del modelo esencial para que los diseñadores y programadores comiencen a implantar el sistema?
3. ¿Qué información adicional necesita añadirse al modelo esencial?
4. ¿Qué es un modelo de implantación del usuario? ¿Cuáles son sus principales componentes?
5. ¿Cuáles son las dos cuestiones de implantación principales que generalmente preocupan mucho a los usuarios en un proyecto de un sistema de información?

13 Los cálculos de costo-beneficio se discuten en el Apéndice C.

6. Defina el concepto de frontera de automatización de un sistema. ¿Qué otro término o sinónimo puede usarse?

7. ¿Por qué se preocupan los usuarios por el formato de las entradas y salidas de un sistema de información?

8. Dé cuatro ejemplos de cuestiones de formato (que involucren entradas, salidas, o ambas) que el usuario desee especificar como parte del modelo de implantación del usuario.

9. Dé tres ejemplos de cuestiones de formato asociadas con los sistemas en línea que un usuario quisiera especificar como parte de modelo de implantación del usuario.

10. ¿Cómo afecta la introducción de las computadoras personales el trabajo que debe hacer el analista para desarrollar el modelo de implantación del usuario?

11. Dé tres ejemplos de preguntas que se necesite responder en el modelo de implantación del usuario si va a haber computadoras personales controladas por el usuario como parte de la implantación del sistema.

12. ¿Cómo afecta la introducción de lenguajes de cuarta generación en muchas organizaciones el trabajo que debe realizar el analista para desarrollar el modelo de implantación del usuario?

13. ¿Cómo afecta el concepto de creación de prototipos el desarrollo del modelo de implantación del usuario en un proyecto típico de desarrollo de sistemas?

14. ¿Cómo afecta la posible compra de paquetes de software comerciales el desarrollo del modelo de implantación de usuario en un proyecto típico de desarrollo de sistemas?

15. ¿Qué error cometen muchas organizaciones al desarrollar el modelo esencial en una situación donde esperan usar un paquete de software comercial?

16. ¿Cuáles son los tres casos extremos que pueden ocurrir cuando se está determinando la frontera de automatización en un sistema de información?

17. Bajo qué condiciones es probable que el usuario no se preocupe demasiado acerca de dónde vaya a quedar la frontera de automatización en un proyecto de desarrollo de sistemas? Qué tan probable cree que esto sea en una organización típica?

18. ¿Bajo qué condiciones es probable que el usuario se decida por un sistema completamente automatizado al estarse determinando la frontera de automatización, con todas las funciones realizadas por la computadora y todos los datos almacenados en forma computarizada?

19. ¿Bajo qué condiciones es probable que el usuario se decida por un sistema completamente manual al estarse determinando la frontera de automatización? ¿Qué tan probable cree que sea?

20. ¿Cuántas fronteras alternas de automatización cree que deba explorar con los usuarios el equipo que realiza el proyecto, antes de decidirse finalmente por alguna? Justifique su respuesta.

21. Desde el punto de vista del analista, ¿qué sería lo más sencillo que podría sucederle a los procesos y datos que se hayan colocado fuera de la frontera de automatización una vez determinada ésta?

22. ¿Qué es lo más probable que tenga que hacer el analista con los procesos y datos manuales después de determinarse la frontera de automatización?

23. ¿Cuáles son las tres cuestiones principales que se deben tratar cuando se define la frontera de automatización en el modelo de implantación del usuario?

24. ¿Dónde debe documentar el analista los detalles de la mayoría de las cuestiones de automatización que se discuten con el usuario?

25. Dé dos ejemplos de restricciones de implantación posibles respecto a algún dato que pueda determinarse como parte de la frontera de automatización.

26. ¿Cómo puede usarse de manera efectiva el diagrama de transición de estados durante el desarrollo del modelo de implantación del usuario?

27. ¿Qué tipo de actividades de apoyo manual podría requerirse especificar durante el desarrollo del modelo de implantación del usuario?

28. ¿Cuáles son los cinco tipos principales de restricciones operacionales posibles sobre un sistema que deben especificarse en el modelo de implantación del usuario?

29. ¿Por qué es importante especificar en el modelo de implantación del usuario el volumen de datos que el sistema debe manejar?

30. Dé tres ejemplos de restricciones políticas que puedan imponerse a un sistema como parte del modelo de implantación del usuario.

31. ¿El cajero automático de su banco es un sistema dirigido por menús o por órdenes? ¿Cuáles son las ventajas y desventajas del enfoque tomado por el sistema?

PARTE IV: SEGUIMIENTO

22 PASANDO AL DISEÑO

Para diseños compactos y ensambles enredosos,
utilícese alguna fuerza natural,
excepcionalmente desarrollada y no convencional,
que produzca sensación,
y continuamente en movimiento.
De enorme fuerza y no conformista,
sin consideración del fracaso.
Fuerza no restringida que al desarrollarse rápido extingue su forma original.

Adaptación de un poema de John Dryden,
Absalom and Achitophel, 1680.

En este capítulo se aprenderá:

1. Los tres niveles del diseño de sistemas.
2. Los tres criterios principales para evaluar el diseño de un sistema.
3. Cómo dibujar un diagrama de estructura.
4. Cómo usar el acoplamiento y la cohesion para evaluar un diseño.

Una vez completado el modelo de implantación del usuario concluye oficialmente la labor de análisis de sistemas. Más allá de ese punto, todo se vuelve cuestión de implantación. La parte visible de esta labor es la programación y la prueba, que discutiremos en el Capítulo 23. Sin embargo, la programación debe ir precedida por una actividad de nivel superior: el *diseño*.

Como analista, puede no sentir interés por los detalles del diseño de sistemas o de programas; sin embargo, como se vio en el capítulo anterior, la labor del analista y la del diseñador no siempre se pueden separar. Sobre todo en el área del modelo de implantación del usuario, el analista debe asegurarse de entender los requerimientos del usuario, mientras que el diseñador debe asegurar que dichos requerimientos se puedan implantar de manera realista con la tecnología computacional actual. Por ello, es importante entender el proceso que enfrenta el diseñador cuando el analista termina su labor.

Existe otra razón para tener interés en el diseño de sistemas: tal vez le toque hacerlo. Sobre todo en los sistemas pequeños y medianos, a menudo se espera que el mismo individuo documente los requerimientos del usuario y además desarrolle el diseño. Por ello, se puede esperar que decida sobre la mejor manera de asociar el modelo de los requerimientos del usuario en diferentes configuraciones de procesadores; tal vez tenga que decidir cómo implantar de la mejor manera el modelo lógico de datos (que se documentó con los DER) con un sistema de administración de bases de datos; y tal vez tenga que decidir cómo asignar a las funciones del sistema las distintas tareas dentro de cada procesador.

No es propósito de este libro discutir las actividades del diseño de sistemas con gran detalle; esto lo logran mejor los textos dedicados al tema, tales como [Page-Jones, 1988], [Yourdon y Constantine, 1989], [Ward y Mellor, 1985], [Jackson, 1975], [Orr, 1977], y otros. Sin embargo, examinaremos brevemente las principales etapas del diseño y algunos de los objetivos más importantes que el diseñador de sistemas debe tratar de lograr. Dado que el diseño de sistemas y de programas son de hecho materias en sí, definitivamente examine las referencias que se dan al final de este capítulo si requiere más información.

22.1 LAS ETAPAS DEL DISEÑO

La actividad de diseño involucra el desarrollo de una serie de *modelos*, de forma similar a la que el analista desarrolla modelos durante la fase de análisis de un proyecto. Los modelos específicos de diseño y su relación con los modelos de análisis que se discuten en este libro se ilustran en la figura 22.1.

Los modelos más importantes para el diseñador son el *modelo de implantación de sistemas* y el modelo de implantación de programas. El modelo de implantación de sistemas se divide luego en un modelo del *procesador*, y uno de *tareas*.

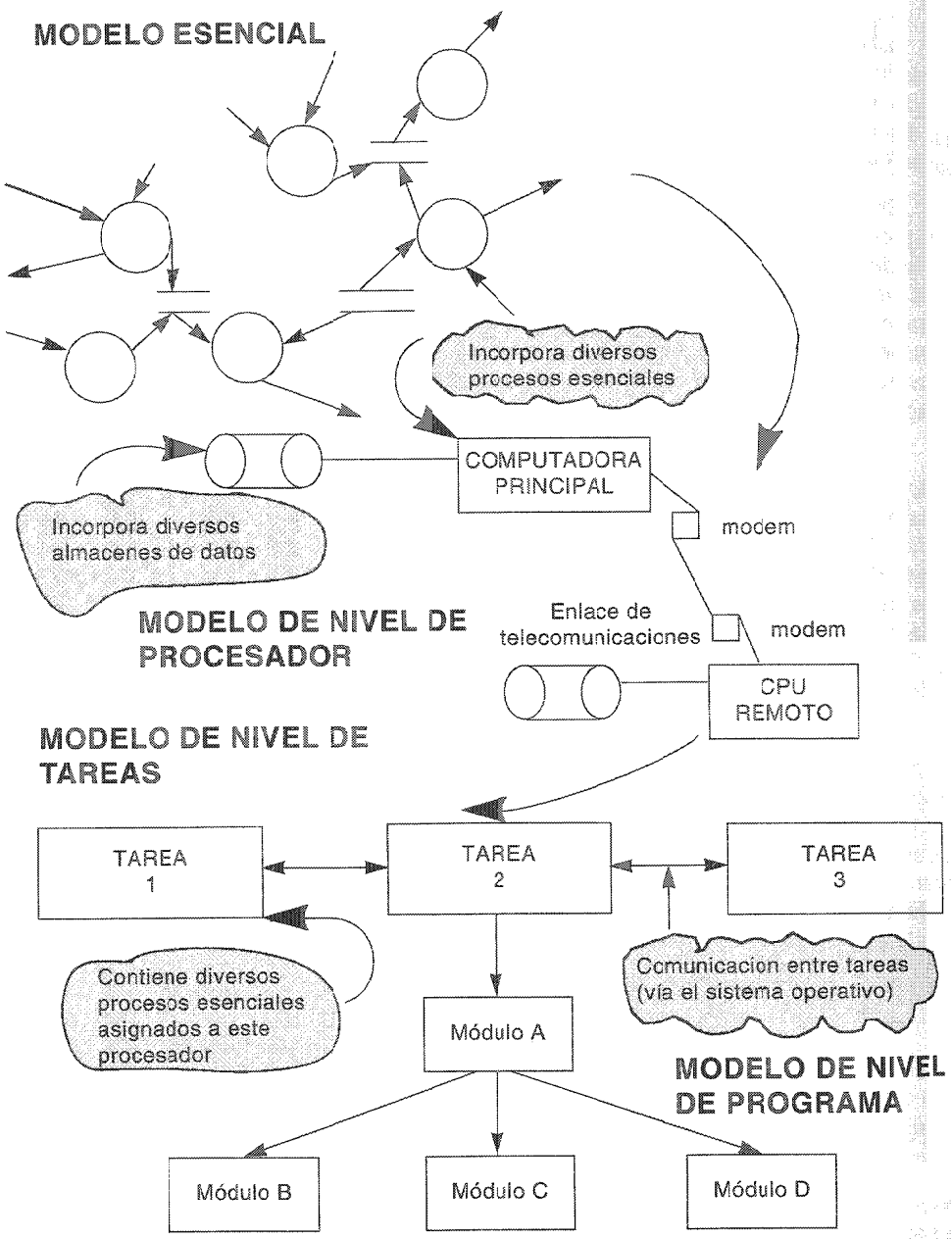


Figura 22.1: **Modelos de análisis y de diseño**

22.1.1 El modelo del procesador

La primera tarea que enfrenta el diseñador de sistemas es decidir cómo asignar el modelo esencial (o para ser más exactos, la parte automatizada del modelo de implantación del usuario) a las piezas principales de hardware y software del sistema. En el nivel del modelo del procesador, el diseñador del sistema trata principalmente de decidir cómo se asignar el modelo esencial a los distintos procesadores (CPU) y cómo deben comunicarse entre sí. Existe típicamente una variedad de opciones:

- El modelo esencial completo se le puede asignar a un solo procesador. Esto se conoce como la solución de computadora principal.
- Cada burbuja de la figura 0 del DFD del modelo esencial se puede asignar a un procesador distinto (normalmente una mini o microcomputadora).[1] Esto se conoce como la solución distribuida.
- Se puede escoger una combinación de computadoras principales, minis y micros para minimizar costos, maximizar confiabilidad o lograr algún otro objetivo.

Así como se deben asignar *procesos* a los componentes apropiados de hardware, los *almacenes de datos* se deben igualmente asignar. El diseñador debe decidir si un almacén se realizará como base de datos en el procesador 1 o el 2. Dado que la mayor parte de los almacenes se comparten entre muchos procesos, también debe decidir si se deben asignar copias del almacén a diferentes procesadores. La actividad de asignar procesos y almacenes a los procesadores se ilustra en la figura 22.2.

Observe que cualquier implantación diferente a la de un solo procesador involucrará algún mecanismo de comunicación entre procesadores; lo que tradicionalmente hemos mostrado como flujos de datos ahora debe especificarse en términos físicos. Algunas de las opciones disponibles al diseñador del sistema para comunicación de procesador a procesador son:

- Conexión directa entre procesadores. Esto puede implantarse conectando los procesadores mediante un cable, un canal o una red local. Este tipo de comunicación generalmente permite que los datos se transmitan de un procesador a otro a velocidades que van desde los 50,000 bits por segundo (abreviado como 50KB) a varios millones de bits por segundo.

---

1 Note que esto no es realista para algo que no sea un sistema trivial, dada la tecnología de computadoras de fines de los años 80. Si un sistema tuviera 500 burbujas de nivel inferior en su DFD de modelo esencial, ¿sería realista considerar la implantación del sistema con 500 procesadores separados? Esto cambiará para mediados de los años 90.

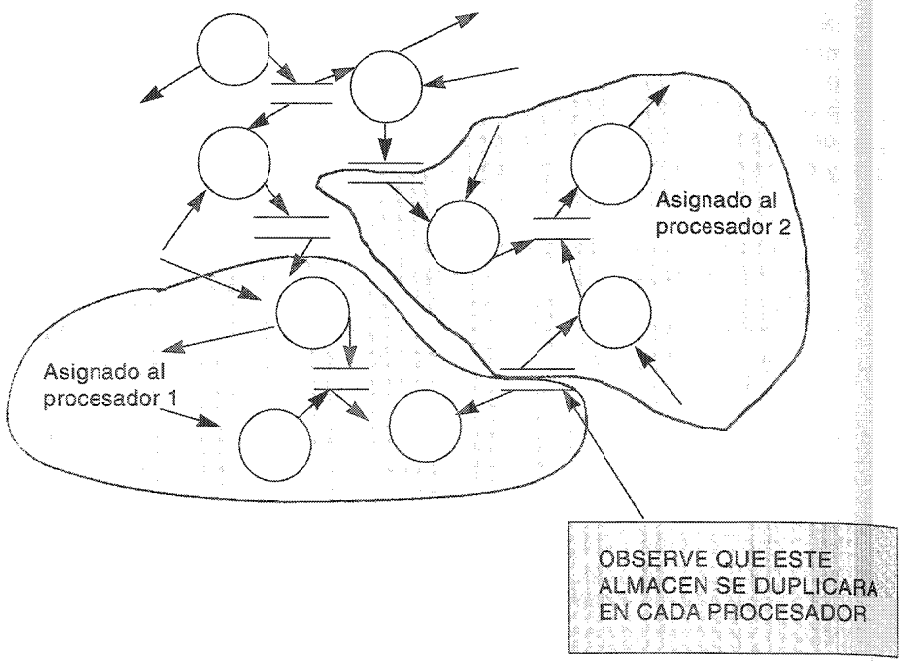


Figura 22.2: Asignación de procesos y almacenes a los procesadores

- Enlace de telecomunicaciones entre procesadores. Esto es común si los procesadores están separados físicamente por algunos cientos de metros. Dependiendo de la naturaleza del enlace de telecomunicaciones, se transmiten datos entre procesadores a velocidades que van desde 300 hasta 50,000 bits por segundo.

- Enlace indirecto entre procesadores. Los datos pueden escribirse en cinta magnética, disco flexible, tarjetas perforadas o algún otro medio de almacenamiento en un solo procesador y luego ser llevados físicamente al otro para ser empleados como entradas.

El último caso es un tanto extremo, pero ilustra un punto importante: la comunicación de procesador a procesador generalmente es mucho más lenta que la comunicación entre procesos (burbujas) dentro de un mismo procesador. Por tanto, el diseñador generalmente tratará de agrupar procesos y almacenes que tienen gran volumen de comunicación dentro del mismo procesador.

El diseñador debe tomar en cuenta varios factores al hacer estas asignaciones. Típicamente, las cuestiones principales son:

- *Costo.* Dependiendo de la naturaleza del sistema, pudiera ser o no ser más barata una implantación de un solo procesador. Para algunas aplicaciones, la solución más económica puede ser un grupo de microcomputadoras de bajo costo; para otras sería más práctico y económico hacer la implantación en la computadora principal existente en la organización.[2]

- *Eficiencia.* El diseñador de sistemas generalmente se preocupa por el tiempo de respuesta de los sistemas en línea y por la longitud del ciclo para los sistemas de cómputo por lote. Por tanto, debe escoger procesadores y dispositivos de almacenamiento de datos suficientemente rápidos y poderosos para satisfacer los requerimientos de desempeño en el modelo de implantación del usuario. En algunos casos puede escoger una implantación de múltiples procesadores para que las diferentes partes del sistema se ejecuten de manera paralela, acelerando así el tiempo de respuesta. Al mismo tiempo debe preocuparse por la ineficiencia de la comunicación de procesador a procesador, como se discutió anteriormente.

  Por ejemplo, suponga que el diseñador ve que el sistema contiene una función de ecición y una de proceso, como muestra la figura 22.3. Al poner cada función en un procesador aparte, sabe que se podrá editar una transacción mientras simultáneamente lleva a cabo el proceso de otra, mejorando así la eficiencia global del sistema. Por otro lado, las transacciones editadas se tendrán que mandar de un procesador a otro y esto puede ser muy eficiente si se hace a través de una conexión directa, o puede ser muy ineficiente si la comunicación se realiza mediante líneas de telecomunicación lentas.

- *Seguridad.* El usuario final podría tener requerimientos de seguridad que dicten que algunos (o todos) los procesadores y/o datos delicados se coloquen en lugares protegidos. Estos requerimientos también dictaminan la naturaleza (o la ausencia) de la comunicación de procesador a procesador. Por ejemplo, el diseñador podría estar impedido de transmitir datos de un procesador a otro mediante líneas telefónicas ordinarias si la información es confidencial.

---

2 Tenga en mente que existe un presupuesto para todo el proyecto, que debió determinarse como parte del proceso de análisis (ver el Capítulo 5). Por ello, el diseñador debe escoger el sistema más eficiente que se ajuste al presupuesto. Sin embargo, tenga en mente el hecho de que los presupuestos pueden cambiar: los que se desarrollaron durante la fase de análisis del proyecto fueron sólo estimaciones y pueden estar sujetas a revisión si el diseñador muestra que se necesita más dinero para lograr una implantación aceptable.

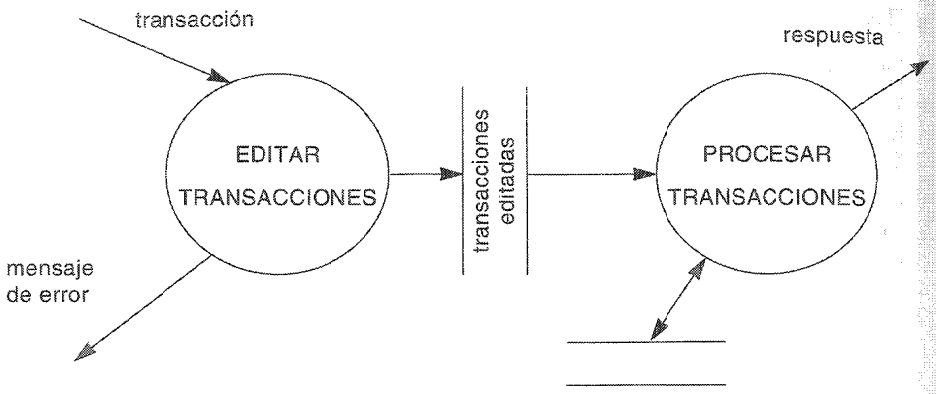


Figura 22.3: **Comunicación de procesador a procesador**

- *Confiabilidad.* El usuario final típicamente especifica los requerimientos de confiabilidad para un nuevo sistema. Estos requerimientos pueden expresarse en términos de tiempo promedio entre fallas (MTBF), tiempo promedio de reparación (MTTR) o disponibilidad del sistema.[3] En todo caso, esto podría tener influencia dramática sobre el tipo de configuración de procesadores que se escoja. Podría decidirse separar los procesos en diferentes procesadores para que haya siempre una porción del sistema disponible, incluso si otras partes se vuelven inoperables por fallas de hardware. Como alternativa, se puede decidir tener copias redundantes de procesos y/o datos en múltiples procesadores, tal vez incluso con procesadores extra que pueden usarse en caso de falla. Esto se muestra en la figura 22.4; incluso si llega a fallar el procesador activo (lo cual es tal vez más probable aún, dado que se trata de una computadora principal grande y compleja), los procesadores individuales de edición pueden continuar operando, recolectando transacciones, editándolas y almacenándolas para procesarlas posteriormente. De manera similar, si se descompone uno de los procesadores de edición, los demás pueden continuar operando.

- *Restricciones políticas y operacionales.* La configuración de hardware puede verse influenciada también por restricciones políticas impuestas directamente por el usuario final, por otros niveles de administración dentro

3 La disponibilidad del sistema usualmente se define como el porcentaje de tiempo en el que está disponible. Puede calcularse con base en el MTBF y el MTTR de la siguiente forma:

Disponibilidad = MTBF / (MTBF + MTTR).

de la organización o por el departamento de operaciones a cargo del mantenimiento y operación de todos los sistemas de cómputo. Esto puede llevar a la elección de una configuración específica de hardware, o excluir la elección de ciertos proveedores. De manera similar, se pueden presentar restricciones ambientales (por ejemplo, temperatura, humedad, exposición a radiaciones, polvo/tierra, vibraciones), y esto puede tener una influencia enorme sobre la configuración de procesadores que se escoja.

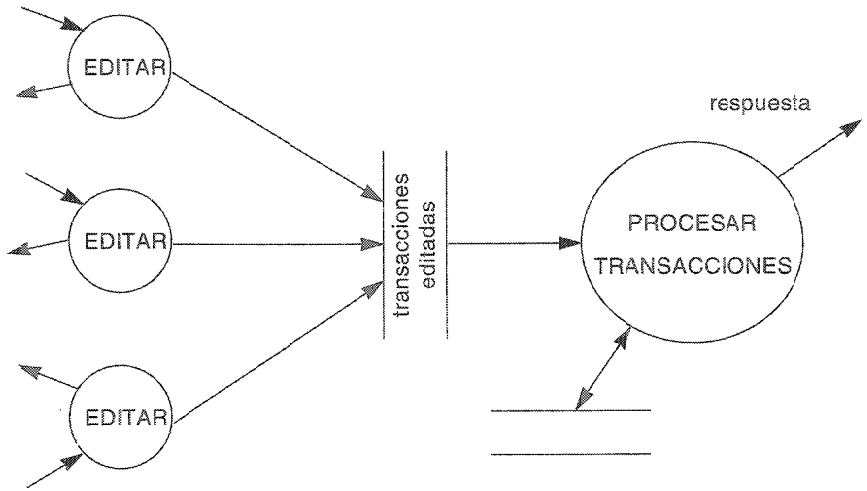


Figura 22.4: **Procesadores múltiples para mayor confiabilidad**

22.1.2 El modelo de tareas

Una vez que se han asignado procesos y almacenes a los procesadores, el diseñador debe, procesador por procesador, asignar procesos y almacenes a las tareas individuales de cada uno. La noción de tarea es común a casi cualquier marca de hardware de computadora, aunque la terminología difiera de un proveedor a otro: algunos usan el término partición y otros punto de control. Sin importar el término, la figura 22.5 muestra cómo divide un procesador típico su espacio de almacenamiento disponible en áreas separadas, donde cada una se administra con un sistema operativo central. El diseñador generalmente tiene que aceptar el sistema operativo del proveedor (aunque tal vez tenga posibilidad de escoger entre diversos sistemas operativos para una computadora dada), pero sí tiene la libertad de decidir cuáles porciones del modelo esencial asignadas a dicho procesador deben asignarse a tareas individuales dentro de éste.

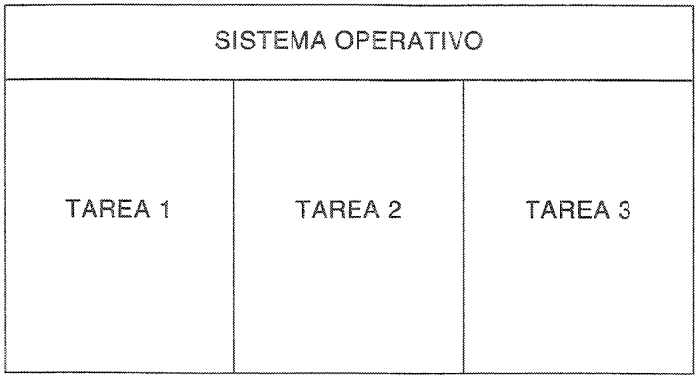


Figura 22.5: **Organización de las tareas dentro de un procesador**

Observe que los procesos dentro de un mismo procesador pueden tener necesidad de comunicarse mediante alguna forma de protocolo de comunicación entre tareas. El mecanismo para hacerlo varía de un proveedor a otro, pero sucede de manera casi universal que la comunicación se realiza a través del sistema operativo del proveedor, como ilustra la figura 22.6. Así como la transmisión de datos de un procesador a otro es relativamente lenta e ineficiente, la comunicación de datos (o señales de control) de una tarea a otra dentro del mismo procesador también es ineficiente. La comunicación entre procesos en la misma tarea usualmente es más eficiente. Por eso, el diseñador generalmente trata de mantener los procesos con mayor volumen de comunicación dentro de la misma tarea.

Dentro de un procesador individual, no siempre está claro si las actividades ocurren de manera sicronizada o no; es decir, no siempre queda claro si está sucediendo una cosa o muchas a la vez. Típicamente, cada procesador individual sólo tiene un CPU, que puede estar ejecutando instrucciones para un proceso a la vez; sin embargo, si un proceso está esperando entradas o salidas provenientes de un dispositivo de almacenamiento (por ejemplo, disco, cinta, terminal de video, etc.), el sistema operativo del procesador puede pasarle el control a otra tarea. Por tanto, el diseñador puede considerar cada tarea como una actividad independiente no sincronizada.

22.1.3 El modelo de implantación de programas

Finalmente llegamos al nivel de una tarea individual; hasta aquí el diseñador ya logró completar dos niveles de asignación de procesos y almacenamiento de datos. Dentro de una tarea individual, la computadora opera de una manera *no sincro-*

nizada: sólo se puede llevar a cabo una actividad a la vez. El modelo más común de organización de la actividad en una sola unidad sincronizada es el *diagrama de estructura,* que muestra la organización jerárquica de módulos dentro de una tarea. La figura 22.7 muestra los principales componentes de un diagrama de estructura.

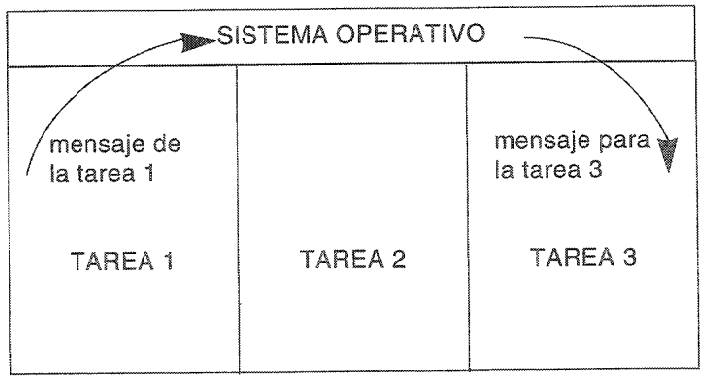


Figura 22.6: **Comunicación entre tareas dentro de un procesador**

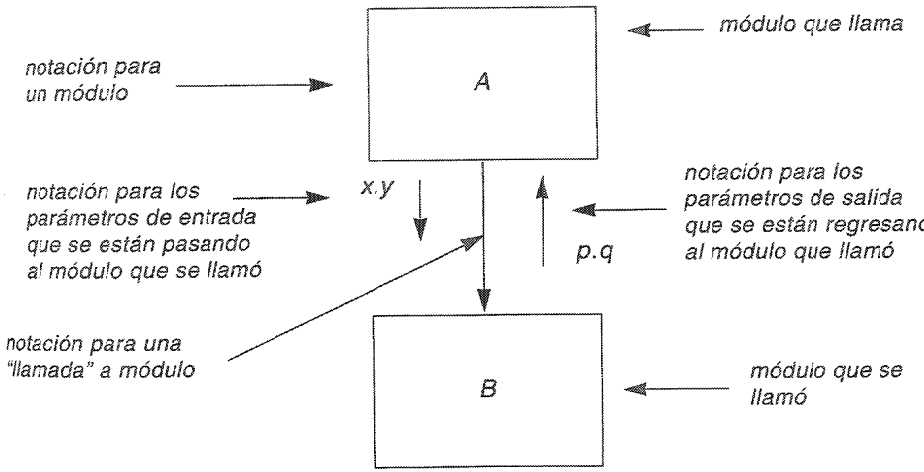


Figura 22.7: **Componentes de un diagrama de estructura**

Debe leerse este pequeño diagrama de estructura de la forma siguiente:

- El módulo A es el módulo ejecutivo del nivel superior del sistema que consta de los módulos A y B. La razón por la cual A se identifica como el módulo de nivel superior no *es* porque esté topológicamente por encima del módulo B, sino porque ningún otro módulo lo llama. El módulo B, por otro lado, se llama subordinado del módulo A. (El módulo A es llamado o invocado por el sistema operativo de la computadora).

- El módulo A contiene una o más instrucciones ejecutables, incluyendo una llamada al módulo B. Esta llamada puede hacerse como una declaración CALL en lenguaje FORTRAN. O una declaración PERFORM o CALL USING de COBOL. O simplemente invocando el nombre de B en otros lenguajes. El diagrama de estructura evita deliberadamente describir cuántas veces llama el módulo A al B. Eso depende de la lógica interna del programa dentro del módulo A. Por tanto puede haber una instrucción del siguiente tipo dentro del módulo A:

```
SI comienza-guerra-nuclear
    LLAMA Módulo-B
EN OTRO CASO
    ...
```

  en cuyo caso el módulo B pudiera no llamarse jamás. Pero también puede existir una instrucción del siguiente tipo en el módulo A:

```
HACER MIENTRAS haya más pedidos en el archivo
PEDIDOS
    LLAMA Módulo B
FIN
```

  en cuyo caso el módulo B puede llamarse miles de veces.

- Cuando se llama al módulo B, la ejecución del módulo A se suspende. El módulo B se empieza a ejecutar en su primera declaración ejecutable. Cuando termina, sale o regresa al módulo A. El módulo A continúa entonces su ejecución en el punto donde la suspendió.

- El módulo A puede o no pasar parámetros de entrada al módulo B como parte de la llamada, y el módulo B puede regresar o no parámetros de salida cuando regrese al módulo A. En el ejemplo que se muestra en la figura 22.7, el módulo A pasa los parámetros X y Y al módulo B, y éste le regresa los parámetros P y Q. Las definiciones detalladas de X, Y, P y Q normalmente se deben encontrar en un diccionario de datos. La mecánica de la transmisión de los parámentros varía de un lenguaje de programación a otro.

En la figura 22.8 se muestra un ejemplo de un diagrama de estructura completo. Note que contiene cuatro niveles de módulos; esto normalmente representaría un programa de alrededor de quinientas a mil instrucciones, suponiendo que cada módulo representa alrededor de cincuenta a cien.[4]

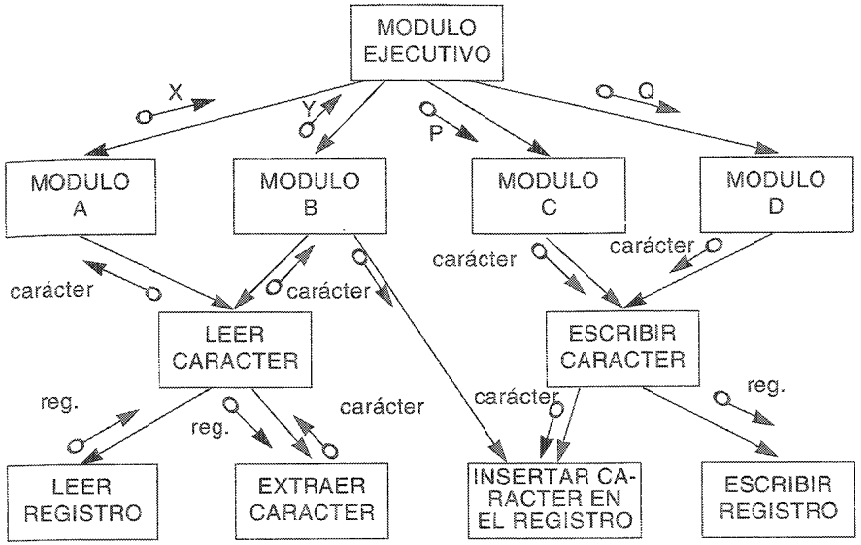


Figura 22.8: **Ejemplo de diagrama estructurado**

Existe una pregunta obvia al llegar aquí: ¿Cómo transforma el diseñador un modelo de red de procesos en el diagrama de flujo de datos en el modelo sincronizado representado por el diagrama de estructura? Varios textos sobre diseño de sistemas, incluyendo [Page-Jones, 1988] y [Yourdon y Constantine, 1989], discuten esta cuestión con gran detalle. Como ilustra la figura 22.9, hay una estrategia de recetas para transformar el modelo de red de flujo de datos en un modelo de diagrama de estructura sincronizado; de hecho, la estrategia generalmente se conoce como diseño centrado en la transformación. Esta es tan sólo una de diversas estrategias para convertir un modelo de red de flujo de datos en un modelo jerárquico sincronizado; [Page-Jones, 1988], [Yourdon y Constantine, 1989] y [Ward y Mellor, 1985] discuten varias estrategias de éstas. Note que cada burbuja de proceso en el diagrama de flujo de la figura 22.9 se convierte en un módulo en el diagrama de estructura deriva-

4 Desde luego, un módulo llamado EXTRAER CARACTER no suena como si requiriera de 50 a 100 instrucciones; tal vez requiera sólo dos o tres en un lenguaje programación de alto nivel típico. En un nivel de lenguaje cercano a la máquina, sin embargo, típicamente se requerirían muchas más.

do; ésta es una situación realista si los procesos son relativamente pequeños y simples (por ejemplo, si la especificación del proceso ocupa menos de una página de lenguaje estructurado). Además del módulo que realiza los procesos de flujo de datos, es evidente que el diagrama de estructura también contiene módulos destinados a coordinar y administrar la actividad global, y módulos que se encargan de traer entradas al sistema y obtener salidas de él.

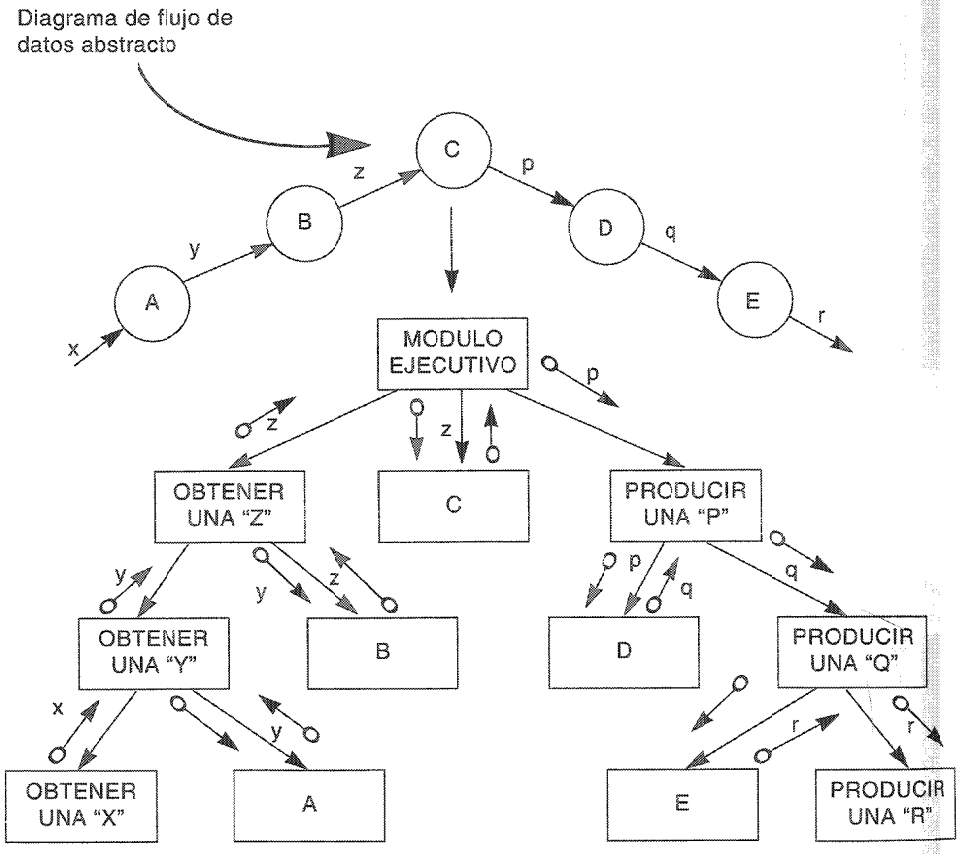


Figura 22.9: **Estrategia de diseño centrada en transformaciones**

Otras estrategias de diseño utilizan el diagrama de entidad-relación u otras formas de diagramas de estructura de datos como punto de partida para obtener el diagrama de estructura apropiado; véase [Jackson, 1975] y [Orr, 1977] para más información acerca de estas estrategias de diseño.

22.2 METAS Y OBJETIVOS DEL DISEÑO

Además de lograr los objetivos que se especifican en el modelo de implantación del usuario, el diseñador también se ocupa de la calidad global del diseño. La capacidad que los programadores exhiban para implantar un sistema de alta calidad y libre de errores depende en gran medida de la naturaleza del diseño; de manera similar, la capacidad de los programadores de mantenimiento para realizar cambios en el sistema después de haberlo puesto en operación depende de la calidad del diseño.

El campo del diseño estructurado ofrece guías para ayuda al diseñador a determinar los módulos, y sus interconexiones, que mejor realizarán los requerimientos especificados por el analista; todos libros que se ennumeran al final de este capítulo detallan estas pautas. Las dos reglas más importantes son las referentes al acoplamiento y la cohesión; a continuación se discuten éstas y algunas otras reglas comunes.

- *Cohesión.* Grado en el cual los componentes de un módulo (típicamente las instrucciones individuales que conforman un módulo) son necesarios y suficientes para llevar a cabo una sola función bien definida. En la práctica, esto significa que el diseñador debe asegurarse de no fragmentar los procesos esenciales en módulos, y también debe asegurarse de no juntar procesos no relacionados (que se representan por burbujas en el DFD) en módulos sin sentido. Los mejores módulos son aquéllos que son *funcionalmente cohesivos* (es decir, módulos en los cuales cada instrucción es necesaria para poder llevar a cabo una sola tarea bien *definida).* Los peores módulos son los que son *coincidentalmente cohesivos* (es decir, cuyos cuyas instrucciones no tienen una relación significativa entre uno y otro).[5]

- *Acoplamiento.* Grado en el cual los módulos se interconectan o se relacionan entre ellos. Entre más fuerte sea el acoplamiento entre módulos en un sistema, más difícil es implantarlo y mantenerlo, pues entonces se necesitará un estudio cuidadoso para la modificación o cambio y modificación de algún módulo o módulos. En la práctica, esto significa que cada módulo debe tener interfases sencillas y limpias con otros, y que se

5 Algunos ejemplos de módulos funcionalmaente cohesivos son CALCULAR-RAIZ-CUADRADA, CALCULAR-SALARIO-NETO y VALIDAR-DOMICILIO-DEL-CLIENTE. Un ejemplo de uno coincidentalmente cohesivo es FUNCIONES-MISCELANEAS.

debe compartir un número mínimo de datos entre módulos. También significa que un módulo dado no debe modificar la *lógica interna* o los datos de algún otro módulo; lo que se conoce como una conexión patológica. (La temida declaración ALTER de COBOL es un buen ejemplo.)

- *Tamaño del módulo.* De ser posible, cada módulo debe ser lo suficientemente pequeño como para caber en una sola página (o para que pueda desplegarse en una sola pantalla). Desde luego, a veces no es posible determinar qué tan grande va a ser un módulo hasta haberlo escrito, pero las actividades iniciales de diseño a menudo darán al diseñador una buena pista de que el módulo va a ser grande y complejo. Si es así, debe partirse en uno o más niveles de submódulos. (En raras ocasiones, los diseñadores crean módulos que son triviales. Por ejemplo, módulos que consisten en sólo dos o tres renglones de código. En este caso, pueden juntarse varios en un solo supermódulo mayor.)

- *Alcance del control.* El número de subordinados inmediatos que un módulo administrador puede llamar se conoce como el alcance del control. Un módulo no debe poder llamar a más de una media docena de módulos de nivel inferior. La razón es evitar la complejidad: si el módulo tiene, digamos, 25 módulos de nivel inferior, entonces probablemente contendrá tanta lógica compleja de programa (en la forma de declaraciones **SI** anidadas, o de iteraciones **HACER-MIENTRAS** anidadas, etc.) que nadie lo podrá entender. La solución es introducir un nivel intermedio de módulos administradores, como haría un administrador de una organización humana si se ve en la necesidad de tratar de supervisar directamente a 25 subordinados inmediatos.[6]

- *Alcance del efecto/alcance del control.* Esta regla sugiere que cualquier módulo afectado por el resultado de alguna decisión debe ser subordinado (aunque no necesariamente un subordinado inmediato) del módulo que toma la decisión. Es un tanto análogo a la regla de administración que dice que cualquier empleado afectado por los resultados de la decisión de algún administrador (es decir, dentro del alcance del efecto de la decisión) debe estar dentro del alcance de control del administrador (es decir trabajando entre la jerarquía de personas que se reportan con el administrador). Violar esta regla en un ambiente de diseño estructurado usualmente lleva paso innecesario de banderas y condiciones (lo cual incrementa el acoplamiento entre módulos), la toma redundante de decisiones o (en el peor de los casos) conexiones patológicas entre módulos.

6 Existe una excepción a esto conocida, como centro de transacciones. Si el módulo administrador toma una sola decisión para invocar a uno solo de sus subordinados, entonces su lógica probablemente es bastante sencilla. En este caso, no nos tenemos que preocupar acerca del alcance de control.

22.3 RESUMEN

Hay mucho más que aprender acerca del diseño, pero con esta introducción debe entenderse el proceso por el que pasa el diseñador. Como hemos visto, el primer paso es hacer coincidir el modelo esencial de los requerimientos del ususario con una configuración de procesadores. Luego, dentro de cada procesador, el diseñador debe decidir cómo asignar procesos y datos a diferentes tareas. Finalmente, deben organizarse los procesos dentro de cada tarea en una jerarquía de módulos, utilizando el diagrama de estructura como herramienta.

Observe también que probablemente se tendrán que añadir procesos adicionales y reservas de datos a la implantación del modelo para considerar las características específicas de la tecnología de implantación. Por ejemplo, pueden requerirse procesos adicionales para revisión de errores, edición y actividades de validación que no se mostraron en el modelo esencial; y para transportar flujos de datos entre procesadores podrían ser necesarios también otros procesos. Una vez logrado esto puede comenzar la programación. Los temas de programación y prueba se discuten en el Capítulo 23.

REFERENCIAS

1. Meilir Page-Jones, *The Practical Guide to Structured Systems* Design, 2ª edición, Englewood Cliffs, N.J.: Prentice-Hall, 1988.

2. Edward Yourdon y Larry L. *Constantine, Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design.* Englewood, N.J.: Prentice-Hall, 1989.

3. Paul Ward y Steve Mellor, *Structured Development for Real-Time Systems,* volumen 3. Nueva York: YOURDON Press, 1986.

4. Michael Jackson, *Principles of Program Design.* Nueva York: Academic Press, 1975.

5. Ken Orr, *Structured Systems Development.* Nueva York: YOURDON Press, 1977.

PREGUNTAS Y EJERCICIOS

1. ¿Qué actividad sigue al desarrollo del modelo de implantación del usuario en un proyecto típico del desarrollo de sistemas?

2. ¿Cuáles son las tres principales etapas del diseño en un proyecto típico de desarrollo de sistemas? ¿Qué modelos se desarrollan durante estas tres etapas?

3. ¿Por qué son importantes los modelos durante la fase de diseño de un proyecto?

4. ¿Cúal es el principal propósito del modelo del procesador durante la actividad de diseño?

5. Dé tres ejemplos de cómo pueden hacerse coincidir los procesos de un modelo esencial con los procesadores en un modelo de implantación.

6. ¿Qué decisiones deben tomarse, durante la actividad de modelado del procesador, sobre de los almacenes de datos que se identificaron en el modelo esencial?

7. Enumere tres métodos comunes para la comunicación entre procesadores.

8. ¿Qué factores debe tomar en cuenta el diseñador cuando escoge alguno de estos tres métodos? ¿Cuál de estos factores cree que sea el más importante?

9. Si está trabajando en un proyecto de desarrollo de sistemas donde la *confiabilidad* es prioritaria, ¿cómo afectaría esto su decisión acerca de la asignación de procesos y almacenes esenciales a diferentes procesadores ?

10. Dé un ejemplo de cómo pueden las restricciones políticas influir en la asignación de tareas y almacenes esenciales a diferentes procesadores.

11. ¿Qué es un modelo de tareas en el contexto de este capítulo? ¿Cuáles son sus componentes?

12. Dé tres sinónimos comunes de tarea.

13. ¿Bajo qué circunstancias pudieran estar operando diferentes tareas al mismo tiempo?

14. Proyecto de investigación: Escoja una computadora y un sistema operativo comunes. Describa cómo pueden comunicarse entre sí las diferentes tareas que operan bajo el control del sistema operativo ¿Cuál es el sobrecosto típico (en términos de tiempo de CPU, utilización de memoria y otros recursos importantes de hardware) de dicha comunicación entre tareas?

15. Dé una definición de modelo de implantación de programa. ¿Cuáles son sus componentes?

16. ¿Cómo debe transformar el diseñador un modelo esencial de DFD orientado a redes no sincronizado, en un modelo jerárquico sincronizado?

17. ¿Bajo qué condiciones se convierte cada burbuja del modelo esencial en un módulo del modelo de implantación de programa?

18. Enumere dos estrategias comunes de diseño. Dé una breve descripción de cada una.

19. ¿Cuál es el objetivo primario que trata de alcanzar el diseñador cuando traduce el modelo esencial a un modelo de implantación?

20. ¿Qué otros objetivos trata de alcanzar el diseñador cuando crea un modelo de implantación?

23 PROGRAMACION Y PRUEBA

Es imposible disociar el lenguaje de la ciencia, o la ciencia del lenguaje, porque toda ciencia natural involucra siempre tres cosas: la secuencia de los fenómenos sobre los cuales se basa la ciencia; los conceptos abstractos que traen dichos fenómenos a la mente, y las palabras con las cuales se expresan los conceptos. Para llamar a un concepto se necesita una palabra; para describir un fenómeno, se necesita un concepto. Las tres cosas reflejan una sola realidad.

Antoine Laurent Lavoisier
Traité Elementaire de Chimie, 1789

Lo que tenemos cue hacer es siempre estar probando con curiosidad nuevas opiniones y cortejando nuevas impresiones.

Walter Pater, *El Renacimiento*, 1873

En este capítulo se aprenderá:

1. El papel del analista de sistemas en la programación y la prueba.
2. Por qué es ventajoso el seguimiento rápido durante la programación y prueba.
3. Lo que el analista debe buscar al examinar un programa.
4. Los principales tipos de prueba que se deben realizar.

La programación y la prueba normalmente comienzan, como pudiera esperarse, cuando termina la actividad de diseño. La fase de programación o implantación de un proyecto típico involucra la escritura de instrucciones en COBOL, Pascal o algún otro lenguaje de programación para implantar lo que el analista ha especificado y el diseñador ha organizado en módulos. La prueba, como el nombre implica, involucra ejercitar el sistema para asegurar que produzca las salidas apropiadas y exhiba el comportamiento adecuado para una gama amplia de entradas.

¿Por qué debiera interesarle esto como analista? ¿Acaso no es verdad que no estará involucrado con el proyecto a estas alturas? No necesariamente. Por varias razones, la labor de los programadores y probadores puede influir en su *trabajo;* y la forma en que usted organice su trabajo puede influir en el de ellos. La interrelación entre el análisis de sistemas y la programación/prueba es el tema de este capítulo.

23.1 EL PAPEL DEL ANALISTA EN LA PROGRAMACION Y LA PRUEBA

En el caso extremo, el analista terminará su labor de especificación del sistema y luego pasará algún tiempo con el equipo de diseño mientras se desarrolla el modelo de implantación del usuario y se llevan a cabo las primeras etapas del diseño. Pero para cuando inicia la primera etapa de programación, el analista puede haber comenzado con otro proyecto. Hay algunas razones por las cuales usted, como analista, puede requerir seguir involucrado en el proyecto al comenzar la actividad de programación:

- *Una razón sencilla.* Usted es el líder del proyecto y está a cargo de los programadores. Obviamente, no los puede abandonar. Estará involucrado con el proyecto hasta la prueba final, la aceptación y la entrega al usuario final. Por ello, será importante que sepa si los programadores escribieron código de alta calidad, y será igualmente importante que sepa si probaron de manera adecuada sus programas.
- *Otra razón sencilla.* Usted es un analista junior y su título es de programador/analista o analista/programador. Por ello, además de desarrollar las especificaciones del sistema, estará involucrado en la escritura de programas.
- *Una razón más interesante.* Usted forma parte de un grupo que escribe casos de prueba que se usarán para ejercitar los programas que los programadores escriben. En muchos proyectos, tal vez uno o más usuarios se le unan en esta actividad, partiendo de la teoría de que los usuarios son los más aptos para pensar en casos excepcionales y poco usuales de prueba. El desarrollo de datos de prueba puede empezar tan pronto como se termina la especificación (de hecho, incluso antes de haberla terminado por completo); como lo dice Tom DeMarco, "La especificación es la prueba del sistema". Dado que por lo pronto sólo conoce el contenido ló-

gico de las entradas y salidas, probablemente tendrá que esperar hasta que el modelo de implantación del usuario quede terminado antes de poder determinar el formato físico de dichas entradas y salidas. También necesitará el modelo de implantación del usuario para conocer las restricciones operacionales (tiempo de respuesta, volúmenes, etc.) que se necesitan probar. Pero esta actividad fácilmente lo puede tener ocupado hasta el final del proyecto porque si los programas fallan con sus casos de prueba, necesitará trabajar con los programadores para determinar si el caso de prueba está mal, o si es el código de ellos.

- *Una razón menos obvia.* Puede verse involucrado en el desarrollo de manuales de usuario, preparación de los usuarios o en la planeación de la instalación del nuevo sistema y conversión de datos desde el otro sistema. En la mayor parte de los casos, esto puede llevarse a cabo de manera paralela con la programación y prueba del nuevo sistema. Siendo usted el analista que estará involucrado con el proyecto desde el principio, a menudo se le considerará el candidato ideal para este trabajo.

- *Una razón descorazonadora.* Tal vez los programadores no comprendan su especificación. O su especificación puede estar incompleta, ser inconsistente, o ser contradictoria. Pero estas cosas suelen suceder, y puede encontrarse con el hecho de que los programadores necesitan consultarlo periódicamente para revisar y aclarar la especificación cuando la traducen a un lenguaje de programación. Otra variante sería que le pidieran cambiar la especificación por ser demasiado difícil de implantar. En este caso, desde luego, tendrá que ser el mediador (además de intérprete) entre los programadores y los demás analistas de sistemas.

- *Otras razones descorazonadoras.* Puede ser que los usuarios hayan comenzado a cambiar de opinión con respecto a los requerimientos, incluso cuando los programadores están implantando los que decían querer. Aparte del hecho de que algunos usuarios gustan de hacer esto por diversión, existen algunas buenas razones para ello; los usuarios viven en un mundo dinámico y a menudo deben reaccionar a una política cambiante que les impone la legislación gubernamental, los requerimientos de sus clientes o las condiciones generales del mercado. Por tanto, puede encontrarse con que está cambiando la especificación incluso cuando los programadores ya están implantando la especificación, lo cual no hará feliz a nadie, pero tiene que hacerse de todos modos. Esto se discute más a fondo en la sección 23.4.

23.2 EL IMPACTO DEL ANALISIS, LA PROGRAMACION Y LA PRUEBA SOBRE LA ESTRUCTURA ORGANIZACIONAL

A lo largo de este libro ha sido evidente que el análisis estructurado involucra una progresión continua desde los aspectos de modelado de alto nivel (por ejemplo, diagramas de flujo de datos de nivel superior) a aspectos de modelado de bajo nivel como el desarrollo de especificaciones de proceso y el diccionario de datos completo y detallado. De manera similar, el proceso de diseño involucra el desarrollo de modelos de diseño que van desde diagramas de estructura de alto nivel hasta formas de nivel tan bajo como el pseudocódigo y los diagramas de flujo. La programación debe seguir este mismo patrón: se escriben programas para los módulos ejecutivos de alto nivel, y tarde o temprano se desarrollarán para los módulos de bajo nivel que llevan a cabo cálculos detallados, validan datos de entrada, etc.

Lo que aún no hemos discutido es la relación que existe entre los niveles del sistema y los niveles de la *organización* que construye el sistema. Pero probablemente le habrá dado la impresión, tras leer la mayor parte de este libro, que los que ostentan el título de analistas de sistemas son los que tienen la responsabilidad de todo el trabajo de análisis de sistemas; que quienes tienen el título de diseñadores son responsables de la labor de diseño, y los que tienen el de programador serán responsables de escribir los programas.

Pero existe un problema con este enfoque; de hecho, dos problemas relacionados. Primero, quienes tienen el título de analistas usualmente son personas relativamente maduros, con varios años de experiencia. A pesar de que generalmente disfrutan la labor de dibujar diagramas de flujo de datos y diagramas de entidad-relación, les resulta difícil emocionarse ante la perspectiva de tener que escribir cientos de especificaciones de proceso y definir miles de datos.

Y también está el otro lado del problema: si los analistas en jefe realizan esta labor detallada, ¿qué hacen los programadores? Su labor se vuelve casi mecánica, y consiste básicamente en traducir las especificaciones a COBOL o FORTRAN. Cualquier creatividad que hayan pensado que su trabajo podía tener desaparece.[1]

Una solución para este dilema aparente es permitir a esta gente madura hacer todas las actividades de nivel superior del proyecto, y a los novatos jóvenes todas las actividades detalladas de nivel inferior. Esto significa, por ejemplo, que el personal con experiencia (los que tienen el título de analista en jefe o algo igualmente impresionante) no sólo haría las actividades de alto nivel del análisis (como el dibujo de los diagramas de flujo de datos y otros) sino también las actividades de diseño de alto nivel, e incluso llegar a escribir código de alto nivel. Los novatos, mientras tan-

[1] En realidad, las cosas pudieran ser aún peores o un poco mejores. La peor situación (desde el punto de vista del programador) sería que los programadores no se necesitaran en absoluto. Si la especificación del proceso se escribe en un lenguaje lo suficientemente formal, se puede compilar sin necesidad de intervención humana. Esto ya está sucediendo en casos aislados (por ejemplo, especificaciones del proceso escritas en lenguaje Ada y luego compiladas directamente). Por otro lado, existe la posibilidad de que el programador todavía tenga mucho trabajo creativo que hacer si el analista escribe la especificación usando el enfoque de precondición/postcondición que se discute en el Capítulo 11. En este caso, el analista habrá especificado las entradas y salidas para cada módulo, pero dejó al diseñador y, finalmente, al programador, la labor de determinar el mejor algoritmo.

to, estarían involucrados en el proyecto desde el principio (o tan pronto como los jefes hayan completado los aspectos de alto nivel del análisis) y participarían en el trabajo de escribir las especificaciones de procesos y módulos, en desarrollar entradas para el diccionario de datos y escribir el código para los módulos de nivel inferior.

La ventaja para los programadores es que les toca hacer el trabajo creativo de escribir las especificaciones de proceso y tienen el placer de traducir sus propias especificaciones a código. Esto los involucra en el proceso de análisis de sistemas en una etapa más temprana de su carrera que lo que hubiera sido posible de otra manera. También tiene la ventaja de mantener a la gente madura en contacto con la tecnología, al forzarlos a continuar realizando alguna labor de diseño y programación.

No todos los analistas de sistemas maduros creen que esto es una buena idea, aunque admitan que no disfrutan el tener que escribir todas las especificaciones detalladas de procesos como parte de su trabajo. De cualquier forma, cobra fuerza la idea de que si la labor de programación es *precedida por un análisis detallado de sistemas del tipo que se describe en este libro,* se convertirá en un trabajo mecánico y de baja categoría, que puede desaparecer por completo si se desarrollan generadores inteligentes de código que compilen directamente las especificaciones de proceso. Por tanto, se puede esperar que las organizaciones cambien gradualmente sus asignaciones de trabajo a lo largo de los siguientes cinco o diez años en conformidad con las ideas anteriores.

23.3 IMPLANTACION DESCENDENTE Y SEGUIMIENTO RAPIDO

Puede haber tenido otra impresión al leer el material de este libro: que las actividades de análisis deben realizarse y *completarse* antes de que puedan comenzar las actividades de diseño y programación. Aunque muchos proyectos de hecho trabajan de esta manera, no es estrictamente necesario. El análisis, diseño y programación se pueden realizar de manera paralela.

El concepto de desarrollo paralelo de la especificación, el diseño y el código de un sistema a veces se conoce como seguimiento rápido y en algunos libros se conoce como implantación descendente (véase por ejemplo,[Yourdon, 1988]). Esto no sucede únicamente en el campo de las computadoras. Se discutió la idea brevemente en el Capítulo 5; revise el concepto de implantación descendente como parte del ciclo de vida del desarrollo global del sistema que se discutió en ese capítulo.

La industria de la construcción y muchas disciplinas ingenieriles siguen este enfoque en muchos proyectos. Como muchos administradores de estos proyectos dicen: "no hace falta conocer el número de chapas de las puertas de una construcción antes de construir los cimientos". En el caso del desarrollo de un sistema de información esto significa que los productos de alto nivel del análisis de sistemas, es decir, los documentos que constituyen el marco de referencia, tales como diagramas de flujo de datos, diagramas de entidad-relación y diagramas de transición de estados pueden usarse como base para el diseño de alto nivel. Y éste puede usarse como fundamento para escribir código de alto nivel *aun antes de haber terminado los detalles del análisis de sistemas.*

Existe una gran flexibilidad en este enfoque; se puede terminar el 80 por ciento de la labor de análisis antes de comenzar con el diseño y la programación, o se puede terminar sólo el 10 por ciento. El plan de proyecto que requiere casi completar el análisis antes de comenzar el diseño usualmente se conoce como de enfoque conservador; un plan de proyecto que requiera un traslape casi inmediato del análisis, el diseño y la programación se conoce como de enfoque radical. Cada administrador decide qué tan radical o conservador quiere que su proyecto sea, y puede cambiar de opinión dinámicamente durante el proyecto.

¿Por qué consideraría un administrador del proyecto seguir el enfoque radical? ¿Por qué podría alguien comenzar la labor de diseño y programación antes de concluir la de análisis? Existen muchas razones, de las cuales las más importantes son las siguientes :

- Como la labor de análisis, diseño y programación se realiza de manera concurrente, usualmente se tiene la oportunidad de acortar *dramáticamente* el *tiempo total* necesario para un proyecto. En muchos entornos, esto puede ser de crucial importancia, por ejemplo, en el caso de que un sistema determinado deba concluirse de manera definitiva para cierta fecha.

- La labor de desarrollo concurrente puede usarse como una forma de hacer el prototipo: permite al equipo del proyecto mostrar al usuario una versión esquelética del sistema antes de concluir la labor detallada de análisis. Esto puede evitar malos entendidos entre el usuario y el analista, que pueden suceder incluso con una especificación de estructura cuidadosamente desarrollada.

- El comenzar la labor de programación pronto, suele evitar diversos problemas referentes a la demanda de recursos, tales como tiempo de cómputo, que de otra manera se convertirían en un obstáculo. Por ejemplo, a menudo el enfoque conservador requiere cantidades enormes de tiempo de máquina durante las etapas finales de prueba y esto puede ser un gran problema.

El que el proyecto decida seguir un enfoque conservador o uno radical va más allá del alcance de este libro; algunos ambientes de proyecto pueden favorecer un enfoque conservador y otros exigir un enfoque altamente radical. Lo principal de lo que hay que estar conscientes es que el enfoque de análisis estructurado descrito en este libro no excluye ninguno de los dos enfoques, ni tampoco insiste en alguno de ellos.

23.4 PROGRAMACION Y LENGUAJES DE PROGRAMACION

Si aún está involucrado en el proyecto durante la etapa de implantación, debe tener cuando menos una comprensión general de las técnicas de programación. En esta sección discutiremos:

- Cuatro generaciones de lenguajes de programación
- Asuntos importantes en la programación
- Cosas que debe buscar si examina la codificación de los programadores

23.4.1 Las cuatro generaciones de lenguajes de programación

Se ha estado escribiendo programas de computación desde que se desarrollaron las computadoras de propósito general, hace 40 años. Los programas se escriben con lenguajes de programación de los cuales como ejemplos comunes tenemos BASIC, COBOL y FORTRAN. Es conveniente agrupar los distintos lenguajes de programación (existen cientos de lenguajes distintos que se utilizan en todo el mundo) en cuatro generaciones distintas:

- Lenguajes de primera generación: fueron los lenguajes de máquina que se usaron en los años 50; los programadores que intentaban que la computadora hiciera algo útil codificaban sus instrucciones con unos y ceros binarios. Existen ocasiones hoy en día en las que un sistema de cómputo defectuoso produce páginas y páginas de dígitos; todavía existen algunos jóvenes mal informados que creen que el lenguaje de máquina es la mejor manera de jugar con las computadoras personales, pero el resto del mundo dejó de pensar en el lenguaje de máquina hace unos 25 años.

- Lenguajes de segunda generación: son los sucesores del lenguaje de máquina; generalmente se conocen como lenguajes de ensamble o ensambladores. Los lenguajes de segunda generación son de bajo nivel en el sentido de que el programador tiene que escribir una declaración por cada instrucción de máquina. Por ello, aunque conceptualmente puede pensar en términos de la declaración X = Y + Z, tendría que traducir las siguientes declaraciones al lenguaje ensamblador:

```
LIMPIAR ACUMULADOR
CARGAR Y AL ACUMULADOR
AÑADIR Z A LOS CONTENIDOS DEL ACUMULADOR
ALMACENAR ACUMULADOR EN X
```

  Incluso este pequeño ejemplo muestra la principal desventaja del lenguaje ensamblador. En lugar de pensar en términos del problema que quiere resolver, el programador debe pensar en términos de la máquina. Alrededor de 1960 se comenzó a introducir lenguajes más poderosos; muchos programadores sanos hace mucho abandonaron el lenguaje ensamblador. Desafortunadamente, aún existen algunas situaciones en las que se necesitan. Muchas involucran computadoras muy pequeñas y de bajo poder (que pueden fabricarse muy económicamente y son lo suficientemente pequeñas como para caber, digamos, en un reloj digital), que no tienen la capacidad de tolerar el volumen de trabajo asociado con los lenguajes de mayor nivel.

- Lenguajes de tercera generación: son la norma actual; incluyen BASIC, COBOL, FORTRAN, Pascal, C, Ada y muchos más. Son de alto nivel en el sentido de que una sola declaración (tal como "MOVE A TO B" en COBOL) usualmente representa cinco o diez declaraciones de lenguaje ensamblador (y a veces hasta cien); son de alto nivel en un sentido más importante, porque permiten al programador expresar pensamientos en una forma un tanto más compatible con el área del problema en el que está trabajando. Sin embargo, son de bajo nivel en algunos aspectos importantes. Requieren que el programador esté íntimamente involucrado en la tediosa labor de dar formato a los reportes de la computadora, al igual que en la edición y validacion de las entradas del programa. A menudo el programador piensa: "este reporte debe tener el encabezado estándar en la parte superior de cada página, con el número de página a la derecha y la fecha en la izquierda, como todos los demás", pero puede tener que escribir 20 o 30 declaraciones de COBOL para lograrlo.

  Los lenguajes de tercera generación también se caracterizan como lenguajes guiados por *procedimientos*. Requieren que el programador piense con cuidado la secuencia de los cálculos o procedimientos necesarios para lograr alguna acción. En una aplicación científica, por ejemplo, el programador puede saber que quiere añadir el arreglo A al arreglo B; sin embargo, puede verse forzado a escribir los pasos detallados del procedimiento para cada uno de los elementos de los dos arreglos, en lugar de simplemente decir, "sumar estos dos arreglos" sin tener que preocuparse por los pasos del procedimiento.

- Lenguajes de cuarta generación: los lenguajes de cuarta generación, o 4GLs, son la moda actual y muchos consultores de computación los consideran el desarrollo más importante en el campo de software en los últimos 20 años. Algunos han existido durante casi una década, pero sólo se hicieron populares en los últimos años. Como ejemplos hay FOCUS, IDEAL, MARK IV, RAMIS, MANTIS, MAPPER, dBASE-IV Plus y Rbase-5000. La mayor parte tienen características de programación estructurada ausentes en los lenguajes de tercera generación, pero incluyen otras. En lo particular, la mayoría de los detalles tediosos de programación relacionados con introducir datos a la computadora (por medio de una terminal) se ocultan al programador; con una sola orden sencilla, el

programador puede especificar que la computadora debe aceptar un tipo especificado de datos desde el teclado, validarlo, y volver a almacenarlo en un elemento de datos designado. La misma labor puede requerir 10 o 20 declaraciones en un lenguaje de programación de tercera generación, o de 100 a 200 en uno de segunda.

De manera similar, muchos detalles tediosos de programación asociados con la producción de reportes de salida (por ejemplo, reportes de inventario, cheques, facturas o un resumen de los pedidos del día) se manejan automáticamente por medio de lenguajes de cuarta generación. Si la colocación precisa de la información en un reporte no es muy importante (como a menudo suele suceder), el programador no tiene siquiera que especificarlo; de lo contrario, (como es el caso de un cheque procucido por computadora, donde el monto debe imprimirse en un lugar específico), los detalles se especifican fácilmente con unas cuantas instrucciones de 4GL.

23.4.2 Cuestiones importantes en programación

Sin tomar en cuenta el lenguaje de programación que se use, hay cuestiones comunes que todos los programadores enfrentan. Como analista, debe estar familiarizado con ellas; las más comunes se mencionan a continuación:

- *Productividad:* probablemente, la cuestión más importante de la programación actual sea la productividad: escribir más software, más rápidamente. La principal razón de esto es la enorme cantidad de sistemas y aplicaciones que siguen en espera en las grandes organizaciones: una organización grande típica tiene un retraso de entre cuatro y siete años en los nuevos trabajos por efectuar.[2] Por ello, se deben alentar los lenguajes y técnicas de programación que promueven la productividad; exceptuando casos raros, la productividad se considera más importante actualmente que la eficiencia.

- *Eficiencia:* en algunas aplicaciones, la eficiencia sigue siendo de importancia. Esto sucede en muchos sistemas de tiempo real, y puede darse en otros tipos de sistemas que procesan grandes volúmenes de datos (por ejemplo, muchos de los sistemas que operan en las oficinas del Seguro Social, al igual que otros sistemas enormes en bancos, reservación en aerolíneas, compañías de bolsa y compañías de seguros). Para estas aplicaciones, usualmente resulta importante minimizar la cantidad de tiempo de CPU requerido por el programa; también puede ser importante minimizar la utilización de memoria, al igual que la de otros recursos como el disco. Observe que la meta de eficiencia usualmente entra en conflicto con otras metas discutidas en esta sección: si se emplea mucho tiempo en el desarrollo de un programa eficiente, es probable que sea menos mantenible y menos transportable, y que tenga más errores residuales sutiles, además de que tal vez reduzca la productividad de la persona que escribió el programa.

- *Corrección:* se podría argumentar que esto es lo más importante. Después de todo, si el programa no funciona correctamente, no importa qué tan eficiente sea. Se prefieren lenguajes de programación como Ada y Pascal si la corrección es de importancia crítica (como, por ejemplo, si se estuviera construyendo el sistema de la Guerra de las Galaxias, o el sistema de control para un reactor nuclear), porque son de "tipos rígidos": se requiere que el programador declare la naturaleza de sus variables (es decir, si son enteros, de caracteres, de punto flotante, etc.) y el lenguaje revisa todo cuidadosamente para evitar referencias ilegales a los datos.

- *Portabilidad:* en algunos ambientes esto es importante; el usuario puede desear ejecutar el mismo sistema en varios tipos distintos de computadoras. Algunos lenguajes de programación son más portátiles que otros; irónicamente, esto es más cierto en lenguajes de tercera generación (C, Pascal, FORTRAN, COBOL, etc.) que en los de cuarta. Sin embargo, no existe un lenguaje universalmente portátil; siempre hay forma de que el programador aproveche las características especiales de una computadora o un sistema operativo específicos. Por ello, además del lenguaje de programación debemos preocuparnos por el estilo de programación, si la portabilidad es un factor importante.

- *Mantenibilidad:* finalmente, debemos recordar que los sistemas viven durante mucho tiempo, por lo que el software debe mantenerse. El mantenimiento se discute con más detalle en el Capítulo 24.

23.4.3 Cosas de las que hay que tener cuidado

Como analista, puede tener la oportunidad de observar el trabajo que realizan los programadores del proyecto; de hecho, puede que sea su supervisor. Como se indicó anteriormente, debe estar al tanto de que la productividad, la eficiencia, lo correcto, la portabilidad y la mantenibilidad sean cuestiones de importancia. ¿Pero cómo se logran estas metas? Puede consultar otros textos, como [Yourdon, 1976] y [Kernighan y Plauger, 1975], para ver discusiones detalladas acerca de técnicas de programación; sin embargo, las siguientes son cuestiones clave en la programación:

- *La programación estructurada:* Suponiendo que los programas se escriban en un lenguaje de tercera o cuarta generación, debe seguirse un enfoque de programación estructurada, en el que la lógica del programa (las decisiones y ciclos) se organiza en combinaciones anidadas de construc-

2 Esto no significa de cuatro a siete años de trabajo para una sola persona, sino más bien de cuatro a siete años de trabajo para toda la organización de desarrollo de sistemas de información. Para más detalles véase [Yourdon, 1986].

ciones SI-ENTONCES-OTRO y HACER-MIENTRAS. Casi todos los textos modernos de programación enseñan un enfoque estructurado; vea, por ejemplo, [Wells, 1986], [Benton y Weekes, 1985], [Yourdon, Gane y Sarson, 1976], y [Yourdon y Lister, 1977].

- *Módulos pequeños:* Es esencial que los programas se organicen en pequeños módulos para que la lógica de programación quepa en una sola página de listado de programa. Es importante recordar que la complejidad de un programa no aumenta linealmente con su tamaño: un programa de 100 pasos casi siempre tendrá más del doble de complejidad que uno de 50. Como se vio en el Capítulo 22, esto está principalmente bajo el control del diseñador; pero puede no tener posibilidad de determinar qué tan grande será un módulo, sobre todo si no está familiarizado con el lenguaje de programación que se usará en el proyecto. Por ello, el programador puede tener que continuar la actividad de diseño, partiendo un módulo en submódulos de menor nivel, para que cada uno represente no más de 50 pasos de programación.

- *Sencillez de estilo:* Muchos textos de programación, tales como [Yourdon, 1976] y [Kernighan y Plauger, 1975], tienen reglas detalladas para la escritura de programas sencillos, es decir, programas que el programador promedio puede entender y que se le pueden pasar al programador de mantenimiento; entre estas reglas se tiene la sugerencia de que el programador intente evitar declaraciones de programación con expresiones booleanas compuestas, como

SI A Y B O NO C Y D ENTONCES AÑADIR 3 A X

Es interesante notar que en los últimos diez años se han desarrollado diversos modelos matemáticos sobre la complejidad de los programas; uno de los más populares es el modelo de complejidad ciclomática de McCabe [McCabe, 1976], que proporciona una medida cuantitativa de la complejidad intrínseca de un programa.[3] Algunas organizaciones actualmente insisten en que todos los programas nuevos deben pasar por un verificador de complejidad automatizado para asegurar que no sean demasiado complejos.

23.5 PRUEBAS

Es probable que el proceso de probar el sistema tome tanto como la mitad del tiempo programado para su desarrollo, dependiendo de qué tan cuidadosamente se hayan hecho las actividades iniciales de análisis, diseño y programación. Incluso si se hizo una labor perfecta de análisis, diseño y programación, se debe hacer algún

3 Para los lenguajes de tercera generación como COBOL, la complejidad ciclomática es aproximadamente igual al número de instrucciones SI del programa. Para más información véase [DeMarco, 1982].

esfuerzo para verificar que no haya errores. Si, por otro lado, se hizo un trabajo imperfecto (que suele ser el caso casi siempre), entonces la prueba se vuelve iterativa: la primera tanda de pruebas muestra la presencia de errores, y las posteriores verifican si los programas corregidos funcionan correctamente.

¿Qué necesita saber acerca de las pruebas como analista del sistema? Esto dependerá desde luego, de qué tan involucrado esté en el proceso. En muchos casos, el analista trabaja de manera cercana con el usuario para desarrollar un conjunto eficaz y de gran alcance de casos de prueba *basados en el modelo esencial y el modelo de implantación del usuario.* Este proceso de desarrollar casos de prueba de aceptación puede llevarse a cabo en paralelo con las actividades de implantación del diseño y de la programación, para que, cuando los programadores terminen de escribir sus programas y de realizar sus propias pruebas locales, el equipo del analista/usuario esté listo con sus propios casos de prueba.

Además de este concepto básico (que la descripción de los requerimientos del usuario forme la base de los casos de prueba finales), debe estar familiarizado con los diferentes tipos de prueba, al igual que con algunos conceptos relacionados de cerca con las pruebas, que se discuten a continuación:

23.5.1 Tipos de prueba

A estas alturas tal vez piense que no existe más que un tipo de prueba: ¿qué más podría haber que el hecho de simplemente idear casos de prueba y luego revisar si el sistema trabaja correctamente?

Lo primero que hay que entender es que hay distintas *estrategias* de prueba; las dos más comunes se conocen como prueba ascendente y descendente. El enfoque ascendente empieza por probar módulos individuales pequeños separadamente; esto a menudo se conoce como prueba de unidades, prueba de módulos, o prueba de programas. Luego, los módulos individuales se combinan para formar unidades cada vez más grandes que se probarán en masa; esto se conoce como prueba de subsistemas. Finalmente, todos los componentes del sistema se combinan para probarse; esto se conoce como prueba del sistema, y suele estar seguido de las pruebas de aceptación, donde se permite al usuario usar sus propios casos de prueba para verificar que el sistema esté trabajando de manera correcta.

El enfoque de prueba descendente empieza con un esqueleto del sistema; es decir, la estrategia de prueba supone que se han desarrollado los módulos ejecutivos de alto nivel del sistema, pero que los de bajo nivel existen sólo como módulos vacíos.[4] Dado que muchas de las funciones detalladas del sistema no se han implantado, las pruebas iniciales están muy limitadas; el propósito es simplemente comenzar

4 Un ejemplo de módulo vacío es uno que no procesa nada, sino que simplemente termina luego de ser llamado. Otro ejemplo es un módulo que devuelve los mismos parámetros de salida independientemente de los parámetros de entrada que se le pasaron cuando se llamó. De esta forma,

a ejercitar las interfases entre los subsistemas principales. Las pruebas siguientes abarcan y tratan aspectos cada vez más detallados del sistema. El enfoque descendente de prueba generalmente se considera preferible para muchos sistemas en la actualidad; para más detalles al respecto, véase [Yourdon, 1986].

Además de estos conceptos básicos, debería estar familiarizado con los siguientes tipos de prueba:

- *Prueba funcional:* Esta es la forma más común de prueba; su propósito es asegurar que el sistema realiza sus funciones normales de manera correcta. Así, los casos de prueba se desarrollan y se alimentan al sistema; las salidas (y los resultados de los archivos actualizados) se examinan para ver si son correctos.

- *Prueba de recuperación:* El propósito de este tipo de prueba es asegurar que el sistema pueda recuperarse adecuadamente de diversos tipos de fallas. Esto es de particular importancia en los sistemas en línea grandes, al igual que en varios tipos de sistemas de tiempo real que controlan dispositivos físicos y/o procesos de fabricación. Las pruebas de recuperación pueden requerir que el equipo que realiza el proyecto simule (o provoque) fallas de hardware, fallas de corriente, fallas en el sistema operativo, etc.

- *Prueba de desempeño:* El propósito de este tipo de prueba es asegurar que el sistema pueda manejar el volumen de datos y transacciones de entrada especificados en el modelo de implantación del usuario, además de asegurar que tenga el tiempo de respuesta requerido. Esto puede requerir que el equipo que realiza el proyecto simule una gran red de terminales en línea, de manera que se pueda engañar al sistema para que "crea" que está operando con una gran carga.

Existe un último concepto del que debe estar al tanto: la noción de prueba exhaustiva. En el proyecto ideal, se generarían casos de prueba para cubrir cada entrada posible y cada combinación posible de situaciones que el sistema pudiera enfrentar alguna vez; luego, se probaría de manera exhaustiva para asegurar que su comportamiento sea perfecto. Sólo existe un problema con esto: no funciona. El número de casos de prueba para un sistema grande y complejo típico es tan increíblemente grande, a menudo del orden de 10'100 casos de prueba distintos o más, que aún si se pudiera realizar una prueba cada milésima de segundo tomaría más de la edad del universo terminar todas las pruebas. Consecuentemente, *nadie realiza*

la prueba descendente inicial de un sistema de nómina puede consistir en módulos vacíos que le pagan a todo munco $100 a la semana, sin tomar en cuenta su clasificación salarial; el módulo vacío de los cálculos de impuestos tal vez siempre deduzca $10 en impuestos a todos los cheques de la nómina. El objetivo de la prueba descendente inicial simplemente será determinar si el sistema funciona o no y si es en realidad capaz de generar un conjunto fijo de cheques de $100.

pruebas verdaderamente exhaustivas en algo que no sea un sistema trivial; cuando más, quienes desarrollan el sistema pueden aspirar a crear casos de prueba que ejerciten (o cubran) un gran porcentaje de los diferentes caminos de decisión que pueda tomar.[5] Esto hace que sea aún más importante asegurar que el modelo de los requerimientos del usuario y los diversos modelos de implantación sean tan correctos como se pueda.

Suponga que, por ejemplo, se quisiera desarrollar casos de prueba para una porción de un sistema que calcula el salario neto de un empleado, como muestra la figura 23.1. Suponga que el **salario bruto** se define en el diccionario de datos como un entero (es decir, un salario expresado en cantidades enteras) que va desde 0 a 10,000. Entonces parecería que una prueba realmente exhaustiva consistiría en especificar el **salario neto** correcto para cada una de las 10,000 cantidades posibles de **salario bruto.** Es de suponerse que, si nuestro equipo de implantación llevara a cabo dichos 10,000 casos de prueba y verificara que de hecho sí se produce el **salario neto,** entonces se podría confiar en que el proceso estuviera operando correctamente.

Figura 23.1: **Una pequeña porción de un sistema**

Pero, espere. ¿Qué pasa con los valores potencialmente incorrectos de **salario bruto**? ¿Qué tal si el usuario proporciona un **salario bruto** negativo? ¿Qué tal si proporciona uno de 100,000? Dado que potencialmente existe un número infinito de valores potenciales de **salario bruto,**[6] y dado que no tenemos conocimiento del comportamiento interno del programa que realiza la función **CALCULAR SALARIO NETO,** nos enfrentamos a un número aparentemente infinito de casos de prueba. Si se desarrollan al final de la fase de análisis del proyecto, utilizando el diccionario de datos y la especificación del proceso, entonces no existe manera de saber cómo fun-

5.[Dunn, 1984] y [Myers, 1979] proporcionan discusiones detalladas acerca del alcance de las pruebas.

6 En realicad, el número de casos de prueba no es infinito, debido a la precisión limitada de los números que se almacenan en la memoria de la computadora. Si se almacena como entero, una computadora típica almacenará como máximo el número 232 o 264. Si se almacena en punto flotante, tal vez se pueda representar números de tamaño 10'100 o mayores, pero usualmente sólo hay 8 o 9 dígitos significativos de precisión. Por tanto, esto no representa el infinito pero, no obstante, es un número muy grande.

cionará finalmente el programa cuando el programador escriba el código; por ello, nos vemos forzados a llevar a cabo una prueba de caja negra.

Si se conoce la lógica y estructura internas del programa (es decir, después de que el programador escribió el código), entonces se pueden basar los casos de prueba en la lógica del programa existente y realizar lo que se conoce como prueba de "caja de vidrio". Generalmente se puede demostrar, por ejemplo, que si el programa identifica de manera correcta un valor de **salario bruto** menor que cero, entonces identificará correctamente todos los valores negativos de **salario bruto.** En general, debe poderse demostrar que el programa tendrá un comportamiento consistente para varios niveles de salario bruto, reduciendo así el número requerido de casos de prueba a un número manejable. Aunque esto no constituye una prueba exhaustiva, es de suponerse que podríamos lograr un nivel razonablemente alto de confianza si se han desarrollado casos de prueba para todos los caminos significativos que el sistema puede tomar.

Pero, un momento. **CALCULAR SALARIO NETO** es tan sólo un proceso, es decir, una burbuja de entre cientos o incluso miles, en un sistema grande. Si se necesitan, digamos, 1000 casos de prueba para verificar que CALCULAR SALARIO NETO opera correctamente (en términos de corrección funcional), entonces perfectamente se pueden necesitar 1000 pruebas para cada uno de los otros 1000 procesos del sistema. El número total de casos de prueba distintos podría ascender a 1000 * 1000= 1,000,000. Esto es conservador (y no toma en cuenta la dimensión de complejidad adicional causada por pruebas de entrada y salida, pruebas de recuperación, etc.).

Así que debe admitirse desde un principio la imposibilidad de realizar pruebas exhaustivas. Pero es posible, como se hizo notar anteriormente, escoger con cuidado casos de prueba, para pasar por tantos caminos lógicos del sistema como sea posible. Aun así, debemos estar preparados para un volumen grande, por no decir enorme, de pruebas. Para realizar esto de manera efectiva, el equipo que desarrolla el sistema necesita tres cosas: planes de prueba, descripciones de pruebas y procedimientos de prueba. Un *plan de prueba* es exactamente lo que parece: un documento organizado que describe las actividades de prueba. Un documento de planeación de pruebas típico contendrá la siguiente información:

- *Propósito de la prueba:* cuál es el objetivo de la prueba, y qué parte del sistema se está probando.
- *Localización y horario de la prueba:* dónde y cuándo se hará.
- *Descripciones de la prueba:* descripción de las entradas que se proporcionarán al sistema, y las salidas y resultados que se anticipan. Usualmente se darán descripciones de las entradas de prueba en el formato de diccionario de datos que se discutió en el Capítulo 10.

- *Procedimientos de prueba:* descripción de cómo se deben preparar y presentar los datos de prueba al sistema; cómo capturar los resultados de salida, cómo analizar los resultados de las pruebas, y cualesquiera otros procedimientos operacionales que se deban observar.

23.5.2 Conceptos relacionados

Aunque la mayor parte de las organizaciones llevan a cabo pruebas de la manera que se discutió anteriormente, existen algunos conceptos relacionados que se pueden usar para aumentar el proceso estándar de prueba, que incluyen:

- Recorridos y revisiones (walkthroughs)
- Inspecciones
- Pruebas de corrección

Los recorridos y revisiones, que se discuten en el Apéndice D, son una forma de supervisión hecha por un grupo revisor de productos técnicos; se usan ampliamente en proceso de datos para revisar diagramas de flujo de datos (y otros productos del análisis de sistemas), diagramas de estructura (y otros productos del diseño), además de programas. Aunque esto es distinto a hacer pruebas, su objetivo es el mismo: descubrir posibles errores en el sistema.

Las inspecciones son similares a los recorridos, pero tienen un plan más formal de puntos a examinar en el programa (o en la especificación, o en el diseño, dependiendo del tipo de inspección) antes de que se pueda aprobar. Por analogía, considere lo que sucede cada año cuando inspecciona su automóvil: el mecánico tiene una lista específica de puntos, como frenos, luces, emisiones del escape, etc., que debe examinar antes de poner la calcomanía en el auto.

Finalmente, existe un número limitado de casos donde se desarrollan *pruebas formales de corrección* de un programa; este proceso es un tanto análogo al proceso de desarrollar las demostraciones de geometría que se estudiaron en la escuela. Desafortunadamente, es extremadamente difícil y tardado desarrollar pruebas rigurosas de la corrección de un programa de computadora, y rara vez se ha hecho para algo más grande cue que unos cuantos cientos de instrucciones. Sin embargo, por lo menos un proyecto del gobierno de los EUA desarrolló pruebas de este tipo, auxiliado por computadora, para un sistema de alrededor de 10,000 instrucciones; aunque costó cerca de $500,000 dólares estadounidenses y tomó 6 meses de trabajo, puede justificarse para ciertos sistemas de alto riesgo o máxima seguridad. Para más acerca de esto, vea el Capítulo 6 de [Dunn, 1984] o los reportes en [Elspas et al, 1972] y [Dunlop y Basili, 1982].

23.6 MANTENIMIENTO DE LA ESPECIFICACION DURANTE LA PROGRAMACION: PRELUDIO AL CAPITULO 24

Como se mencionó anteriormente, es posible que la especificación estructurada cambie durante el proceso de programación. Esto puede suceder como resultado de la estrategia de seguimiento rápido que se describió antes, o porque las especificaciones originales estaban mal, o simplemente porque los usuarios cambian de opinión sobre sus requerimientos. En cualquier caso, es una realidad, y resalta un punto importante: la especificación del proceso no se puede considerar congelada después de concluida la fase de análisis. Debe considerarse como un documento vivo que requerirá mantenimiento continuo *incluso antes de que el sistema mismo haya entrado a la fase de mantenimiento.* El Capítulo 24 trata con mayor detalle este asunto.

23.7 ¿QUE OCURRE DESPUES DE LAS PRUEBAS?

Tal vez piense que su labor terminó cuando termina de probar el sistema. Desafortunadamente, queda aún algo qué hacer, aunque ya no en su papel de analista. Sin embargo, alguien (a menudo un grupo grande de "personas") debe llevar a cabo las actividades finales en un proyecto de desarrollo de sistemas:

- Conversión
- Instalación
- Capacitación

La *conversión* es la tarea de traducir los archivos, formas y bases de datos actuales del usuario al formato que el nuevo sistema requiere. En algunos casos raros, esto puede ser una actividad no relevante, porque ya no hay datos. Sin embargo, si el usuario está reemplazando un sistema actual por uno nuevo, es probable que esto sea una tarea difícil y delicada. Se necesita desarrollar un plan de conversión, de preferencia en cuanto se complete el modelo de implantación del usuario, para cubrir los siguientes puntos:

- Si el usuario ya tiene datos existentes asociados con un sistema existente, probablemente querrá usarlos hasta el último momento posible antes de pasarse al sistema nuevo. Por ello, es difícil considerar los datos existentes como estáticos.
- Pudiera haber un volumen tan grande de datos existentes que sea imprácticó considerar convertirlo todo a la vez. Los archivos y registros podrían tener que convertirse en forma incremental. Esto obviamente requiere de una planeación y coordinación cuidadosa.
- La conversión debe llevarse a cabo de una manera automatizada; esto sólo se puede hacer si los archivos y datos actuales existen en alguna forma automatizada. De ser así, debiera ser relativamente fácil escribir un programa (o usar un paquete comercial existente) para traducir los archivos actuales al formato requerido por el sistema nuevo. Sin embargo, a veces resulta difícil convertir los datos en forma automatizada, sobre todo si los archivos existentes se tienen en distintas computadoras, en distintos formatos, etc. De hecho, desarrollar el software de conversión puede resultar ser por sí mismo un proyecto importante de desarrollo de sistemas.
- Los datos existentes pueden contener errores; de hecho, si se crearon y mantuvieron manualmente, puede estar casi seguro de que habrá errores. Por ello, parte del proceso de conversión es la detección y corrección de errores, que puede volver aún más difícil y tardado el proceso. Algunos archivos y registros existentes pueden resultar ilegibles o incomprensibles; en otros casos, puede ser obvio que los datos existentes están mal, pero podría no ser claro cuáles son los valores correctos.
- Además de convertir archivos existentes, puede ser necesario convertir programas y procedimientos existentes. En algunos casos, los programas y procedimientos existentes pueden usarse en su forma actual; en otros, se tendrán que desechar y reemplazar por completo.

La *instalación* del nuevo sistema puede ser un asunto instantáneo, pero a menudo es una tarea enorme. Usualmente, se debe hacer lo siguiente:

- A la instalación del nuevo sistema debe precederle la preparación de la sede de la computadora, usualmente con varios meses de anticipación. Esto implica construir o rentar un local de cómputo con la corriente, espacio, iluminación y control ambiental (temperatura, humedad, polvo, electricidad estática, etc.) apropiados. Esto muchas veces se hace en conjunto con el proveedor de hardware o el departamento de operaciones de cómputo de la organización.
- Se puede requerir la preparación de la sede del usuario también, sobre todo en el caso de sistemas en línea que tienen terminales e impresoras en el área de trabajo del usuario. En el caso sencillo, se pueden distribuir las terminales al área de trabajo del usuario justo antes de instalar el sistema; sin embargo, en algunos casos, puece requerirse construir un lugar de trabajo totalmente nuevo (considere, por ejemplo, una terminal de reservaciones de una aerolínea en un aeropuerto).
- La instalación del hardware, cuando el sistema requiere de su propia computadora, usualmente la efectúa el proveedor. En ocasiones se involucran varios proveedores, sobre todo para sistemas en línea y de tiempo real. En el caso de un sistema sencillo desarrollado para una computadora personal, la instalación puede ser tan sencilla como sacar la computadora de su caja y conectarla.

- La instalación del software, que involucra cargar todos los programas que se escribieron para el nuevo sistema en la o las computadoras adecuadas, y prepararlos para su operación.

Tenga en mente que lo recién descrito supone que existe una sola instalación en una sola sede. Pero a menudo no es así; para un sistema grande y distribuido, pudiera haber una sola sede de computadoras central, y docenas o incluso cientos de sedes de usuarios. Por ello, puede ser necesario instalar el sistema por etapas, con la visita de equipos de instalación especialmente capacitados a cada sede de usuarios de acuerdo con un programa preestablecido. En este caso, la instalación y cambio al nuevo sistema no puede ser inmediata, sino que debe irse haciendo gradualmente durante un periodo de días, semanas o incluso meses.

La *capacitación* es la tarea final del equipo de desarrollo del sistema: la capacitación de los usuarios (obviamente), además de la preparación del personal de operaciones, los programadores de mantenimiento y varios niveles de administración. Se debe desarrollar un plan de capacitación pronto, pues hay mucho trabajo que hacer, y debe estar listo al mismo tiempo (si es que no antes) de que el sistema comience a operar. El plan de capacitación debe considerar los siguientes aspectos:

- ¿Cómo se llevará a cabo? Muchos proyectos de desarrollo de sistemas dependen de *manuales para el usuario* y *guías de referencia* para proporcionar a los usuarios documentos escritos. Sin embargo, podrían convenir clases y seminarios en vivo, además de pláticas de orientación para administradores y personas que necesitan estar al tanto del sistema aunque no interactúen con él a diario. Con la tecnología actual existe una gran gama de opciones de medios didácticos: videocassettes o videodiscos, enseñanza por computadora, e incluso versiones de simulacro del sistema real para que los usuarios puedan ingresar transacciones y aprender a interactuar con él.

  En el caso extremo, la capacitación puede consistir en opciones de ayuda altamente elaboradas integradas al sistema mismo. Esto se está volviendo cada vez más popular con la proliferación de las computadoras personales, pero no es muy práctico para sistemas grandes con una comunidad grande y diversificada de usuarios; por otro lado, se puede usar para aumentar y reforzar otras formas de capacitación.

- ¿Quién llevará a cabo la capacitación? En algunos casos, los miembros del equipo de desarrollo de sistemas participan en el proceso, sobre todo dado que se supone que son los mejores expertos sobre cómo funciona el sistema. Sin embargo, tenga en mente que el mejor programador (o analista) no siempre es el mejor maestro; de hecho, quienes desarrollan el sistema suelen comportarse de una manera muy defensiva si los usuarios empiezan a hacer preguntas que consideran hostiles. Además, están (casi por definición) terriblemente ocupados con el diseño, codificación y prueba del sistema hasta el último momento. Los analistas podrían tener más tiempo después de terminar el modelo esencial y el modelo de implantación del usuario.

- ¿A quién se preparará y en qué horario? Obviamente, se necesita capacitar a los usuarios antes de que usen el sistema; por otro lado, no resulta efectivo prepararlos seis meses antes de que puedan ver el nuevo sistema. Por ello, la capacitación debe hacerse en un tiempo relativamente corto; pero esto, a su vez, a menudo interferirá con el trabajo cotidiano normal que los usuarios tratan de hacer. Por tanto, se debe negociar con ellos un programa cuidadoso de actividades de capacitación.

23.8 RESUMEN

Este capítulo cubrió una amplia gama de tópicos: programación, pruebas, conversión, instalación y capacitación. El espacio disponible no nos permite mostrarlos de manera detallada, pero el tratamiento breve que se proporciona debe dar al analista una visión general de estas actividades finales en el proyecto de desarrollo de sistemas. Se encuentran detalles adicionales en las referencias al final de este capítulo.

REFERENCIAS

1. Edward Yourdon, *Managing the Systems life Cycle,* 2ª edición, Englewood Cliffs, N.J.: Prentice Hall, 1988.
2. Edward Yourdon, *Nations at Risk.* Nueva York: YOURDON Press, 1986.
3. Edward Yourdon, *Techniques of Program Structure and Design,* 2ª edición, Englewood Cliffs, N.J: Prentice- Hall, 1976.
4. Brian Kernighan y P.J. Plauger, *The Elements of Programming Style.* Reading, Mass.: Addison-Wesley, 1975.
5. Timothy Wells, *Structured Systems Development in COBOL.* Nueva York: YOURDON Press, 1986.
6. Timothy Wells, *Structured Systems Development in BASIC.* Nueva York: YOURDON Press, 1985.
7. Timothy Wells, *Structured Systems Development in Pascal.* Nueva York: YOURDON Press, 1986.
8. Stan Benton y Leonard Weekes, Program *It Right: Structured Programming in BASIC.* Nueva York: YOURDON Press, 1985.
9. Edward Yourdon, Chris Gane y Trish Sarson, *Learning to Program in Structured COBOL,* Parte I. Nueva York: YOURDON Press, 1976.

10. Edward Yourdon y Timothy Lister, *Learning to Program in Structured COBOL*, Parte II. Nueva York: YOURDON Press, 1977.

11. Tom DeMarco, *Controlling Software Projects*. Nueva York: YOURDON Press, 1982.

12. Glenford Myers, *The Art of Software Testing*. Nueva York: Wiley, 1979.

13. Tom McCabe, "A Complexity Measure", *IEEE Transactions on Software Engineering*, Vol. SE-2, número 12 (diciembre de 1976), pp. 308-320.

14. Edward Yourdon, *Managing the Structured Techniques*, 3ª edición, Nueva York: YOURDON Press, 1986.

15. Robert Dunn, *Software Defect Removal*. Nueva York: McGraw-Hill, 1984.

16. B. Elspas y otros, "An Assessment of Techniques of Proving program Correctness", *ACM Computing Surveys*, Vol. 4 (junio de 1972), pp. 97-147.

17. D. Dunlop y V. Basili, "A Comparative Analysis of Functional Correctness", *ACM Computing Surveys*, Vol. 14 (junio de 1982), pp. 229-244.

PREGUNTAS Y EJERCICIOS

1. ¿Qué actividades incian en un proyecto de desarrollo de sistemas después de que termina el diseño?

2. ¿Cuáles son las seis razones por las que el analista puede necesitar seguir involucrado con un proyecto durante las actividades de programación y prueba?

3. Si el analista es el jefe del proyecto, ¿cree que sea importante que esté familiarizado con las técnicas de programación y las estrategias de prueba? ¿Por qué?

4. En su organización, ¿se espera que los analistas participen en las actividades de diseño y programación? ¿Cree que sea buena idea? ¿Por qué?

5. ¿Por qué es probable que el analista se vea involucrado en el desarrollo de datos de prueba para el sistema? ¿Quién más es probable que se vea involucrado?

6. ¿Qué debe hacer el analista si los programadores piden que se cambie la especificación del sistema durante la fase de programación del proyecto?

7. ¿Qué debe hacer el analista si los usuarios piden cambiar los requerimientos del sistema después de que los programadores comenzaron a implantarlo? ¿Qué tan probable cree que sea una situación como ésta?

8. ¿Por qué es posible que un usuario quiera cambiar los requerimientos del sistema después de concluida la fase de análisis?

9. ¿Qué dificultades pueden esperarse si un analista experimentado debe hacer toda la labor de análisis de un proyecto?

10. ¿Qué tipo de reacción negativa puede esperarse de los programadores en una organización si los analistas realizan todas las actividades detalladas de especificación que se discuten a lo largo de este libro?

11. ¿Qué tipo de estructura organizacional podría tenerse para acomodar la combinación de personal experimentado/ personal nuevo y de actividades técnicas de alto/bajo nivel en un proyecto?

12. ¿Pueden automatizarse los aspectos de programación si las actividades de análisis y diseño de sistemas se han realizado competamente y en detalle? ¿Por qué? ¿Cree que esta situación cambie durante los siguientes 5 a 10 años?

13. ¿Es necesario completar todas las actividades de análisis de sistemas antes de comenzar con la labor de programación? ¿Por qué sí o por qué no?

14. ¿Qué significa seguimiento rápido?

15. ¿En qué otras industrias aparte de la de desarrollo de sistemas se efectúa el seguimiento rápido?

16. ¿Qué es un enfoque conservador para implantar un sistema? ¿Qué es un enfoque radical?

17. ¿Cuáles son las tres principales razones por las que un administrador de proyecto pudiera adoptar un enfoque radical en la implantación de sistemas?

18. ¿Por qué no puede considerarse congelada la especificación de proceso al final de la fase de análisis del proyecto?

24 MANTENIMIENTO DE LA ESPECIFICACION

Hasta ahora, el profesional clave de la computadora era alguien que podía aprender lo suficiente acerca de las necesidades de las organizaciones como para expresarlas en lenguaje de computadora. En el futuro, a medida que nuestra sociedad se vuelva irrevocablemente computarizada, el profesional clave será alguien que pueda aprender lo suficiente acerca de sistemas computarizados como para expresarlos en lenguaje humano. Sin ese alguien, habremos perdido el control de nuestra sociedad. Ese alguien es el irgeniero en reversa . Los que mantienen el software son los ingenieros en reversa de nuestra sociedad.

Nicholas Zvegintzov, editor
Software Maintenance News

En este capítulo se aprenderá:

1. Por qué es importante tener al día las especificaciones.
2. Qué tipo de cambios se necesitan hacer a una especificación.

Para muchos analistas, el proyecto termina cuando se termina la especificación estructurada y el usuario la acepta. En ese momento se entrega la especificación al equipo de implantación constituido por los diseñadores y programadores que construirán un sistema a partir de la especificación.

Desde luego, algunos analistas siguen colaborando con el proyecto a lo largo de las fases de diseño e implantación. A veces el analista sirve de administrador del proyecto, guiando y dirigiendo los esfuerzos del equipo de implantación. A veces sigue colaborando con la realización de análisis, es decir, sirviendo como intermediario entre el usuario y el equipo de implantación. También puede participar en el desarrollo de manuales para el usuario, datos de prueba de aceptación, planeación de la instalación y varias actividades complementarias que se hacen de manera concurrente con el proceso de implantación.

Sin embargo, casi todos los analistas dejan el proyecto en cuanto se completa el desarrollo y se pone en operación el nuevo sistema. Algunos programadores se quedan para actividades de mantenimiento, pero cuando se termina la fase de desarrollo se termina la fiesta, y la mayoría de los analistas, diseñadores y programadores se transfieren a otros proyectos nuevos (y a menudo a compañías nuevas, donde pueden percibir un salario mayor al actual).

Pero el trabajo hecho por el analista (todo el trabajo que se discutió a lo largo de este libro) sigue siendo importante. Así como los *programas* deben mantenerse durante los 5, 10 o 20 años de vida operacional del sistema, de igual manera debe mantenerse su especificación. O, por decirlo de otra manera, cambiarán diversos aspectos de la implantación del sistema durante su vida, y para cada uno de estos cambios debe haber uno correspondiente en la especificación.

Aunque el analista original pudiera no permanecer con el proyecto durante la vida operacional de éste, es importante que deje un legado que se *pueda* mantener. Este capítulo discute el mantenimiento de la especificación del sistema.

24.1 POR QUE ES IMPORTANTE

Hasta aquí podría estar un tanto confundido; después de todo, piensa, es perfectamente obvio que la especificación del sistema puede actualizarse. ¿Por qué no hacerlo? Desafortunadamente, la historia del campo de desarrollo de sistemas sugiere algo distinto: la gran mayoría, probablemente más del 80 por ciento, de los sistemas que están en operación actualmente no tienen una declaración precisa y actualizada de los requerimientos de usuario que realizan.

Este no es un fenómeno exclusivo del campo de la computación. ¿Cuántas casas de cien años de antigüedad tienen documentos actualizados que describen la instalación eléctrica, la tubería, la calefacción u otros detalles arquitectónicos? La verdad es que a menudo resulta más fácil hacerle una corrección, mejoría o cambio "rápido y sucio" a un sistema existente, que empezar a cambiar el documento de los requerimientos y luego propagar dicho cambio al documento de diseño y la implantación misma. Esto sucede sobre todo si se necesita hacer el cambio para arreglar un

problema inmediato, presionante y urgente.[1] "Ya cambiaremos los documentos más tarde", dice el encargado de mantenimiento, "pero primero tenemos que arreglar el problema mismo". La documentación es lo último que se quiere hacer, y muchas veces no se hace.

Los sistemas de información tienen una característica importante en cuanto al mantenimiento: duran más quienes los desarrollan originalmente. Esto también se da para las casas; ni el arquitecto ni el usuario final de una casa victoriana construida en 1880 están disponibles para consultarles hoy en día algo. También sucede para muchos sistemas de información; después de 10 o 20 años, el sistema está siendo empleado por usuarios de tercera generación (de los cuales muchos no tienen idea del por qué se desarrolló para empezar) y está siendo mantenido por programadores de mantenimiento de tercera generación (de los cuales algunos ni idea tienen de por qué quienes originalmente lo desarrollaron adoptaron esa estrategia de diseño en particular).[2] Esta es la razón por la cuál Nicholas Zvegintzov describe a los programadores de mantenimiento como "ingenieros en reversa de la sociedad".

Hay otra cosa importante sobre los sistemas de información: tienden a ser complejos desde el principio, y se vuelven cada vez más complejos al pasar años de mantenimiento. Si el sistema fuera sencillo (por ejemplo, unas 250 instrucciones de Pascal), entonces se mantendría fácilmente aún si no tuviera documentación. Pero un sistema típico tiene por lo menos 100,000 instrucciones; muchos de los más grandes que se mantienen en la actualidad tienen más de 500,000, y algunos tienen más de un millón de instrucciones. Ningún individuo puede entender la complejidad de un sistema tal, *sobre todo* si 1) no estuvo involucrado en el desarrollo del sistema original y, 2) no se documentaron los requerimientos y el diseño originales. Y sin embargo eso es precisamente lo que pedimos de la mayoría de los programadores de mantenimiento.[3]

Existen docenas, si es que no cientos, de ejemplos de organizaciones con problemas severos de mantenimiento del tipo descrito anteriormente. Casi cualquier organización importante que empezó a computarizarse hace 20 años ahora se enfrenta a sistemas de 20 años de antigüedad cuya implantación es un misterio y, peor aún, cuyos requerimientos de usuario son un misterio.

---

1 Una encuesta en [Lientz y Swanson, 1980] mostró que aproximadamente un 9% de todo trabajo de mantenimiento consiste en "reparaciones de emergencia de programa".

2 Un estudio hecho por el fabricante británico de computadoras ICL en los años 70 reveló que a un sistema típico lo mantienen *siete generaciones* de programadores de mantenimiento antes de ser desechado finalmente. Esto sugiere que mucho de lo que llamamos programación de mantenimiento podría describirse más precisamente como arqueología.

3 Uno de los ejemplos más extremos de un sistema grande y complejo con requerimientos continuos de mantenimiento es el proyecto de Estación Espacial que actualmente desarrolla la NASA. Su propósito es colonizar e industrializar la porción cercana del sistema solar. Está programado para dentro de 30 años, y requerirá mantenimiento permanente.

La única solución a esta crisis en el futuro es mantener documentación precisa y actualizada por la duración del sistema mismo. ¿Pero, cómo hacerlo?

24.2 PRERREQUISITOS NECESARIOS

No se puede mantener actualizados un sistema y su documentación asociada a menos que ésta sea precisa. Este es un punto de partida: debe asegurarse que cuando un nuevo sistema se ponga en operación todos los documentos relacionados estén completos y sean consistentes, actualizados y precisos.

A lo largo de este libro hemos discutido las características de un modelo preciso de los requerimientos del usuario, además de las reglas a seguir para asegurar que el modelo del sistema sea completo e internamente consistente. Para que pueda mantenerse con éxito, deben obligatoriamente seguirse estas reglas, y la persona independiente o grupo que lo haga debe certificar que los documentos sean precisos antes de poner el sistema en operación.

Además de certificar que los documentos mismos sean precisos, debe asegurarse que exista un mecanismo para hacerles cambios posteriores. De nada servirá que la especificación estructurada se haya inscrito en tablas de piedra como registro permanente para generaciones futuras; la especificación debe verse como un documento viviente, sujeto a cambios continuos, aunque controlados.

24.3 COMO HACERLO

La primera y más fundamental de las reglas para el mantenimiento de sistemas es la siguiente: *cualquier* cambio propuesto al sistema operacional existente debe, en todos los casos, empezar con un examen de su impacto sobre las especificaciones o requerimientos del sistema. Esto debe hacerse en todos los casos que se mencionan a continuación, y con cualquier otro cambio propuesto al sistema:

- El usuario decide que quisiera añadir una nueva función al sistema actual.
- El usuario no está contento con la forma en la que se realiza alguna función actual y quiere cambiarla.
- El usuario quiere un nuevo reporte de salida además de los que ya tiene.
- El usuario quiere modificar el formato u organización de un reporte de salida existente.
- Los programadores de mantenimiento desean recodificar un módulo para hacerlo más eficiente.
- El departamento de operaciones ha anunciado que planea mejorar los sistemas de cómputo actuales de la organización y se necesitarán algunos cambios de programación.

- El usuario se queja de que el sistema produce salidas incorrectas para ciertas combinaciones de entradas.
- La organización de desarrollo de sistemas ha decidido que Ada se adopte como nuevo lenguaje de programación. Se hacen planes para convertir todo el software existente a Ada.
- Se requiere que el sistema mande salidas a una nueva dependencia gubernamental, que no existía cuando se desarrolló originalmente.

Cualquier cambio como éstos debe ilustrarse, documentarse y ser verificado con el usuario, haciendo al modelo del sistema los cambios pertinentes. Esto usualmente se hace llenando una forma conocida como solicitud de cambio del sistema. El cambio de mantenimiento puede involucrar alguno, o todos, los siguientes detalles:

- Añadir terminadores nuevos al diagrama de contexto, o eliminar anteriores. Los flujos de datos entre el sistema y sus terminadores podrían añadirse, eliminarse o cambiarse. Las funciones que previamente desempeñaban los terminadores podrían efectuarse ahora dentro del sistema; de manera inversa, ciertas funciones que el sistema hacía podrían considerarse ahora fuera de él y dentro de los dominios de un terminador.
- Puede ser necesario añadir nuevos eventos a la lista, o eliminar otros.
- Si el cambio es substancial, puede modificarse la declaración de propósitos en el modelo ambiental.
- Los modelos de flujo de datos, modelos de entidad-relación o modelos de transición de estados pueden requerir cambios.
- Las especificaciones de proceso y el diccionario de datos pueden necesitar modificarse o refinarse.
- Varios aspectos del modelo de implantación del usuario pueden requerir cambios que involucren la interfase humano-máquina o las restricciones de implantación que se refieren al tiempo de respuesta, etc.

Ningún cambio de éstos vendrá gratis. Es posible que algunos sean mínimos y sólo requieran unos cuantos minutos de trabajo para ser incorporados, es decir, sólo tomaría minutos hacer los cambios necesarios a la especificación y a los programas existentes. Sin embargo, la persona o grupo que realiza los cambios tiene la obligación de escribir una *declaración de impacto:* esto es, una declaración precisa y detallada de los cambios necesarios en la especificación del sistema para poder implantar el cambio propuesto. Además, debe existir una declaración de impacto económico: es decir, una declaración del costo del cambio y el beneficio que se estima que traerá. Es sobre todo importante si la actividad de mantenimiento cambiará el enfoque del sistema.

Desde luego, habrá algunos cambios que no causen impacto en la especificación del sistema: una corrección de programación para arreglar un error, un cambio de codificación para mejorar la legibilidad o la eficiencia del sistema existente, o un cambio del hardware o software existentes (compilador, sistema operativo, sistema de administración de bases de datos, etc.). Sin embargo, incluso en estos casos debe generarse una declaración de impacto económico para que el usuario y la organización de desarrollo de sistemas entiendan los costos y beneficios asociados con dicho cambio.

Cualquier cambio del sistema comúnmente resultará en un cambio del software y/o hardware; también puede resultar en el cambio de los manuales del usuario, procedimientos de operación y varios otros componentes del sistema. *Pero el documento más importante de actualizar es definitivamente la declaración de requerimientos del usuario.* Sin él, los cambios o modificaciones futuros se volverán cada vez más difíciles de hacer; y el cambio a un sistema totalmente nuevo será infinitamente más caro, tardado y doloroso de lo que debería.

No hay duda de que un analista veterano con 20 años de experiencia vería esta petición de especificación de sistema actualizado con ojos enfermos. Después de todo, el proceso de análisis y la tarea de crear una especificación precisa han sido tan difíciles durante tantos años, que la idea de mantenerla permanentemente actualizada parece casi risible.

La respuesta es, a la larga, la automatización. Las estaciones de trabajo automatizadas de análisis de sistemas del tipo descrito en el Apéndice A están disponibles a costos accesibles, y representan una dramática mejoría sobre la tecnología usada por la mayoría de los analistas hoy, como los sistemas de procesamiento de palabras representan una dramática mejoría sobre la máquina de escribir eléctrica de los años 60. Hay planes más ambiciosos para desarrollar ambientes de ingeniería de software integrados que abarquen todo y que sirvan de depósito central para *todos* los documentos asociados con el desarrollo de un sistema. Sin embargo, tal tecnología avanzada probablemente no se desarrolle por completo hasta mediados de los años 90.

Sin embargo, queda mucho que hacer aun con la tecnología disponible actualmente. Simplemente no hay excusa para hacer cambios a un sistema existente sin hacer el cambio correspondiente a su especificación. Sin embargo, para que esto funcione se requiere una administración fuerte y disciplinada dentro de la organización.

24.4 RESUMEN

Existe una cantidad creciente de libros sobre el tema del mantenimiento de software, además de por lo menos una sociedad profesional (la Asociación de Mantenimiento de Software en los Estados Unidos) que se ocupa de cuestiones de mantenimiento. El énfasis actual es sobre la administración y refinamiento de programas

existentes, aunque también hay algo sobre el uso de buenas técnicas de diseño para crear programas mantenibles. La industria de desarrollo de sistemas está apenas dándose cuenta de que nunca se podrá tener software mantenible sin especificaciones mantenibles.

REFERENCIAS

1. Bennet Lientz y B. Swanson, *Software Maintenance Management.* Reading, Mass.: Addison Wesley, 1980.
2. James Martin y Carma McClure, *Software Maintenance: The Problem and Its Solution.* Englewood Cliffs, N.J.: Prentice-Hall, 1983.
3. Girish Parikh, editor, *Techniques of Program and Systems Maintenance.* Lincoln, Neb.: Ethnotech, Inc., 1980.
4. Carma McClure, *Managing Software Development and Maintenance.* Nueva York: Van Nostrand Reinhold, 1981.
5. Robert Glass and R.A. Noiseux, *Software Maintenance Guidebook.* Englewood Cliffs, N.J.: Prentice-Hall, 1981.
6. Ned Chapin, "Software Maintenance with fourth-generation Languages", *ACM Software Engineering Notes,* volumen 9, número 1, enero 1984, pp. 41-42.
7. R.N. Britcher y J.J. Craig, "Using Modern Design Practices to Upgrade Aging Software Systems," *IEEE Software,* volumen 3, número 3, mayo de 1986, pp. 16-24.
8. Salah Bendifallah y Walt Scacchi, "Understanding Maintenance Work", *IEEE Transactions on Software Engineering,* volumen SE-13, número 3, marzo de 1987.

PREGUNTAS Y EJERCICIOS

1. ¿Por qué es necesario mantener la especificación del sistema incluso después de haberlo desarrollado por completo?
2. ¿Por qué suelen los programadores de mantenimiento hacer cambios a un sistema operacional sin actualizar los documentos de especificación asociados?
3. Proyecto de Investigación: Averigüe la edad promedio de los sistemas que operan en su organización. Algo aún más interesante: investigue cuánto tiempo más se espera que continúen operando antes de ser reemplazados.
4. Proyecto de Investigación: Averigüe cuántos de los sistemas que actualmente están en operación tienen especificaciones actualizadas. ¿Están al tanto de estas estadísticas los analistas y usuarios de su organización?

5. ¿Qué dificultades se ocasionan si un sistema lo usan usuarios y lo mantienen programadores que no estuvieron involucrados con su desarrollo original?
6. Dé seis ejemplos de los tipos de cambios que el usuario podría desear hacerle a un sistema operacional.
7. ¿Por qué es posible que se hayan añadido terminadores nuevos al diagrama de contexto durante el mantenimiento de un sistema?
8. ¿Por qué es posible que se hayan añadido acontecimientos nuevos a la lista durante el mantenimiento del sistema?
9. ¿Bajo qué condiciones podría necesitar cambios la declaración de propósitos de un sistema durante su mantenimiento?
10. ¿Qué es una declaración de impacto? ¿Por qué es importante?
11. ¿Por qué ha sido difícil mantener actualizados los documentos del análisis de sistemas (el modelo esencial del sistema) en la mayoría de las organizaciones?
12. ¿Qué tipo de desarrollo tecnológico es probable que se requiera para asegurar que los documentos del análisis de sistemas se mantengan actualizados?

25 EL FUTURO DEL ANALISIS ESTRUCTURADO

> Todo intento de predecir el futuro con algún detalle parecerá risible dentro de algunos años. Este libro tiene un fin más realista y, sin embargo, más ambicioso. No trata de describir *el* futuro, sino de definir los límites dentro de los cuales deben estar los posibles futuros. Si consideramos el tiempo que nos queda por delante como un país inexplorado y sin mapas, lo que intento hacer es estudiar sus fronteras y darme una idea de su extensión. La geografía detallada de su interior debe seguir desconocida hasta que lleguemos a ella.
>
> Arthur C. Clarke, *Perfiles del futuro*

A lo largo de este libro hemos visto una evolución de ideas y técnicas en el campo del análisis de sistemas. Caben en tres periodos amplios:

1. El análisis de sistemas convencional, anterior a los años 70, caracterizado por (si acaso) especificaciones tipo novela victoriana que eran difíciles de leer y entender, y casi imposibles de mantener.

2. El análisis estructurado clásico, de mediados de los años 70, a mediados de los 80, como se describe en [DeMarco, 1978], [Gane y Sarson, 1977] y otros. Esto se caracterizó por primeras versiones de modelos gráficos y énfasis en el modelado de las implantaciones actuales de un sistema antes de modelar el nuevo.

3. El análisis estructurado moderno, como lo describen este libro y otros recientes tales como [Ward y Mellor, 1985] y [McMenamin y Palmer, 1984].

Este capítulo resume algunos de los cambios de principal importancia que sucedieron desde la introducción del análisis estructurado clásico a fines de los años 70, y su uso como punto de partida para la discusión de probables cambios en los siguientes 5 a 10 años.

25.1 QUE HA CAMBIADO

Diversos aspectos del análisis estructurado han cambiado gradualmente a lo largo de los últimos 10 años. Las principales áreas de cambio incluyen lo siguiente:

- *Cambios de terminología.* Ahora podemos usar el término modelo ambiental para describir lo que solía llamarse simplemente diagrama de contexto. Esto se debe a que el análisis estructurado clásico no incluía una lista de acontecimientos como parte del modelo formal del sistema. Además, ahora usamos el término *esencial* en lugar de *lógico* para describir un modelo que se concentra en lo que el sistema tiene que hacer, y el término de *implantación* en lugar de *físico* para describir un modelo que se concentra en cómo se desarrollará el sistema. Estos obviamente son cambios mínimos, pero ayudaron a reducir la confusión cuando se habla con usuarios que se preguntan si lo opuesto a un sistema lógico es un sistema ilógico.

- *Partición de acontecimientos.* Uno de los acontecimientos más significativos del análisis de sistemas, que se discutió en los Capítulos 20 y 21, fue el uso de una lista de acontecimientos para guiar el desarrollo inicial del modelo de comportamiento. Esto reemplaza al enfoque de la partición estrictamente descendente del diagrama de contexto en un diagrama de flujo de datos de alto nivel (figura 0), y de éste en niveles inferiores, etc. Aunque este enfoque descendente no está mal en sentido alguno, ha sido difícil para muchos analistas; el enfoque de partición por acontecimientos, que es un enfoque horizontal, ha mostrado ser de más éxito en muchos proyectos de análisis.

- *La desenfatización del modelado físico actual.* Como se señaló en el Capítulo 17, existe un buen número de razones por las cuales el analista se pudiera sentir tentado a modelar la implantación actual de un sistema. Pero una y otra vez hemos encontrado que ésta es una tentación peligrosa y que el analista pasa más tiempo con esta actividad del que de hecho amerita. Aunque no excluimos el modelo físico actual, sí tratamos de desenfatizarlo y de evitarlo. El análisis estructurado moderno enfatiza el desarrollo, tan pronto como sea posible, de un modelo esencial del sistema nuevo.

- *Herramientas de modelado en tiempo real.* El análisis estructurado clásico estaba destinado principalmente al desarrollo directo de sistemas de negocios; no consideraba interrupciones, señales ni cuestiones de tiempo. Sin embargo, muchos de los sistemas complejos actuales sí incluyen una variedad de cuestiones de tiempo real, y las herramientas de modelado del análisis se han extendido acorde con ello. Se han utilizado flujos y procesos de control para aumentar los diagramas de flujo de datos, y se

han introducido los *diagramas de transición de estados* como nueva herramienta para mostrar los requerimientos dependientes del tiempo de un sistema.

- *Integración más cercana del modelado de procesos y de datos.* El análisis estructurado usaba *diagramas de estructura de datos* para modelar las relaciones entre almacenes en el diagrama de flujo de datos. Sin embargo, las relaciones a menudo se complicaban por la notación, y la notación tendía a causar discusiones intensas y debates sobre el diseño e implantación de una base de datos física. El diagrama de entidad-relación que se presenta en este libro proporciona un modelo más lógico o conceptual de los datos requeridos por el sistema, y permite que las relaciones entre entidades de datos se describan rigurosamente y con detalle. Además, las reglas de balanceo que se discuten en el Capítulo 14 aseguran que el modelo de datos (que se documenta con DERs) sea completamente consistente y compatible con el modelo del proceso (que se documenta con DFDs y especificaciones de proceso).

Es importante estar familiarizado con estos cambios, o podría encontrarse trabajando para una organización que aún no los ha incorporado a sus estándares; cuando se estaba escribiendo este libro, muchas organizaciones grandes que visité en los EUA todavía estaban usando métodos de desarrollo de sistemas que tenían por lo menos 10 años de antigüedad.

25.2 ACONTECIMIENTOS FUTUROS EN EL ANALISIS ESTRUCTURADO

Nadie puede decir que conoce el futuro con detalle; cuando más, como señala Arthur C. Clarke en la introducción de este capítulo, se desea encontrar algún señalamiento del futuro. Los acontecimientos recientes sugieren un número de tendencias que probablemente continúen hasta bien acentrada la siguiente década. Incluyen lo siguiente:

25.2.1 Mayor difusión del análisis de sistemas

Las computadoras, como todos lo sabemos, se están convirtiendo en una parte de la vida de todo mundo. Consecuentemente, encontramos que un segmento cada vez mayor de la sociedad está aprendiendo a usarlas y a hablar acerca de ellas; de más importancia aún (en el contexto de este libro) es el hecho de que muchas personas se están familiarizando con el análisis estructurado y otros aspectos de la ingeniería de software. Sobre todo estoy interesado en el impacto futuro del análisis estructurado sobre tres grupos: los niveles superiores de administración en nuestras organizaciones gubernamentales y de negocios, los niños, y profesionales de la computación en los países del tercer mundo.

En la mayor parte de las organizaciones grandes normalmente es el caso que los niveles superiores de administración están formados por gente de más de 40 años, o de 50 o 60. Esto significa que recibieron su educación y pasaron los años formativos de su carrera hace 20, 30 o incluso 40 años. Las computadoras ciertamente existían hace 20 años (o incluso hace 30), pero no eran ampliamente accesibles y no eran parte de la tecnología o la cultura con la que crecieron. Pero eso está empezando a cambiar; se comienza a ver niveles superiores de administración que 1) empezaron su carrera en la organización de proceso de datos o de sistemas de información[1], o bien 2) empezaron su carrera en alguna otra parte de la organización (por ejemplo, ventas, contabilidad o manufactura) cuya operación diaria se vio dramáticamente afectada por la tecnología de las computadoras. Esto significa que, como analista, podrá esperar que los niveles superiores de la administración estén cada vez más conscientes de la importancia estratégica de los sistemas de proceso de información de su organización, y que cada vez estarán más interesados en ver modelos de alto nivel de los sistemas nuevos importantes. Si trata de mostrar un diagrama de flujo de datos al vicepresidente de su organización hoy en día, lo más probable es que no lo entendería, y tampoco entendería el por qué *debiera* entenderlo. Dentro de los próximos 5 años, creo que los niveles superiores de administración empezarán a darse cuenta que es tan importante poder leer (y criticar) un modelo de sistema como leer y criticar una hoja de balance o una declaración de pérdidas y ganancias.

Los niños también se familiarizarán cada vez más con el análisis estructurado en los próximos años. La programación y diseño estructurados ya se están enseñando en el nivel de la preparatoria en algunas partes de los Estados Unidos. El análisis estructurado, que alguna vez fuera tema de seminarios de nivel de posgrado, ahora se enseña en el tercer y cuarto años de las licenciaturas en computación y de administración, y pronto será parte de una materia estándar de primer año. La división alguna vez fue tema avanzado y ahora se le enseña a los niños pequeños; de manera similar, el análisis estructurado será un tema que los niños aprenderán durante su proceso educativo normal.

Se estima que un niño nacido en 1980 terminará la escuela secundaria hacia fines de siglo habiendo escrito unos 10,000 renglones de código; esto es, a grandes rasgos, equivalente a 2 años de experiencia de tiempo completo en programación para el programador adulto de hoy. Además de esta experiencia en programación, se puede esperar que la generación actual de niños tenga cada vez más experiencia en análisis y diseño de sistemas. No toda esta generación acabará escogiendo como carrera la de analista o programador; de hecho, sólo una pequeña parte escoge-

1 Tres ejemplos de esto son John Reed, el actual presidente de Citicorp; Richard Crandall, jefe de American Airlines, y Frank Lautenberg, anterior presidente de ADP (la compañía de servicios de nómina) y actualmente uno de los dos senadores del estado de Nueva Jersey en los EUA. Existen también diversos ex-programadores y ex-analistas que son miembros del congreso de los Estados Unidos.

rá este camino. Pero el resto de los niños de hoy, ya sea que escojan ser contadores, o ingenieros, agentes de ventas, maestros o políticos, formarán una comunidad de *usuarios inteligentes* de sistemas de información; los usuarios sabrán mucho más acerca de lo que se puede esperar de un analista y qué preguntarle. Al parecer, gran parte de nuestro trabajo actual radica en la dificultad de tratar con usuarios ignorantes, y esto posiblemente será superfluo en un futuro.

Existe otro aspecto de la creciente difusión del análisis estructurado que debe mencionarse: su impacto sobre la industria de software del tercer mundo. En la última década la competencia internacional se volvió más intensa en muchas industrias manufactureras, y la industria estadounidense ha perdido terreno (o ido a la quiebra) al enfrentarse a las japonesas, coreanas, chinas o brasileñas que ofrecen productos de alta calidad a precios competitivos. *El mismo fenómeno se está empezando a dar en la industria de desarrollo de sistemas.* Las técnicas de ingeniería de software, incluyendo las técnicas de análisis estructurado que se discuten en este libro, pueden ayudar a una organización competitiva a desarrollar sistemas con un grado de productividad diez veces mayor que la de muchas organizaciones norteamericanas y con un nivel de calidad (expresado en términos de tiempo medio entre fallas o número de errores) *cien veces mayor* que la de las industrias norteamericanas del mismo tipo.[2] Y dado que, en cada vez mayor grado, *todos* los productos y servicios dependen de sistemas de información basados en computadora, esto tiene implicaciones profundas para toda la industria estadounidense.

25.2.2 Proliferación de herramientas automatizadas

A lo largo de este libro hemos mencionado la posibilidad de usar herramientas basadas en estaciones de trabajo para automatizar diversos aspectos del análisis estructurado, sobre todo las actividades de creación de modelos gráficos de sistemas y su revisión para verificar que estén completos y sean correctos.

El Apéndice A describe las características de muchos paquetes de herramientas de este estilo disponibles en el mercado cerca de 1990, lo mismo que las características que es probable que incluyan durante los próximos años. Lo importante es que estos productos existen ahora y que se volverán cada vez más poderosos durante la siguiente década.

Pero pocos analistas usan estas herramientas hoy. En 1987 se estimaba que menos del 2 por ciento de los analistas de sistemas de Norteamérica y Europa tenían acceso personal a una herramienta automatizada apropiada. Esto significa que la organización típica de desarrollo de sistemas tiene una o dos estaciones de trabajo para el dibujo de diagramas de flujo de datos, diagramas de entidad-relación, etc.

2 Para una discusión referente a esto vea D. Tajima y T. Matsubara, "The Computer Industry in Japan", *Computer,* volumen 14, número 5 (mayo de 1981), pp. 89-96.

Estas estaciones pueden ser compartidas por toda la organización de cien o más personas, pero más a menudo las usa algún equipo de proyecto aislado que tuvo la suerte, tenacidad o visión para invertir en esta tecnología.

Para 1990 se estima que aproximadamente un 10 por ciento de los analistas de Norteamérica y Europa (y otras partes civilizadas del mundo también) tendrán sus propias estaciones de trabajo personales. Y para mediados de los años 90 es razonable esperar que por lo menos un 50 por ciento de los analistas las tengan. Cuando se haya alcanzado la masa crítica será razonable argumentar que nuestro enfoque del análisis de sistemas habrá cambiado fundamentalmente, porque la mayoría de los analistas tendrán herramientas nuevas poderosas. Como analogía es interesante discutir las mejorías que se pueden lograr en el área de la carpintería utilizando una sierra eléctrica en lugar de una manual, pero el asunto no tiene caso si sólo el 1 por ciento de los carpinteros cuentan con electricidad. Y el poder de una herramienta dada sí afecta la forma en la que trabajamos con el mundo que nos rodea; Craig Brod recalcó este punto elocuentemente en una obra clásica titulada *Technostress* ([Brod, 1984]):

> Las herramientas siempre han impulsado grandes cambios en las sociedades humanas. Las herramientas nos crean tanto como nosotros a ellas. La lanza, por ejemplo, contribuyó con mucho más que simplemente extender el alcance del cazador; cambió la forma de caminar y el uso de sus brazos. Fomentó la mejor coordinación de manos y ojos; llevó a organizaciones sociales para rastrear, matar y traer presas grandes. Amplió la brecha entre el cazador inexperto y el experto e hizo más importante el intercambio de información, a medida que las expediciones de caza se volvían cada vez más complejas. Hubo otros efectos, menos obvios: los cambios en la dieta de las sociedades cazadoras llevaron a compartir los alimentos y a la formación de nuevas relaciones sociales. El valor de la artesanía aumentó. La gente comenzó a planear por adelantado, almacenando armas para reusarlas luego. Todas estas exigencias relacionadas con nuevas herramientas, a su vez, llevaron a un mayor desarrollo del cerebro. La complejidad del cerebro llevó a nuevas herramientas, y éstas hicieron que los cerebros aún más complejos fueran una ventaja para la supervivencia de la especie.

Parece evidente hasta aquí que la tecnología de las herramientas automatizadas continuará avanzando durante los próximos 10 años. El paquete de herramientas del analista de mediados de los años 90 tendrá casi seguramente complejas facilidades para revisión de errores, además de la capacidad de generar código e incluso sugerir (utilizando técnicas de inteligencia artificial) posibilidades para reutilizar código de las bibliotecas de software.

25.2.3 El impacto de los desastres de mantenimiento

En el Capítulo anterior mencionamos al usuario del modelo de análisis estructurado para facilitar el mantenimiento y la modificación de sistemas. Pero ésta es

una cuestión que a menudo parece abstracta, filosófica, y *políticamente sin importancia* durante la fase de desarrollo de un proyecto, donde al parecer el énfasis consiste en entregar el sistema al usuario. Entregarle de preferencia un sistema que opere y, con suerte, que sea el que el usuario quería. Pero, si no fuera así, entonces *cualquier* sistema que aparentemente trabaje y satisfaga por lo menos algunos de los requerimientos. La realidad política simple es que algunos administradores de alto nivel en las organizaciones no aprecian aún del todo la importancia del análisis estructurado y de los modelos rigurosos y formales de los sistemas. Incluso entre las filas del proceso electrónico de datos, el análisis estructurado no goza de la importancia que se adjudica a la necesidad política de entregar a tiempo al usuario un sistema que opere (o que supuestamente lo haga).

Como sugerí anteriormente, la situación cambiará al familiarizarse cada vez más la comunidad usuaria con la tecnología de las computadoras, y al intensificarse cada vez más la competencia de los países del tercer mundo. Pero existe otro fenómeno que resaltará de manera dramática la necesidad de los modelos actualizados de sistemas, mantenidos tan diligentemente como el código fuente: *los desastres de mantenimiento que ocasionen el colapso de los sistemas actuales.*

En el caso extremo, esto puede suceder debido a que un sistema existente, grande, complejo y no documentado, aborte o se detenga completamente, sin que nadie pueda determinar cómo arreglarlo. Pero esto es poco probable; es más probable que la *causa* de la falla se identifique y simplemente se excluya. Correrá entonces la voz de que: "Ya no se puede ingresar una transacción de tipo X25 al sistema porque causa problemas".

No, la causa más probable de un gran desastre de mantenimiento será *la imposibilidad total de hacer alguna modificación necesaria y urgente a un sistema existente.* A veces una nueva legislación o política de gobierno es la que obliga a tales cambios, pero puede ser también que se requieran debido a cambios en el ambiente de negocios o a la situación competitiva.

Ya muchas organizaciones enfrentan este problema; muchos de sus sistemas que se diseñaron a fines de los años 60 o comienzos de los 70 están al borde del colapso, y un día sí se colapsarán. Parte del problema está relacionado con la *implantación* del sistema, es decir, un código que se ha remendado tantas veces que ya no es posible determinar con precisión cómo opera el sistema.

Pero el problema mayor, en mi opinión, es que nadie sabe o recuerda *qué* se supone que deben hacer estos sistemas en realidad. Los programadores de mantenimiento de la tercera generación interactúan con usuarios de tercera generación para discutir posibles cambios a un sistema cuyos requerimientos originales son un misterio para ambos. En este ambiente, es inevitable que los programadores de mantenimiento tarde o temprano se rindan y se rehusen a realizar más cambios.

Cuando se enfrenta con un problema de este tipo, la administración probablemente tendrá una reacción de "reflejo": se formarán comités, se impondrán estándares y se promulgarán nuevos procedimientos. Así como se ha visto a jefes de gobierno reaccionar ante problemas de desperdicios tóxicos, derrames de combustible y otros, sólo *después* de que el desastre ocurre, pienso que muchos administradores de primer nivel de negocios y de gobierno reaccionarán al problema de los inexistentes modelos de sistemas sólo después de que ocurra un problema importante de mantenimiento.

25.2.4 El matrimonio del análisis estructurado y la inteligencia artificial

En la actualidad se está dedicando mucha atención a la inteligencia artificial en los negocios, el gobierno y la industria computacional: sistemas expertos, sistemas de lenguaje natural, robótica y muchos campos relacionados. Aunque en un tiempo se consideraba a la inteligencia artificial como un tema académico con poca aplicación práctica, y aunque se implantaba en hardware exótico con lenguajes no familiares de programación tales como LISP y PROLOG, ahora se está convirtiendo cada vez más en un tópico de interés común, sobre todo el área de los sistemas expertos, que pueden duplicar la conducta de expertos humanos en ciertos campos definidos precisamente.

Existen cada vez más paquetes de software y textos de inteligencia artificial para ambientes basados en COBOL y PCs (véase por ejemplo [Taylor, 1985], [Derfler, 1985], [Webster, 1985], [Keller, 1986] y [Rose, 1985]. Cada vez más aplicaciones de inteligencia artificial se están adentrando en el mundo de los negocios, desde diagnósticos médicos hasta exploración petrolera o evaluaciones de bolsa, o incluso planeación de impuestos.[3]

¿Qué tiene esto que ver con el análisis estructurado? La conexión entre inteligencia artificial y sistemas expertos funciona en ambos sentidos: el análisis estructurado puede servir de auxiliar en el proceso de construcción de un sistema experto, y la tecnología de los sistemas expertos puede servir de auxiliar en el proceso de realizar el análisis estructurado.

Cuando se construye un sistema experto, a menudo dominan tres aspectos: la interfase humano-máquina, la representación del conocimiento, y la máquina de inferencias que evalúa e interroga a la base de conocimientos. La interfase humano-máquina puede involucrar entradas en lenguaje natural (inglés, francés, alemán,

---

3 Una gran cantidad de trabajo artificial se realiza en hardware especializado utilizando lenguajes de programación especializados como Lisp y Prolog. Sin embargo, 1) la mayoría de las compañías preferirían integrar sus aplicaciones de inteligencia artificial con las demás aplicaciones que ejecutan en su computadora principal estándar IBM; 2) la mayoría de los programadores prefieren usar lenguajes como COBOL a lenguajes tan esotéricos como Prolog y, 3) las aplicaciones de inteligencia artificial orientadas a los negocios tendrán que acceder a la base de conocimientos, *que ya existe en la computadora principal.*

etc.) y una combinación de gráficos, texto y sonido como salidas. La base de conocimientos puede expresarse por medio de una serie de reglas **SI-ENTONCES-OTRO,** o como una serie de marcos.[4] La máquina de inferencias puede basarse en un enfoque de encadenamiento progresivo o regresivo y puede implantarse en un "cascarón *(shell)* experto" comercial.

Pero lo importante acerca de todo esto es que los componentes del sistema experto se están convirtiendo en sólo parte de un sistema mayor, por ejemplo, un sistema operacional que alimenta y actualiza una base de conocimientos o que usa las salidas del componente del sistema experto para realizar otras funciones del sistema. Así, las herramientas de modelado del análisis estructurado pueden usarse para ayudar a modelar el sistema global. Pero lo más importante es que significa que durante los próximos 5 a 10 años, usted tendrá que familiarizarse con la tecnología de los sistemas expertos y artificiales para poder ser un analista con éxito. E libro de Keller ([Keller, 1986]) es bueno para empezar, pues muestra muchas de las interacciones entre el análisis estructurado y los sistemas expertos.

En el sentido inverso, la inteligencia artificial puede coadyuvar al *proceso* de análisis estructurado actuando como tutor para guiar a un analista novato en los diversos pasos y procesos descritos en este libro. Se puede uno imaginar fácilmente, por ejemplo, a un "asistente de analista" haciendo una serie de preguntas al analista humano y produciendo luego un diagrama de contexto propuesto o una lista de acontecimientos. Piense: ¿Puede recordar a estas alturas todas las reglas y guías a seguir plasmadas en las últimas cien páginas, ahora que ha llegado al final del libro? ¿Cree que las podrá recordar dentro de un año? ¿No sería bueno tener un sistema experto disponible en una PC que le recordara cómo dibujar los DFD, DER y DTE balanceados?

Aunque todo esto suena muy bien, no debe preocuparse por la posibilidad de que los sistemas expertos estén desplazando a los analistas humanos. Los investigadores en este campo señalan que todos los sistemas expertos con éxito, que van desde el diagnóstico médico hasta el análisis de la bolsa, lo han tenido debido a que se han concentrado en un campo muy limitado de dominio. Un analista con éxito, por lo contrario, realmente necesita ser experto en distintas áreas: debe comprender la tecnología del análisis estructurado que se presenta en este libro; debe entender el área de aplicación del usuario; debe saber mucho de contabilidad, para poder producir cálculos precisos de costo-beneficio; debe ser experto en comunicaciones y psicología cognoscitiva, para que pueda comunicarse de manera efectiva con los diseñadores y programadores. Las estimaciones actuales (véase por ejemplo, [Barstow, 1987]) son que los sistemas expertos podrán ayudar con la labor del analista de sistemas en sistemas sencillos para mediados de los años 90, pero pasará del final de este siglo antes de que la tecnología de los sistemas expertos sea capaz de realizar el análisis de sistemas más grandes.

4 Ver [Keller, 1986] para una descripción de los marcos.

25.2.5 El impacto de las nuevas generaciones de hardware de las computadoras

Las compañías particulares, universidades, organizaciones de investigación y militares, y gobiernos de todo el mundo están gastando enormes sumas de dinero con el objeto de producir *hardware* dramáticamente más poderoso durante los siguientes 10 ó 15 años. Una estimación reciente, dada por Gordon Bell, de la Fundación Nacional de Ciencias de los Estados Unicos, en una plática en la 1986 Fall Joint Computer Conference, es que la tecnología del hardware de las computadoras mejorará en un factor de 10 dentro de los próximos 5 años; y luego en otro factor de 10 durante los 5 años siguientes, y otro más durante los 5 siguientes. En la misma conferencia, el físico ganador del premio Nobel Ken Wilson hizo una predicción aún más optimista: un factor de mejoría de 100 en los próximos 5 años, seguido de otro factor de 100 en los siguientes 5, y nuevamente por otro de 100 en los siguientes 5. Por ello, dos científicos importantes sugieren que dentro de los siguientes 10 o 15 años podremos esperar que el hardware sea entre 10^3 a 10^6 veces más poderoso que el de las computadoras de hoy.

¿Qué tiene esto que ver con el análisis de sistemas? Simplemente esto: el asunto de definir los requerimientos del usuario para un sistema de información tiene que hacerse dentro del contexto de lo que el usuario y el analista creen *posible* lograr con la tecnología disponible. Pero lo que creemos posible se basa en gran medida en lo que *sabemos* acerca de la tecnología de las computadoras. Puede argumentarse que la mayoría de los usuarios finales y analistas ni siquiera empiezan a hacer uso de la tecnología de hardware existente, así que, ¿qué harán con una tecnología un millón de veces más poderosa?

Experiencias pasadas con otros avances tecnológicos, por ejemplo en el campo de la comunicación (desde señales de humo hasta telégrafo y teléfono, etc.) y del transporte (desde caminar hasta el empleo de caballos, autos o aeroplanos, etc.) da una pista; nuestra primera reacción a la tecnología radicalmente mejorada es continuar haciendo lo mismo que hacíamos antes, pero de una manera un poco más fácil y rápida (y más económica, en muchos casos). Sólo más tarde comenzamos a ver aplicaciones enteramente nuevas para la nueva tecnología.

Por ejemplo, considere el campo del transporte, con el cual todos estamos familiarizados. Aunque el transporte en avión es relativamente nuevo (comparado con la historia de la raza humana), ha estado presente durante toda nuestra vida, y tiene un impacto importante sobre nuestras suposiciones, tanto conscientes como inconscientes, tanto explícitas como implícitas, acerca de la manera en que podemos vivir nuestras vidas. Suponga que alguien le dijera mañana, sin embargo, que existe un tren subterráneo supersónico para llevarlo de la costa oriental a la costa occidental de los Estados Unidos a velocidades de 5000 kilómetros por hora.[5] ¿Cómo afectaría

5 Esto no es ciencia-ficción. En el M.I.T. se han hecho propuestas de ingeniería serias para un tren supersónico.

esto a su vida de negocios? ¿Y a su vida social? ¿Qué cambios empezaría a hacer hoy si estuviera razonablemente seguro de que existirá esta tecnología avanzada dentro de los siguientes 3 a 5 años?

Y ésta es precisamente la posición en la que nos encontramos como analistas para el resto de este siglo; cada vez que se nos dé una tecnología de computación dramáticamente nueva, nuestra primera reacción (y la reacción de los usuarios finales también) será reimplantar las aplicaciones existentes de una manera algo más eficiente. *El reto será encontrar aplicaciones completamente nuevas, es decir, usos de la tecnología de computación, completamente nuevos y diferentes a los que actualmente se desarrollan.*

25.3 CONCLUSION

Es importante tener una perspectiva orientada al futuro al terminar de leer este libro y comenzar a practicar el análisis estructurado en el mundo real. Aunque el modelado, la solución iterativa de problemas, la partición descendente, y los demás conceptos que se discuten en este libro casi seguramente serán válidos para el futuro previsible, muchos de los detalles (por ejemplo, la tecnología disponible para apoyar al análisis estructurado, e incluso técnicas tan específicas como la partición por acontecimientos) pueden cambiar o ser reemplazadas.

No debe esperar que el material que aprendió en este libro sea constante, permanente y ajeno al cambio. Como toda ciencia, y sobre todo como todos los demás aspectos de la ciencia de la *computación,* el campo del análisis de sistemas está destinado a continuar cambiando, evolucionando y (con suerte) mejorando durante el siguiente siglo y más allá. Para algunos, es impresionante darse cuenta que la mitad de lo que se aprende en este campo técnico será obsoleto dentro de 5 años. Para otros, y espero que se incluirá entre ellos, es una fuente de renovación y emociones constantes.

Y con esto llegamos al final del libro. Todavía no es un analista veterano, pero debiera tener bastantes herramientas y técnicas para entrar a la profesión sin temor de caer de bruces. Que disfrute la práctica del análisis de sistemas de información que beneficiarán a la sociedad. Y que regrese en menos de 5 años para ver qué ha cambiado. ¡Ciao!

REFERENCIAS

1. Tom DeMarco, *Structured Analysis and Systems Specification.* Englewood Cliffs, N.J.: Prentice-Hall, 1979.
2. Chris Gane and Trish Sarson, *Structured Systems Analysis: Tools and Techniques.* Englewood Cliffs, N.J.: Prentice-Hall, 1978.
3. Arthur C. Clarke, *Profiles of the Future.* Nueva York: Holt, Rinehart, and Winston, 1984.

4. Jared Taylor, "Lightyear's Ahead of Paper", *PC Magazine,* abril 16, 1985.
5. Frank Derfler, "Expert-Ease Makes Its Own Rules", *PC Magazine*, abril 16, 1985.
6. Robin Webster, "M.1 Makes a Direct Hit", *PC Magazine,* abril 16, 1985.
7. Robert E. Keller, *Expert Technology: Development and Application,* Englewood Cliffs, N.J.: YOURDON Press, 1986.
8. Frank Rose, *Into the Heart of the Mind.* Nueva York: Harper & Row, 1985.
9. D. Barstow, "Artificial Intelligence and Software Engineering", *Proceedings of the 9th International Conference on Software Engineering.* Washington D.C.: IEEE Computer Society Press, marzo de 1987.
10. Paul Ward y Steve Mellor, *Structured Development of Real-Time Systems.* Nueva York: YOURDON Press, 1986.
11. Steve McMenamin y John Palmer, *Essential Systems Analysis.* Nueva York: YOURDON Press, 1984.
12. Craig Brod, *Technostress.* Nueva York: John Wiley, 1984.

PREGUNTAS Y EJERCICIOS

1. ¿Cuáles son las tres etapas en la evolución del análisis de sistemas que se han dado a lo largo de los últimos 20 años?
2. ¿Cuáles son los cinco principales cambios que se han dado en el campo del análisis durante los últimos 10 años?
3. ¿Qué significan, en el contexto de este capítulo, los términos lógico y físico?
4. ¿Qué es partición por acontecimientos? ¿A qué ha reemplazado en el campo del análisis?
5. ¿Por qué se ha desenfatizado el modelaco físico actual en el análisis de sistemas?
6. ¿Qué herramientas adicionales se han añadido al campo del análisis de sistemas para ayudar a modelar sistemas de tiempo real?
7. ¿Qué es un diagrama de estructura de datos? ¿A qué ha reemplazado en el campo del análisis?
8. ¿Cómo están comenzando a afectar las computadoras a los trabajos y actividades de los administradores experimentados en las organizaciones?

9. ¿Por qué tendrá impacto sobre los proyectos de desarrollo de sistemas en los años venideros la enseñanza del análisis estructurado a los niños?

10. ¿Por qué es probable que el análisis de sistemas sea un factor en la competencia internacional entre los EUA, Europa y muchos países del Tercer Mundo?

11. Proyecto de investigación: Qué porcentaje de los programadores y analistas de sistemas en su organización tienen estaciones de trabajo con paquetes de herramientas de análisis?

12. ¿Por qué son importantes las herramientas automatizadas para el análisis de sistemas?

13. ¿Por qué afectarán al análisis estructurado en un futuro los desastres de mantenimiento?

14. ¿Qué relación es probable que veamos entre la inteligencia artificial y el análisis estructurado en un futuro?

15. En qué porcentaje, o múltiplo, se espera que el hardware de las computadoras mejore en los siguientes 10 o 15 años?

16. ¿Por qué tendrán impacto sobre la manera en que se realiza el análisis de sistemas las mejorías continuas en el hardware?

APENDICES

A HERRAMIENTAS AUTOMATIZADAS

El hombre es un animal que usa herramientas... Sin ellas nada es, con ellas lo es todo.

Thomas Carlyle
Sartor Resartus, Libro 1, capítulo 4

A.1 LOS ANTECEDENTES DE LAS HERRAMIENTAS AUTOMATIZADAS

Una herramienta automatizada puede definirse como cualquier cosa que reemplace la labor manual del programador, del analista de sistemas, o incluso del usuario final que de alguna manera debe comunicar sus requerimientos a los profesionales de las computadoras. Por ello, hay muchas cosas que pudieran considerarse como herramientas:

- *Lenguajes de programación de alto nivel,* que van desde COBOL y Pascal hasta los lenguajes actuales de cuarta generación que permiten al programador usar instrucciones de alto nivel, parecidas al inglés, que se traducen automáticamente a las instrucciones primitivas de bajo nivel que entiende la computadora.

- *Listados de referencia, programas "embellecedores de impresión",* y otros programas de aplicación que ofrecen al programador información adicional estática acerca de su programa.

- *Herramientas de prueba, de corrección de errores, simuladores, etc.,* que proporcionan al programador información acerca del comportamiento dinámico de su programa mientras se *ejecuta.* Las herramientas de modelado permiten al programador crear una gran variedad de casos de

prueba para asegurar que el programa se pruebe efectivamente. Las herramientas de corrección permiten rastrear errores cuando se sabe que algo anda mal. Los simuladores proporcionan una representación más visual y gráfica de la ejecución del programa, por ejemplo, mostrándolo en forma de diagrama de flujo en una pantalla de video y simulando su comportamiento a la vez que se ejecuta, mostrando el flujo de control a través del diagrama de flujo.

- *Terminales de tiempo compartido* que reemplazan a los ambientes de desarrollo por lote. Esta batalla se lidió y ganó hace 15 años en la mayoría de las organizaciones de software, pero es importante darse cuenta de que una terminal de tiempo compartido es una herramienta. En los años 60 y comienzos de los 70, los programadores tenían que escribir programas, manualmente, en grandes cuadernos de codificación; las instrucciones se perforaban en tarjetas (¿recuerda las tarjetas perforadas?) y luego se metían a la computadora a media noche. Si algo andaba mal (porque el programador escribió una instrucción sintácticamente incorrecta o porque el operador encargado de la perforación lo hizo mal), a la mañana siguiente le esperaba al programador un reporte de error. Y el ciclo volvería a comenzar. Todo eso desapareció para mediados de los años 70 en la mayoría de las organizaciones: el programador teclea su programa directamente en una terminal de tiempo compartido, que comparte con cientos de programadores y/o usuarios finales más. El programa se revisa contra errores sintácticos en el acto, y se prueba y corrige en el acto. Hoy es difícil imaginar cualquier otro ambiente. Pero esto se debe en parte a que se pueden adquirir terminales simples por menos de 500 dólares estadounidenses. Hace diez años, el costo andaba por los 3,000 o más, y nadie estaba del todo seguro si el programador ameritaba una inversión así.

- *Computadoras personales que permiten el desarrollo de programas fuera de línea.* Actualmente, se invierten 3,000 dólares estadounidenses en una computadora personal. Las terminales simples son aceptables, pero sólo si la computadora personal a la cual están conectadas proporciona un tiempo de respuesta lo suficientemente rápido y consistente como para permitir a los programadores trabajar productivamente. Muchos sistemas simplemente no pueden; para la entrada más trivial proporcionan una respuesta en 5 segundos, y en 10 segundos a entradas más significativas. Una alternativa atractiva es una computadora personal dedicada, que el programador puede usar para crear un programa y hacerle correcciones y revisiones apropiadas utilizando un programa estándar de procesamiento de palabras, para compilarlo y ver si existen errores de sintaxis que la computadora principal rechazaría, y para realizar algunas pruebas fuera de línea.

- *Paquetes de control de programas fuente* que evitan que el programador haga cambios no autorizados a las versiones oficiales de un programa a media noche. En un proyecto grande de programación, una de las dificultades es la administración de la configuración: asegurar que exista un control firme sobre las diversas partes del sistema final. Cada programador trabaja con su propia parte y puede requerir docenas de revisiones antes de terminarla. Pero en dicha parte laboran docenas de programadores más. La anarquía prevalece a menos de que todo mundo sepa cuál versión de cuál parte se va a considerar como versión oficial. Un paquete de control de código fuente es como un bibliotecario automatizado: evita el acceso no autorizado a documentos oficiales.

- *El análisis de sistemas y herramientas automatizadas de diseño.* Las herramientas descritas anteriormente se ocupan principalmente de la labor de escribir programas (es decir, decidir las instrucciones de COBOL o de FORTRAN requeridas para resolver un problema bien definido). Pero no es en esto donde existe la principal dificultad en la construcción de un sistema de software. El verdadero problema aparece en las etapas tempranas del análisis de sistemas (al tratar de determinar qué debe hacer el sistema) y del diseño (al tratar de determinar cuál debe ser su arquitectura global). Comienzan a verse herramientas para ayudar a los analistas y diseñadores de sistemas.

La mayoría de las herramientas descritas anteriormente han existido durante los últimos 10 o 15 años, y muchas se usan ampliamente en las organizaciones de proceso de información. Otras (las automatizadas) son muy nuevas y recién han empezado a filtrarse en la industria de software a partir de 1987. *Son estas herramientas, en mi opinión, las que tienen la posibilidad de salvar a la industria de software de EUA.*

Como hemos visto a lo largo de este libro, el análisis exitoso de sistemas se apoya fuertemente en *modelos* del sistema que se va a computarizar. Las herramientas del analista y del diseñador se ocupan principalmente del desarrollo eficaz de dichos modelos; por ejemplo, ayudan al analista a construir diagramas gráficos que permiten al usuario final entender lo que el sistema hará para él. Las herramientas automatizadas también permiten al analista y al diseñador asegurarse que el modelo esté completo y sea preciso y consistente, de manera que los errores descubiertos durante la fase de programación sólo sean errores de programación, y no un reflejo de continuos malos entendidos entre el usuario final y el analista.[1] Y, finalmente, esas herramientas pueden ayudar al programador a traducir el modelo a

1 Esto es importante, pues sabemos que el 50% de los errores de un proyecto típico de desarrollo de sistemas se deben a malos entendidos entre el usuario final y el analista; un 75% del costo de eliminarlos en un sistema operacional se asocia con errores que se originaron en la fase de análisis.

un programa que funcione. En el futuro podemos esperar que automaticen *completamente* este proceso.

La industria de software ha estado hablando de herramientas de este tipo desde hace 5 años o más; sin embargo no se hizo mucho al respecto. Esto se debió en parte a que la tecnología de la ingeniería de software no se había filtrado aún a la industria, pero más que nada era más bien cuestión de economía. Como señalé anteriormente, no fue sino hasta mediados de los años 70 que la mayoría de las organizaciones de administración de sistemas de información aceptaron la noción de que cada programador debe tener una terminal en su escritorio, y tomó otros 5 años para que muchas las compraran y proporcionaran una computadora principal adicional para el personal de desarrollo de sistemas. (Mientras tanto, dos tres programadores tenían que compartir una terminal, en forma similar a la de dos o tres personas compartiendo el uso de una misma extensión telefónica en una oficina,[2] y el personal completo de desarrollo de sistemas tenía que compartir la computadora principal con cientos de usuarios finales que trataban de hacer trabajo útil en *sus* terminales).

Mientras tanto, las computadoras personales y las estaciones de trabajo gradualmente comenzaban a aparecer en el mercado de los consumidores. A fines de los años 70 y principios de los 80, la mayoría de los programadores ignoraban a las máquinas, pues no eran muy poderosas desde el punto de vista de los estándares de una computadora principal por la cual juzgaban el poder de una computadora. Una estación de trabajo de poder suficientemente grande, capaz de ayudar al analista/diseñador con sus modelos de ingeniería de software costaría entre 50,000 y 100,000 dólares estadounidenses en el periodo de 1980 a 1981, y esto simplemente se salía de las posibilidades de muchas organizacicnes de administración de sistemas de información. Sólo unas cuantas con proyectos y presupuestos enormes consideraban siquiera tal gasto; y entonces lo más que se podía esperar era una estación de trabajo para un departamento entero de cientos de personas. Algunas estaciones de trabajo se desarrollaron en compañías aeroespaciales, compañías contratadas para el desarrollo de sistemas de defensa, y fabricantes de complejas estaciones de trabajo de gráficos, pero la comunidad de administración de sistemas de información ignoraba cuidadosamente el concepto.

Para 1983, las cosas empezaron a cambiar. Las poderosas computadoras personales, con gráficas de alta resolución y capacidad de almacenamiento adecuada habían caído por debajo de la barrera mágica de precio de 10,000 dólares estadounidenses[3] Algunas eran estaciones de trabajo orientadas a la ingeniería,

2 En la opinión de la mayoría de los programadores, es como dos o tres personas compartiendo el mismo cepillo de dientes.

3 Los 10,000 dólares estadounidenses son mágicos pues es la cantidad a partir de la cual se requiere autorización superior para hacer el gasto. Un administrador de proyecto puede ver los beneficios prácticos de una estación de trabajo para ingeniería de software y a menudo puede proporcionar

fabricadas por compañías nuevas y agresivas tales como Apollo Computer y Sun Computer; algunas resultaron ser efímeras, como la computadora Lisa de Apple.

Sin embargo, la mayoría resultaron ser configuraciones a la medida de las necesidades de la inmensamente popular computadora personal de IBM. Al proporcionar una arquitectura abierta, IBM hizo posible que cualquiera construyera una configuración especial para satisfacer sus propias necesidades. Así, la industria de herramientas de software podía construir una estación de trabajo poderosa que consistía de un chasis de PC IBM, una tarjeta gráfica del proveedor A, memoria adicional del proveedor B, y una pantalla de alta resolución del proveedor C.

Esta capacidad de construir una estación de trabajo poderosa con la etiqueta IBM al frente es de importancia crucial en el mercado. La realidad política es que en las organizaciones de negocios como bancos, agencias de seguros y agencias de gobierno no militares, la computadora personal *debe* decir IBM al frente; esto es, desafortunadamente, más importante que la superioridad tecnológica del hardware. A las organizaciones científicas e ingenieriles no les importa tanto la marca de la computadora que usan (aunque muchos preferirían que cualquier computadora personal que compraran se pareciera a una VAX), y a las compañías norteamericanas contratadas para la construcción de sistemas de defensa no les importa qué tipo de computadora usen mientras su costo pueda incluirse en el presupuesto estipulado en el contrato gubernamental.

Existen ahora varias docenas de computadoras en los Estados Unidos y en Europa que construyen productos de software y hardware de estaciones de trabajo para ayudar al analista y al diseñador de sistemas. Una de las fuerzas impulsoras de este tipo de compañías es la creencia de que en los siguientes 5 o 10 años casi cualquier empleado ejecutivo en los Estados Unidos, y sobre todo cualquier programador, analista, diseñador y usuario final de sistemas de cómputo tendrá una poderosa computadora personal en su escritorio. El poder del hardware estará ahí; ahora todo lo que necesitamos hacer es aumentar dicho poder con algunos dispositivos adicionales y algún software poderoso.

La mayoría de los productos pueden ejecutarse en computadoras personales IBM, aunque algunos de los proveedores han escogido las computadoras Apple McIntosh o las más poderosas VAX o Apollo. Casi todos los productos ofrecen al usuario una paleta de imágenes (formas que pueden usarse para crear dibujos) y un

cálculos de costo-beneficio realistas. Pero si la decisión involucra 20,000 dólares, se tomará al nivel de un auxiliar de vicepresidente que ha pasado semanas tratando de mantenerse despierto el tiempo suficiente como para hacer algo útil dentro de la organización. Formará un comité, desarrollará estándares, hará una encuesta de la industria y escribirá notas a docenas de auxiliares de vicepresidente igualmente somnolientos. Mientras todo este proceso de toma de decisiones sucede, el administrador del proyecto se encoge de hombros, trata de olvidar que alguna vez siquiera entregó la propuesta y regresa a usar sus técnicas de la edad de piedra de construcción de sistemas. Como podrá notar, soy totalmente objetivo y no estoy emocionalmente involucrado de ninguna manera en lo referente a este tema.

"ratón" *("mouse")* con el cual seleccionar las imágenes. Esto le puede ser familiar s ha utilizado programas de gráficas como MacPaint y MacDraw en la Macintosh o PC-Draw o EGA-Paint en las tipo IBM. Sin embargo, las herrramientas automatizadas de análisis son mucho más aún: entienden el tema de los dibujos. Y, como veremos a continuación, tienen muchas características que no existen en programas de dibujo sencillos.

A.2. CARACTERISTICAS IMPORTANTES DE LAS HERRAMIENTAS AUTOMATIZADAS

Es fácil pensar en las herramientas automatizadas para analistas y diseñadores de sistemas como nada más que productos de "dibujo electrónico". Ciertamente es verdad que la capacidad gráfica de estos productos es muy visible y "sexy", pero es sólo una de las características importantes. Esas herramientas deben proporcionar las siguientes características para ser de uso significativo en el desarrollo de un sistema complejo:

- Manejo de gráficas para múltiples tipos de modelos
- Características de revisión de errores para asegurar la precisión de los modelos
- Comparación de modelos diferentes
- Apoyo de ingeniería de software adicional

A.2.1 Apoyo gráfico

Como hemos podido apreciar a lo largo de este libro, los modelos del análisis estructurado se apoyan en diversas formas de información: texto, diccionarios de datos y diagramas gráficos. Los textos y los diccionarios de datos pueden automatizarse usando sistemas de procesamiento de palabras y computadoras grandes convencionales; es el apoyo gráfico el que siempre ha faltado. El analista de sistemas necesita una estación de trabajo que le permita componer, revisar y almacenar los siguientes tipos de diagramas:

- Diagramas de flujo de datos (DFD)
- Diagramas de Estructura (DE)
- Diagramas de flujo (DF)
- Diagramas de Entidad-Relación (DER)
- Diagramas de transición de estados (DTE)

Así, la herramienta automatizada del analista le permitiría crear el DFD que se muestra en la figura A.1(a).

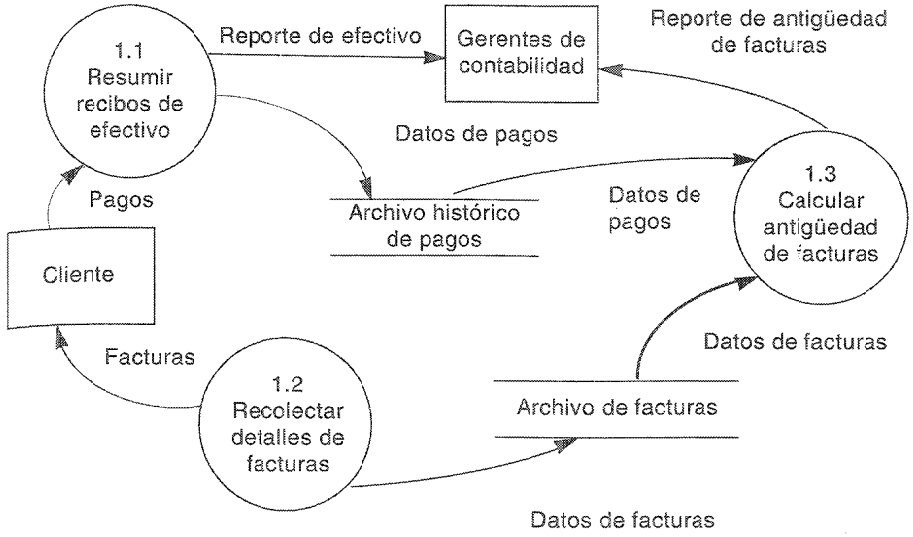


Figura A.1(a): **DFD típico**

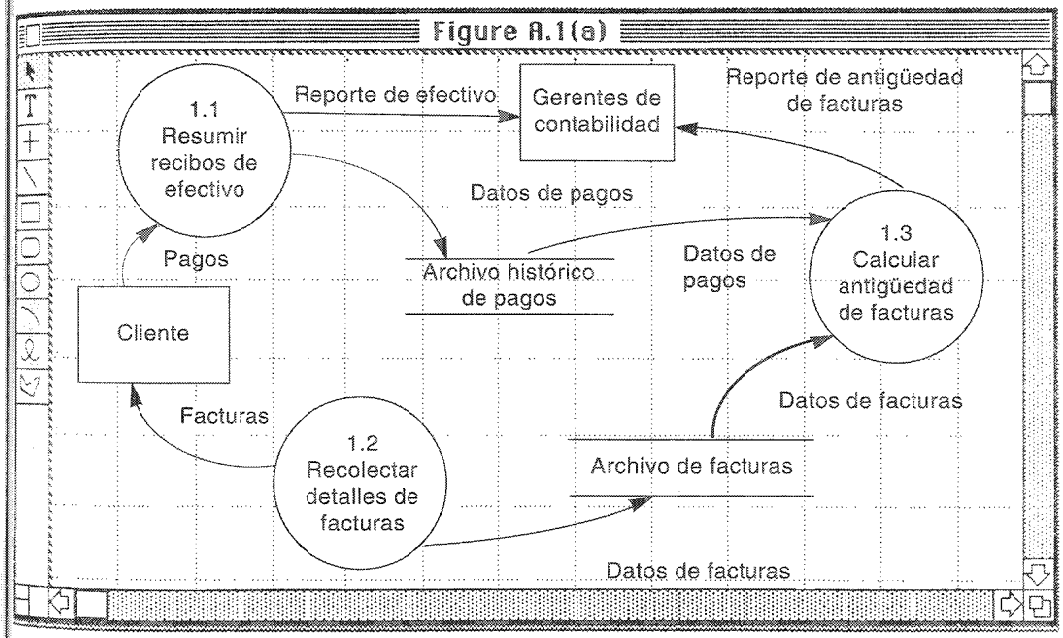


Figura A.1(b): **Pantalla típica de estación de trabajo del analista**

La pantalla de video que el analista ve al crear el diagrama se muestra en la figura A.1(b).

Debo señalar que compuse este diagrama usando un programa sencillo de dibujo llamado MacDraw en la computadora Apple Macintosh que estoy usando para escribir este libro.[4] Me tomó 15 minutos componer el diagrama y otros 30 segundos copiarlo directamente al texto de este capítulo. Pude haber dibujado el diagrama a mano en 3 minutos, y lo hubiera podido adherir a este capítulo usando tijeras y cinta, en alrededor de 30 segundos. El beneficio del apoyo gráfico claramente no radica en el dibujo inicial del diagrama sino en la facilidad de modificación.

En un proyecto típico de desarrollo de sistemas, un diagrama como el de la figura A.1 podría modificarse una docena de veces. De hecho, un analista de Tektronix me dijo que él y un usuario final modificaron un DFD como el de la figura A.1 *más de cien veces* antes de quedar finalmente de acuerdo en que ya se trataba de un modelo preciso de los requerimientos del usuario.[5] Nadie cuerdo consideraría volver a dibujar un diagrama a mano cien veces; sin embargo, hacerle un pequeño cambio a un diagrama en una pantalla de computadora suele tomar sólo alrededor de un minuto. Algunos resultados iniciales del grupo Hartford Insurance Group, que tiene más de 600 estaciones de trabajo instaladas, indican una mejoría de hasta un 40 por ciento en la productividad tan sólo debido al apoyo gráfico automatizado (véase [Crawford, 1985]).

También debo recalcar que los programas de gráficos para usos generales tales como MacDraw o MacPaint (para la Macintosh) o PC Draw o EGAPaint (para la PC IBM) no son realmente apropiados para el ingeniero de software. Para construir modelos formales de ingeniería de software debe decidirse primero qué imágenes o símbolos gráficos se permitirán. Luego deben definirse reglas para dictaminar las propiedades de las imágenes y las conexiones legales entre ellas. La figura A.1, por ejemplo, usa 4 imágenes asociadas con un DFD estándar: un círculo, un rectángulo, una notación para almacén de datos y una línea que muestra el flujo de datos de un lugar a otro. Sin embargo, MacDraw felizmente me hubiera permitido incluir triángulos, hexágonos y cualquier otra representación gráfica en el diagrama. Y Macdraw no sabe que una vez que un flujo de datos se ha conectado con un objeto, ambas cosas deben tratarse de ahí en adelante como un grupo o como un objeto compuesto.[6] En un nivel más simple, tuve dificultades con la creación de burbujas (círculos) del mismo tamaño, y tomaba una eternidad ponerle cabezas a las flechas.

4 La mayoría de los que desarrollan software usan productos CASE ejecutables en computadoras personales IBM, pero los diagramas en este libro no provienen de allí. Para usarlos, tendría que ponerme a determinar como combinar los diagramas con el archivo de texto del libro, que estoy escribiendo con una Macintosh.

5 Obviamente, fue un diagrama mucho más complejo que el de la figura A.1. De hecho, la mayoría de los diagramas de flujo de datos reales lo son; tienen siete u ocho burbujas, tres o cuatro almacenes de datos, una docena o más de flujos de datos, y varios terminadores externos..

A.2.2 Características de revisión de errores

Aunque claramente se necesita de apoyo gráfico, de ninguna manera basta para justificar el gasto de comprar una estación de trabajo computacional de 10,000 dólares estadounidenses. La herramienta automatizada debe examinar el modelo creado por el analista o diseñador para asegurar que sea completo e internamente consistente. La figura A.1, por ejemplo, puede analizarse de la siguiente manera:

- ¿Están conectadas todas las imágenes? ¿Existen almacenes de datos libres o burbujas de proceso que floten alrededor del diagrama, sin salidas ni entradas?
- ¿Tiene cada burbuja por lo menos una salida? Si no, se trata de un agujero negro sospechoso que se traga los datos pero jamás produce salidas.
- ¿Se nombran todos los flujos de datos (las líneas con nombre que unen cuadros y burbujas)? ¿Existen todos los nombres en el diccionario de datos?
- ¿Tienen nombres únicos todos los procesos (burbujas)?

Se puede hacer una revisión similar de errores en los DE, DF, DER y DTE. Y la revisión de errores se puede extender a varios niveles de *modelado*. La figura A.1, por ejemplo, podría ser un subsistema de bajo nivel representado por una sola burbuja (burbuja número 1) en un sistema de contabilidad de mayor nivel, modelado en la figura A.2.

Las herramientas del analista deben asegurar que las entradas y salidas que se muestran en la figura A.1 coincidan con las de la burbuja 1 en la figura A.2. Si no coinciden, el modelo no es consistente, y será un infierno semanas (o meses) más tarde cuando alguien trate de traducirlo a COBOL. El mismo tipo de balanceo se puede aplicar a varios modelos gráficos más que proporcionarán una visión descendente del sistema.

A.2.3 Comparación de modelos diferentes

La característica más importante de las herramientas de trabajo automatizadas del analista/diseñador es la posibilidad de verificar la consistencia de diversos tipos de modelos de un sistema. Existen dos aspectos de esto: la comparación de mode-

6 En realidad, puede decírsele a McDraw que ciertos objetos del dibujo deben agruparse para que se puedan mover en conjunto; pero esto no garantiza que el resultado, tras haberlos movido, se vea como lo desea. Existen paquetes más elaborados, como Design, de Meta Software, y lo son en el sentido de cómo manejan los objetos y los conectores. Pero sin importar lo complejo que sea el paquete, no sirve de gran cosa sin las reglas de revisión de errores que se discuten en la Sección A.1.2.

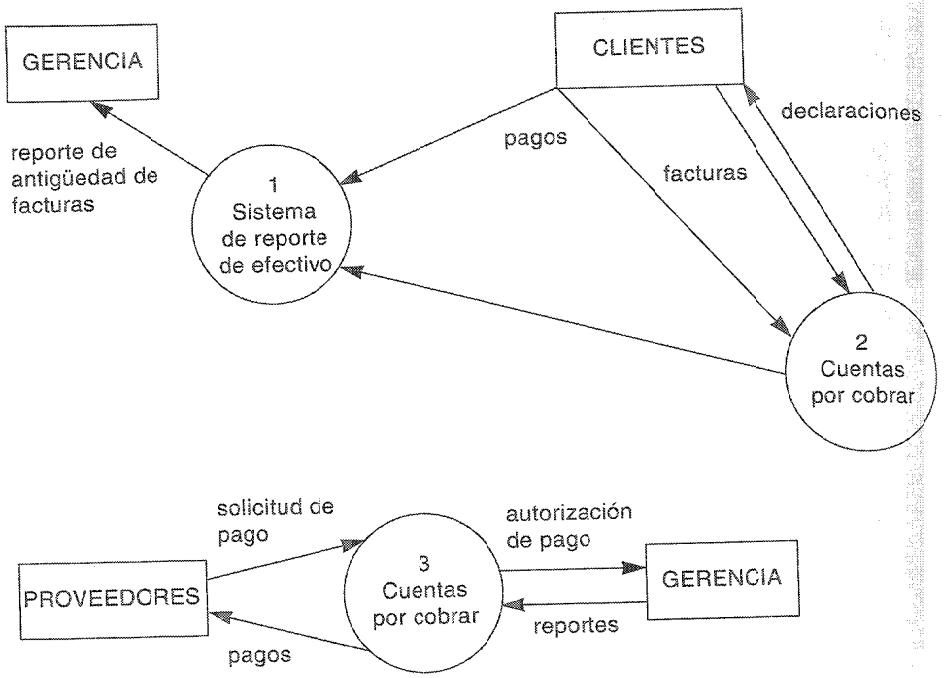


Figura A.2: **DFD de mayor nivel**

los diferentes en una fase de un proyecto, y la comparación de diferentes modelos en diferentes fases de un proyecto.

En la fase de análisis de un proyecto, por ejemplo, el objetivo primario es determinar *qué* desea el usuario del sistema, con poca o ninguna referencia respecto a la tecnología computacional particular que se usará para implantar dichos requerimientos. Para esto se requieren DFD para recalcar la división de estos requerimientos en *funciones* separadas y la interfase entre ellas. Se necesitan DER para recalcar los *objetos* de datos almacenados en el sistema y sus relaciones. Y se requieren DTE para recalcar los *estados* en los que puede encontrarse el sistema y su comportamiento dependiente del tiempo. Además, se usa el diccionario de datos para mantener una definición de todos los datos del sistema y alguna forma de descripción textual para definir la política formal de negocios de cada función de nivel inferior.

La clave es que *todos estos modelos deben ser consistentes entre sí.* Si el DFD se refiere a un dato que no está en el diccionario de datos, algo anda mal; si el diccionario de datos define datos que no aparecen en ningún otro modelo, algo anda mal. Si el DFD muestra almacenes que no están definidos en el DER, hay una inconsistencia; y si el DER muestra objetos que no están definidos en el diccionario de

datos y no se muestran en el DFD, hay una inconsistencia. Como se discutió en el Capítulo 14, toda esta comparación podría hacerse manualmente, pero es una labor tediosa y propensa a errores en el mejor de los casos. Mis varios años de experiencia en ingeniería de software en ambientes típicos de administración de sistemas de información me permiten decir con confianza (desafortunadamente) que no se hará manualmente, a pesar de las exhortaciones de los administradores del proyecto y las mejores intenciones de los técnicos.

Además de la verificación de consistencia entre modelos en una fase de un proyecto, es importante comparar los modelos desarrollados durante *diferentes* fases. Por ejemplo, los modelos desarrollados durante la fase de análisis deben compararse con los desarrollados durante la fase de diseño. La comparación debe demostrar una correspondencia uno-a-uno entre ambos: cada requerimiento descrito en el modelo de análisis (es decir, en los DFD, DER, etc.) debe representarse en alguna parte del modelo de diseño (es decir, los DE, etc.), y cada característica descrita en el modelo de diseño debe corresponder con un requerimiento descrito en alguna parte del modelo de análisis. El problema más común, desde luego, es que se pierde un requerimiento en el modelo de análisis y no aparece en ninguna parte del modelo de diseño.

Esto se da sobre todo cuando el modelo de análisis del sistema lo desarrolla un grupo de personas, y el de diseño (y la subsecuente codificación y prueba) la realiza otro. En el caso más extremo (que a menudo ocurre en proyectos gubernamentales), los dos grupos pueden trabajar para distintas compañías. De cualquier manera, los dos grupos muchas veces representan distintos intereses y perspectivas, y pudiera ser que no se comuniquen bien entre sí en ningún nivel. De aquí que tal vez un requerimiento que el equipo de análisis creía que estaba perfectamente claro no lo pueda entender el equipo de diseño.

A veces el problema es todo lo contrario: el equipo de diseño decide introducir características que el usuario jamás pidió y que jamás se documentaron en el modelo de análisis. Esto puede ocurrir de manera inocente, pero usualmente sucede cuando alguien del equipo de diseño dice: "Aunque no lo hayan pedido, apuesto a que a los usuarios les encantará esto". O el diseñador veterano, algo cínico, dice: "Aunque los usuarios no hayan pedido hoy la característica X, sé por experiencias pasadas que lo querrán la próxima semana. Es más fácil ponerla ahora que esperar a que la pidan". Si esto es o no razonable no viene al caso. Lo importante es que esta discusión se ventile en lugar de permitir que el diseñador tome una decisión unilateral.

De la misma manera, el modelo de diseño debe compararse con el código real. Nuevamente, debe haber una relación de uno-a-uno entre los componentes del código (la implantación misma de los modelos de análisis y diseño) y los componentes del modelo de diseño.

A.3 TECNOLOGIA FUTURA DE LAS HERRAMIENTAS AUTOMATIZADAS

Muchas de las características descritas anteriormente existen en las herramientas de trabajo del analista/diseñador que se encuentran actualmente en el mercado. Algunas de las características pueden implantarse en una manera un tanto primitiva, pero los productos se están mejorando cotidianamente. No obstante, todos los productos deben considerarse como herramientas de primera generación; representan sólo el comienzo de una serie de logros que se obtendrán en los siguientes 5 a 10 años.

Las fechas para las herramientas de desarrollo automatizadas de segunda o tercera generación son un tanto confusas. Parte depende de los recursos de las compañías que las fabrican; parte del desarrollo continuo de tecnología de hardware que permitirá más poder en las estaciones de trabajo personales. Y mucho de ello depende de la cuestión de transferencia de tecnología. Las organizaciones grandes apenas comienzan a experimentar con una o dos estaciones de trabajo a mediados de los años 80, y tomará varios años que se infiltre incluso la tecnología de primera generación a la industria de desarrollo de software.

Sin embargo, tengo la esperanza de que si en 1995 visita una organización grande y profesional de administración de sistemas de información encontrará que todos los programadores (si aún quedan programadores para entonces), los analistas, los usuarios finales y los administradores de proyecto tendrán una estación de trabajo en su escritorio, entre 10 y 100 veces más poderosa que las actuales. Proporcionará las siguientes características:

- Redes para uso del proyecto
- Manejo de metodología de ingeniería de software a la medida
- Control de documentos
- Facilidades para administración de proyectos
- Estadísticas de productividad y métrica de software
- Revisión inicial para evitar complejidad excesiva
- Simulación y pruebas automatizadas
- Pruebas de corrección auxiliadas por computadora
- Generación de código
- Apoyo con inteligencia artificial para código reutilizable
- "Pizarrones" del equipo del proyecto

A.3.1 Redes para uso del proyecto

Las herramientas automatizadas son útiles incluso en proyectos pequeños de una sola persona; también lo es la ingeniería de software. Pero un proyecto pequeño tiene la ventaja de que el trabajo se puede hacer una y otra vez hasta que esté bien, de modo que el uso de modelos y herramientas formales no tiene mucho sentido de urgencia.

Las herramientas automatizadas serán de mayor utilidad en un proyecto de desarrollo de software grande, multimillonario, multianual y multipersonal. En proyectos de este tipo hay varios analistas (a veces una docena o más), varios usuarios finales (a menudo en distintos puntos geográficos), y varios diseñadores y programadores. En este tipo de ambiente, es importante no sólo que el trabajo de cada analista sea internamente consistente, sino también que el trabajo del analista A y el analista B sean consistentes entre sí.

Esto significa que debe existir un nivel de inteligencia por encima del del analista o programador individual. Aunque existen muchas maneras de implantar esto, una de las arquitecturas más atractivas se muestra en la figura A.3. Muchos proveedores de herramientas automatizadas tratan de proporcionar este tipo de redes, típicamente en minicomputadoras Wang o VAX.

La minicomputadora del proyecto debe tener suficiente capacidad de almacenamiento y poder de procesamiento como para realizar revisiones de consistencia en todo el proyecto. Debe también tener suficiente poder para realizar funciones adicionales. Debe permitir a los programadores conectarse directamente a la computadora principal del sistema, para hacer pruebas y otros menesteres normales. Y es el lugar obvio para la inteligencia asociada con muchas de las funciones descritas a continuación.

Añadir una minicomputadora así, con el almacenamiento asociado en disco, canales de comunicación, etc., obviamente incrementa el costo del apoyo automatizado. En dólares estadounidenses de 1985, el costo de una estación independiente apropiadamente configurada era de entre $8,000 y $15,000; con el hardware y software para la minicomputadora del proyecto, el precio fácilmente se doblaría. Vale la pena pagarlo, pero probablemente no se pueda obtener del presupuesto de un solo año para el personal de administración de sistemas de información de una empresa grande; costaría millones de dólares para una organización con algunos cientos de personas encargadas de desarrollo de sistemas. Debe reconocerse como parte de una inversión principal en el esfuerzo por lograr mayor productividad y profesionalismo.

A.3.2 Manejo de metodología de ingeniería de software a la medida

Las herramientas automatizadas normalmente manejan algunas formas específicas de notación y procedimientos de ingeniería de software. Los diagramas de

este apéndice, por ejemplo, además de los anteriores en todo el libro, usan la notación descrita en varios libros de texto de la editorial YOURDON. Pero, sorpresa, varios productos CASE también manejan otras metodologías populares de ingeniería de software, tales como la notación de Gane/Sarson y la de Warnier/Orr. Varias otras herramientas automatizadas de apoyo manejan más de un tipo de metodología de ingeniería de software.

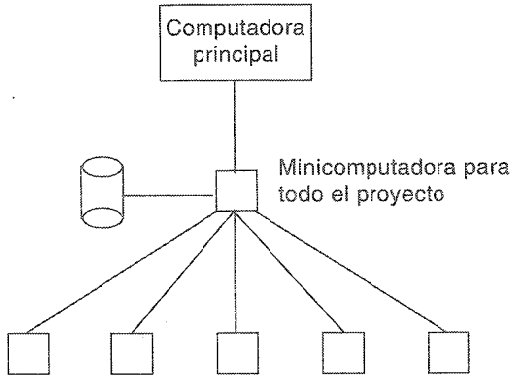


Figura A.3: **Herramienta automatizada del analista/diseñador para todo el proyecto**

Pero existe algo mucho más importante que tan sólo manejar la metodología del proveedor A o B: la herramienta automatizada debe permitir una metodología *a la medida*. Las organizaciones de administración de sistemas de información suelen encontrar que *ninguna* de las metodologías populares de ingeniería de software proporciona una notación adecuada o un conjunto adecuado de reglas para el tipo peculiar de sistema que está desarrollando. Tal vez, por ejemplo, desee usar triángulos para recalcar entradas y salidas de marcianos y de exploradores del capitán Kirk de "Viaje a las Estrellas" (la mayoría no nos tenemos que preocupar por tales entradas y salidas, por lo que jamás se nos ha ocurrido pedir que haya triángulos en nuestra herramienta automatizada). Y tal vez se haya decidido pasar el dictamen de que ningún diagrama de flujo de datos tenga más de 13 burbujas; otra organización puede decidir que ningún sistema tenga más de tres niveles de diagramas de flujo de datos, y así.

Claramente, este tipo de hechura a la medida *debe* permitirse, pues de lo contrario la herramienta gradualmente se dejará de usar cuando los encargados de desarrollar el sistema encuentren que no satisface sus necesidades. Muchas de las

herramientas automatizadas actuales no tienen esta característica; casi todos los productos de segunda generación tendrán esto, o desaparecerán del mercado.

A.3.3 Control de documentos

Como hemos visto, el análisis estructurado se apoya en varios modelos gráficos formales, además de material textual como diccionarios de datos y especificaciones de proceso. Las estaciones de trabajo automatizan el desarrollo y mantenimiento de estos documentos; sin embargo, permiten algo más: el *control* de los mismos. Aunque esto parece fácil, es un concepto radical para muchas organizaciones de administración de sistemas de información. Muchas sólo han empezado recientemente a controlar el código fuente que se produce en la fase de programación del proyecto.

Pero así como resulta desastroso permitir al programador hacer cambios "pequeñitos" a un programa operacional a la media noche, también lo es de igual manera permitirle hacérselos a un DFD o un ERD a la mitad de la fase de análisis de un proyecto, a menos que dicho cambio haya sido autorizado y aprobado.

Para lograr que esto funcione debemos distinguir entre las bibliotecas privadas que cada miembro del proyecto mantiene en su estación de trabajo independiente, y el archivo del proyecto que se mantiene en la minicomputadora del proyecto entero que se muestra en la figura A.3. Es el archivo del proyecto el que se desea controlar. Cuando un analista o diseñador indica que terminó el modelo (o diagrama) y que está listo para incorporarlo al archivo del proyecto, deja de ser el dueño unilateral del material.

A.3.4 Administración del proyecto

El control de documentos es un aspecto de otra característica que la minicomputadora del proyecto puede proporcionar: *la administración del proyecto.* El administrador puede tener su propia estación de trabajo y usar las funciones de la minicomputadora para coordinar y supervisar las actividades de todo el equipo. Con un apoyo de software apropiado, puede saber cuando el analista A termina el DFD con el que estaba trabajando; puede darle a la minicomputadora instrucciones de mandar un DFD al analista B para su revisión y comentarios; luego, puede asignar más trabajo al analista A. Puede usar todo este material para actualizar el calendario y presupuesto del proyecto; y dado que la minicomputadora mantiene su propio registro neutral de cuándo comenzó y cuándo terminó el DFD, es probable que obtenga datos más precisos y sin prejuicios para las actividades de programación del proyecto. El administrador puede usar la función de correo electrónico que seguramente proporciona la arquitectura de la figura A.3 para comunicarse con su personal. Puede ser difícil proporcionar una estimación clara del valor de dicha función, pero la mayoría de los equipos de proyectos encontrarán que ya no pueden vivir sin ella una vez que la tienen.

A.3.5 Estadísticas de productividad y métrica de software

Como se mencionó anteriormente, la minicomputadora del proyecto puede mantener su propio registro de la fecha de comienzo y conclusión de cada parte del trabajo (el desarrollo de un DFD, su revisión, su auditoría, etc.) que realiza el analista, el diseñador o el programador. De esta manera pueden generarse medidas de productividad de manera casi invisible, lo cual con suerte disminuirá el impacto del principio de incertidumbre de Heisenberg.[7] Compare esto con el proyecto típico de desarrollo de software actual, donde los analistas y programadores pasan como una hora a la semana registrando información acerca de cómo pasan su tiempo. Existe una tendencia apenas disfrazada de llenar una forma para hacerla ver como el jefe quiere que se vea (aunque los programadores sean tramposos y charlatanes, no están fuera de contacto con la realidad). Además, si el proceso de registro lleva una hora, entonces afecta al trabajo mismo; esto es algo parecido a lo que un físico llama principio de incertidumbre de Heisenberg.

Casi cualquier otra métrica de software que el equipo del proyecto decida puede llevarse con ayuda de la minicomputadora del proyecto. Así, el equipo puede decidir que es importante saber cuantos DFD se requieren para el sistema, cuántos elementos de datos, cuántas primitivas funcionales o cuántas revisiones se ocuparon antes de que finalmente fuera aceptado por el usuario. Esta información puede ser útil para proyectos futuros y para estimaciones de tamaño y costo del proyecto.

A.3.6 Revisión inicial para evitar complejidad excesiva

Una de las métricas más útiles a la larga es la de la *complejidad*. Existen modelos matemáticos de complejidad de programas que pueden usarse para predecir la dificultad de probar y mantener un programa de computadora.[8] Si los modelos matemáticos se aplican *automáticamente* a cada módulo del programa en el sistema que se está desarrollando, entonces los desarrolladores y el administrador del proyecto tendrán una rápida advertencia sobre las porciones potencialmente peligrosas del sistema y así podrán explorar otros diseños.

A.3.7 Simulación y pruebas auxiliadas por computadora

Como se mencionó anteriormente en este apéndice, existen paquetes de prueba y animadores auxiliados por computadora que ofrecen al programador una representación gráfica de la ejecución de su programa. No existe razón por la cual esta inteligencia no pueda incorporarse a la estación de trabajo remota o a la minicomputadora del proyecto.

7 Aunque el principio de incertidumbre de Heisenberg usualmente se asocia con el campo de la física cuántica, se aplica aquí también: el principio afirma, simplemente, que existen variables que no se pueden medir sin cambiarlas. Si un trabajador tiene que pasar el 10 por ciento de su tiempo midiendo su propio trabajo, entonces su productividad disminuye en por lo menos un 10 por ciento, y el hecho de que es consciente de que se le está midiendo (pues él mismo lo está haciendo) es probable que afecte su comportamiento.

8 Como se discutió en el Capítulo 23, uno de los modelos más populares es la medida ciclomática de complejidad de McCabe. Para más acerca de éste y otros modelos, vea la obra *Controlling Software Projects* de Tom DeMarco (Nueva York: YOURDON Press, 1982).

De hecho, casi todos los programas convencionales de apoyo que se mencionan al principio de este apéndice podrían incorporarse a una herramienta automatizada de trabajo. Al volverse más poderosas las estaciones personales de trabajo, deberá ser posible que se produzca código luego del proceso de modelado (si es que no se puede generar automáticamente), además de compilar y probar. El programador tendrá que mandar el programa a la minicomputadora sólo cuando esté terminado.

A.3.8 Pruebas de corrección auxiliadas por computadora

Como se discutió en el Capítulo 23, el campo de las pruebas de corrección auxiliadas por computadora todavía no se ha desarrollado al grado de que el programador y analista promedio lo puedan usar, pero existe un amplio espectro entre la revisión *informal* de consistencia y las pruebas *formales* de corrección. Con las herramientas de apoyo automatizadas, gradualmente encontraremos que nos alejamos cada vez más de la revisión informal de consistencia para acercarnos a pruebas completas y formales de corrección. Para lograr esto se ocupará un mayor nivel de preparación por parte del programador que el que se puede esperar hoy. Por tanto no se debe esperar esta característica en la mayoría de las estaciones de trabajo orientadas a negocios sino hasta dentro de otros 5 años.

A.3.9 Generación de código

Una meta importante de muchos fabricantes de herramientas es la generación *automática* de código de COBOL o FORTRAN. Así, nadie tendría que ver el código, como actualmente nadie ve los 1 y 0 binarios que son los que de hecho entiende la computadora. En este contexto se estaría tratando con especificaciones computables, desarrolladas por el usuario final y el analista.

Tal vez nunca se logre esto para todos los sistemas, ni se pueda insistir en el nivel necesario de especificación rigurosa y formal de *todos* los usuarios finales. Pero al recalcar cada vez más las actividades de diseño y análisis, fácilmente se puede reducir la programación a una simple tarea de oficinista. Aunque no se automatice completamente, la pueden realizar oficinistas menores, en lugar de graduados de Ciencias de la Computación que cobran $40,000 dólares estadounidenses anuales.

A.3.10 Apoyo con inteligencia artificial para código reutilizable

El concepto de código reusable es aún más llamativo que el concepto de generación de código automatizado. En la gran mayoría de los proyectos (tanto orientados a negocios como a ciencia e ingeniería) la mayor parte del software que se pretende desarrollar ya se ha hecho antes. El sistema nuevo de este año es, de

hecho, casi idéntico al del año pasado, y no muy diferente al del anterior. La mayoría de las primitivas funcionales de nivel inferior ya se han programado antes cientos de veces y existen *gratis* como rutinas de biblioteca proporcionadas como parte del sistema operativo. Lo único que distingue al sistema de este año como único es la combinación particular de estas funciones previamente realizadas, o algún parámetro que se da al programa cuando se ejecuta. Por ejemplo, el sistema de nómina de este año es básicamente el mismo que el del anterior, pero la tasa de impuestos cambió.

Esto sugiere que el analista (y aún más el diseñador del sistema) no debiera ver cada proyecto como un gran experimento de exploración científica, sino más bien como una cacería para ver cuáles módulos ya *existentes* de biblioteca, subrutinas y programas se pueden conectar para satisfacer las necesidades del usuario.

Aquí es donde la inteligencia artificial sería útil. Al correlacionar las características de una función identificada por el analista (por ejemplo, número de entradas, tipo de salidas, especificaciones de transformación o reglas que describen cómo se convierten las entradas en salidas, etc.) con una biblioteca existente de funciones, un sistema experto puede sugerir al diseñador candidatos para implantar el sistema. Y pudiera interactuar con el analista para mostrar que al hacerle algún pequeño cambio a los requerimientos (es decir, al sacrificar un poco los requerimientos originales del usuario) se puede usar una función ya existente de la biblioteca.

A.3.11 Pizarrones del equipo del proyecto

Algunos de los principales investigadores de los Estados Unidos (por ejemplo, en laboratorios de investigación tales como MCC en Austin, Texas) sienten que las verdaderas mejorías a la productividad en los años 90 surgirán de los efectos sinergísticos de un *equipo* de proyecto, más que de un incividuo; esto se debe a que a los grandes sistemas no los construyen individuos sino grupos de grupos de personas (a menudo de compañías distintas). Muchos de los conceptos descritos hasta ahora se ocupan de la mejoría en la productividad de un trabajador individual, pero la inteligencia de la minicomputadora del proyecto puede usarse para proporcionar una visión global conveniente de todo un sistema a medida que crece y toma forma.

Este concepto de un pizarrón de proyecto se está implantando en MCC como parte del proyecto Leonardo; debe proporcionar resultados fascinantes para fines de la década. El grupo de investigación está experimentando con la noción de un pizarrón comunal en la forma de una pantalla del tamaño de una pared. También están investigando la idea de usar el olfato y el oído, además del color, para añadir nuevas dimensiones a los modelos gráficos descritos en este libro. Si tiene éxito, el proyecto Leonardo será la herramienta automatizada de tercera o cuarta generación para la gente que desarrolle software a mediados de los años 90.

A.4 CONCLUSION

Mi inclinación y emoción por este aspecto del desarrollo de software es obvia; no la puedo ocultar. No me disculpo por ello. Verdaderamente siento que representa el único mecanismo para ayudarnos a estar al día y alcanzar a las hordas de programadores en varios países del mundo que acabarían con la mágica industria que se ha creado en los últimos 25 años.

Algunos dirán que esta tecnología es demasiado cara, que ningún programador vale una inversión de 25,000 dólares estadcunidenses. Tal vez así sea, pero dado que el hardware se está volviendo cada vez más barato, los $25,000 de hoy serán $10,000 mañana, y $5,000 el año próximo. Me parece bastante irónico que un país que invierte de $50.000 a $75,000 en equipo para sus campesinos y obreros no quiera invertir unos cuantos miles de dólares para sus trabajadores de la información. Esta renuente aceptación de la necesidad de invertir es el último suspiro de la Revolución Industrial, es decir, el último suspiro de lo que Alvin Toffler llama la "Segunda Ola".

Admito que una profesión de software dominada por estaciones de trabajo automatizadas sí presenta algunas preguntas serias: ¿Vuelve obsoletos a los programadores? ¿Destruirá nuestra creatividad? ¿Podemos darnos el lujo del gasto? ¿Y acaso *garantiza* que mejoraremos dramáticamente nuestra capacidad de producir software de alta calidad más productivamente?

No existe nada mágico en las herramientas de software automatizadas; cualquiera que tenga un poco de sentido común puede responder a estas preguntas. Las herramientas automatizadas seguramente no volverán obsoletos a los programadores a corto plazo ni a los de mantenimiento por 20 años o más. Mientras tanto, ayudará a desenfatizar la programación, lo cual continuará con la tendencia existente desde que se inventó el primer lenguaje de programación a fines de los años 50. No amenaza el trabajo de ningún programador; tenga en mente que hay un atraso de siete años de trabajo en el desarrollo de sistemas en la mayoría de las organizaciones.

¿Acaso una estación de trabajo automatizada destruirá la creatividad de los que desarrollan software? Para nada. Acaso los sistemas CAD/CAM (diseño auxiliado por computadora) destruyen la capacidad de la persona para diseñar un automóvil o un aeroplano estéticamente bello? No. Cien veces, no. Todo lo contrario. La disponibilidad de herramientas de apoyo automatizado ayuda al programador y al analista a concentrarse en la parte *verdaderamente* creativa de su trabajo, y pasar menos tiempo preocupándose por las partes mundanas. Como la herramienta automatizada permite al analista pasar más tiempo inventando más modelos de los requerimientos del usuario, lo vuelve más creativo.

¿Podemos darnos el lujo de pagar el costo de estas estaciones de trabajo? La respuesta es simplemente: no podemos darnos el lujo de no usarlas. Con ellas se

tiene oportunidad de salvar la industria de software de EUA; sin ellas, no hay esperanza. Para aquellos que quieren algo más práctico, tengan en mente que el costo de una estación de trabajo compleja, suponiendo que se incluya el apoyo de la minicomputadora para todo el proyecto, es de alrededor de 25,000 dólares estadounidenses.[9] Esto es más o menos lo mismo que el sueldo anual, en 1987, de un programador típico, con uno o dos años de experiencia. Si se incluyen los costos extra (seguros y pensiones), representa alrededor de seis meses del costo del programador. Dado que fácilmente se puede justificar la amortización del costo de hardware y software asociados durante tres años, su costo es a grandes rasgos igual al 15 por ciento del costo anual del programador. En otras palabras, si la productividad del programador aumenta en un 15 por ciento anual, entonces se paga solo.

Pero, ¿acaso la herramienta automatizada para el desarrollo de software *garantiza* mejorar la productividad en un factor de 10? Cualquiera que pueda plantear esta pregunta en serio, sin duda aún cree en los duendes. ¿Acaso ir a misa todos los domingos garantiza que se irá al cielo? La estupidez, la arrogancia, la pereza y otras debilidades humanas siempre harán posible fallar a pesar de las mejores herramientas y apoyo. No hay manera de evitarlo. Pero si creemos en el poder de los sistemas de información y el apoyo automatizado para los negocios y la sociedad, debemos creer en él para la profesión que construye sistemas para el resto de la raza humana. No siempre debiera resultar cierto que los hijos del zapatero sean los últimos en tener zapatos.

REFERENCIAS

1. Jack Crawford, conferencia en Wang Computer Company, 6 de mayo de 1985.
2. James Martin, *An Information Systems Manifesto*. Englewood Cliffs, N.J.: Prentice-Hall, 1984.
3. Paul Ward y Steve Mellor, *Structured Systems Development for Real-Time Systems,* volúmenes 1-3. Nueva York: YOURDON Press, 1985.
4. *Meilir Page-Jones, The Practical Guide to Structured Systems Design,* 2ª edición, Nueva York: YOURDON Press, 1988.
5. Steve McMenamin y John Palmer, *Essential Systems Analysis*. Nueva York: YOURDON Press, 1984.

9 Se pueden conseguir programas sencillos de dibujo por menos de 100 dólares estadounidenses, e incluso algunos más elaborados a un costo de sólo cientos de dólares. La estación de trabajo más económica, con las características descritas en la sección A.2, costaba alrededor de $895 en 1987; estos precios probablemente disminuyan gradualmente a lo largo de los siguientes años. Incluso una estación de trabajo cara sólo costaba alrededor de $8,000 en 1987.

6. Paul Ward, *Systems Development Without Pain*. Nueva York: YOURDON Press, 1984.
7. Brian Dickinson, *Developing Structured Systems*. Nueva York: YOURDON Press, 1980.
8. Edward Yourdon, *Managing the Systems Life Cycle,* 2ª edición, Nueva York: YOURDON Press, 1988.
9. Edward Yourdon y Larry Constantine, *Structured Design,* Englewood Cliffs, N.J.: Prentice-Hall, 1979.
10. David King, Current *Practices in Software Engineering*. Nueva York: YOURDON Press, 1983.
11. Tom DeMarco, *Structured Analysis and Systems Specification*. Englewood Cliffs, N.J.:Prentice-Hall, 1979.
12. Tom DeMarco, *Concise Notes on Software Engineering*. Nueva York: YOURDON Press, 1979.
13. Chris Gane y Trish Sarson, *Structured Systems Analysis and Design*. Nueva York: Improved Systems Technologies, 1977.
14. Ken Orr, *Structured Systems Development*. Nueva York: YOURDON Press, 1977.

APENDICE

B REGLAS DE ESTIMACION

B.1 INTRODUCCION

Como analista de sistemas, probablemente tendrá que producir varias *estimaciones* para el proyecto. De hecho, tal vez sea el único responsable de producirlas en ocasiones; en otros casos, el administrador del proyecto puede pedirle que le desarrolle las estimaciones.

¿Qué tipo de cosas necesitan estimarse en un proyecto de desarrollo de sistemas? Esto varía de un proyecto a otro pero típicamente lo principal que se requiere estimar es:

- *Recursos humanos.* ¿Cuántos programadores, analistas, diseñadores de bases de datos, expertos en telecomunicaciones, representantes de los usuarios y otros tipos de personas se necesitan para el proyecto?
- *Tiempo.* ¿Cuánto tardará el proyecto? ¿Cuánto tiempo se puede esperar invertir en cada fase típica del proyecto (por ejemplo, la fase de análisis, la de diseño, la de programación, la de prueba, etc.)?
- *Programación de personal.* Además de saber cuántas personas requiere el proyecto, necesitamos saber cuándo se requerirán. Si el proyecto requiere diez programadores, ¿se necesitarán todos al mismo tiempo?
- *Presupuesto.* ¿Cuánto costará desarrollar el sistema? El costo principal será probablemente el de los sueldos del personal de desarrollo, y esto usualmente se puede calcular directamente una vez que se conocen los recursos humanos y la programación del personal. Desde luego, hay otros costos asociados con un proyecto; se dan con detalle en el apéndice C.

Tenga en mente que generalmente tendrá que hacer las estimaciones más de una vez. Suele hacerse un conjunto de estimaciones en las primeras etapas de un

proyecto (por ejemplo, durante el estudio de factibilidad), pero pueden requerirse muchas veces, a medida que los usuarios y la administración exploran las diferentes posibilidades de combinaciones. Un ejemplo obvio de esto es el posible sacrificio de funcionalidad a cambio de tiempo y viceversa (por ejemplo, el administrador del proyecto puede decir al usuario: "Estoy prácticamente seguro de que podemos entregarle el sistema para el 1º de enero si no metemos las funciones X, Y y Z".); otro ejemplo es el de la relación inversa entre personas y tiempo (por ejemplo, el usuario puede decirle al administrador del proyecto, "Si tuviera tres programadores más, ¿podría terminar el proyecto a tiempo?"). Podría tomar varias iteraciones que el equipo del proyecto, la administración y la comunidad usuaria lleguen a un acuerdo aceptable.[1]

B.2 ESTIMACION DE LOS PELIGROS

Tenemos una gran cantidad de experiencia común en el área de la estimación; piense tan sólo en todas las situaciones en donde surgen preguntas del tipo de: "Cuánto tiempo crees que tardemos en manejar hasta la casa de tu tía?" Todos estamos familiarizados intuitivamente con el concepto de una estimación optimista y una pesimista. Pero la estimación de un proyecto de desarrollo de sistemas es algo diferente, pues 1) el enfoque del trabajo es más amplio y un tanto más complejo y, 2) las consecuencias de un error en las estimaciones usualmente son más severas que las que surgen de llegar media hora tarde a la casa de la tía.[2] Existen varios problemas comunes de los que debe estar consciente antes de empezar a calcular presupuestos, tiempos y requerimientos de recursos:

- La diferencia entre estimar y negociar
- La gran variedad de capacidad de los técnicos
- El peligro de estimar su propio trabajo
- La falta de una base de datos de estimación
- La insistencia de la administración en tener estimaciones detalladas y prematuras.
- La dificultad, común a toda la industria, de medir la unidad de trabajo.
- Estimaciones basadas en supuestos de tiempo extra sin sueldo

1 Tenga en cuenta también que las estimaciones ciertamente se tendrán que revisar durante el proyecto al ir cambiando las circunstancias. Los factores externos (condiciones de negocios, nuevos competidores, fusiones, etc.) pueden ocasionar que el usuario cambie de opinión respecto a la funcionalidad requerida, los gastos o la fecha de entrega estipulada. Y factores internos (tales como cambios de personal, dificultades inesperadas de implantación, etc.) también pueden ocasionar el cambio de presupuestos y tiempos, usualmente de manera dramática.

2 Algunas tías pudieran estar violentamente en desacuerdo con esto.

Dependiendo de su puesto dentro del proyecto y de su influencia con la administración y los usuarios, puede tener la posibilidad de prevenir que algunos de estos problemas se vuelvan más serios. Pero incluso si es un analista muy abajo en el escalafón, debe estar consciente de los problemas de estimación, pues a fin de cuentas pueden determinar el éxito o fracaso de su proyecto. Discutiremos cada uno de estos problemas con más detalle a continuación.

B.2.1 La diferencia entre estimar y negociar

A menudo existe una buena cantidad de negociación al principio de un proyecto (y muchas veces a lo largo de toda la fase de su desarrollo). Esto es normal pues la comunidad usuaria entiende poco acerca de la cantidad de trabajo que un sistema de información complejo involucra, y por ello pedirán "que les bajen la luna", es decir, funcionalidad enorme en una cantidad absurdamente pequeña de tiempo, y a cambio de muy poco dinero. Mientras tanto, el administrador se enfrenta al problema de personal y presupuesto limitado; por tanto, necesita trabajar con los usuarios para ayudarles a ver las posibles combinaciones.

Pero este proceso de negociación, tan necesario y tan común en proyectos de desarrollo de sistemas, no debe confundirse con la *estimación*. Lo que debe evitarse son este tipo de conversaciones entre usuario y analista (o quien haga la estimación):

Usuario: "Bien, ¿cuánto va tardar la construcción del sistema XYZ?"

Analista: "Bueno, parece que lo terminaremos para el 1º de abril."

Usuario: "Eso no basta. Lo ocupamos para el 1º de enero".

sin disposición de discutir otras combinaciones de funcionalidad o de recursos.[3] Esto a veces va seguido de llamados del usuario sobre el sentido del deber, patriotismo, etc., del equipo del proyecto, lo cual se discute como problema aparte en la sección B.1.8 .

En algunos casos, esta situación simplemente refleja una falta de comprensión por parte del usuario, que se puede corregir explicándole cuidadosamente las actividades involucradas en el desarrollo de un sistema. En otros casos, sin embargo, refleja el enfoque global del usuario, su "paradigma" de negocios, para tratar con personas y organizaciones que le proporcionan productos y servicios. Para el usuario típico, como se verá, la organización interna de procesamiento de datos no difiere mucho del vendedor externo; la negociación, con el intento de recortar el precio o el tiempo de entrega en algunos meses, es algo muy natural. Y es algo razonable (desde este punto de vista) si la persona u organización que proporciona el servicio tiene un margen de ganancia que se puede recortar mediante buenas negociaciones.

3 Existe otro compromiso, pero casi nadie habla de él explícitamente: la *calidad*. Muchos administradores de proyecto tratan de lograr milagros entregando el material con la funcionalidad requerida dentro del límite de tiempo impuesto por el usuario y con los menos que óptimos recursos existentes; pero el resultado inevitable será un sistema con más errores, y que será menos mantenible de lo que hubiera sido el caso de otra manera.

Incluso en el caso de una organización de procesamiento de datos interna, la negociación (sin aceptar sacrificios en funcionalidad, recursos, presupuesto o tiempo) puede ser razonable si las estimaciones incluyen un margen de error (conocido también como factor de contingencia o de amortiguamiento) que el usuario considere irrazonablemente grande. Pero si proporcionó estimaciones cuidadosas y realistas, y el usuario le regatea hasta reducirle el personal, el presupuesto y el tiempo, entonces su proyecto entra al área de política pesada, para la cual las técnicas y herramientas que se discuten en este libro probablemente no le ayuden mucho. Puede ser que haya llegado la hora de actualizar su currículum.

B.2.2 La gran variedad de capacidad de los técnicos

Es común estimar el trabajo a realizarse basándose en la capacidad promedio (por ejemplo, el programador o analista promedio, capaz de escribir 15 renglones de código o de dibujar cuatro burbujas de un DFD en un día promedio). Es importante tener en mente que estudios realizados en el transcurso de los últimos 20 años han mostrado una gran diferencia entre profesionales mediocres y aquellos con talento en este campo.[4] Si su proyecto tiene un grupo de superprogramadores, puede sobreestimar drásticamente la cantidad de tiempo y dinero necesarios para terminarlo. De mayor preocupación aún es el hecho de que un equipo de gente mediocre puede hacer que el proyecto salga del tiempo y del presupuesto programados por un gran margen de diferencia.

Un problema importante en esta área es que el rendimiento mismo de la gente con experiencia no necesariamente se correlaciona del todo con la mayoría de los exámenes de aptitud. Por ello, debe basar sus estimaciones en la reputación de cada persona, o su experiencia laboral previa, o bien debe suponer simplemente que todos son de tipo promedio. Dado que la mayoría de las organizaciones no guardan registros cuidadosos y detallados sobre la productividad de cada miembro de un equipo de desarrollo de sistemas, será muy difícil obtener evidencia confiable. Lo que le queda es hacer el mejor juicio posible y luego modificar las estimaciones, como vaya siendo necesario, durante el transcurso del proyecto.

4 Uno de los primeros indicadores de esto fue un trabajo titulado: "Exploratory Experimental Studies Concerning Online and Offline Programming Performance", de H. Sackman, W.J. Erickson y E. E. Grant, en el número de enero de 1968 de *Communications of the ACM*. Su estudio mostró una variación de 26:1 entre los mejores y peores programadores, a los cuales se dio la misma tarea de programación. Esta variación entre buenos y malos programadores se ha verificado muchas veces durante los últimos 20 años.

B.2.3 El peligro de estimar su propio trabajo

Uno de los peores errores que puede cometer es estimar su propio trabajo; es casi tan malo como usar la estimación de una sola persona, puesto que el juicio de esa persona puede verse afectado por varios factores.

Si hace la estimación de su propio trabajo, es probable que caiga presa de uno o más de los siguientes mitos:

- *"Soy mejor que la mayoría de los que me rodean aquí, y estoy seguro de que puedo terminar el proyecto mucho más rápido".* Es muy común sobreestimar la capacidad propia. Cuando estima la de otra persona, tiende a ser conservador; cuando estima la propia, tiende a ser optimista.

- *"Sé que el jefe ocupa esto rapidísimo, y de verdad lo quiero ayudar".* En la mayoría de los casos, este es un sentimiento altruista: es muy natural querer ayudar al éxito de su jefe. Pero puede nublar su juicio acerca del tiempo requerido para terminar el proyecto. En el peor de los casos, se hacen estimaciones optimistas en un intento de impresionar a su jefe (tenga en mente que él está haciendo lo mismo con su jefe, y el de él al suyo, etc.) para lograr un aumento o ascenso. Si sabe lo que hace, y es capaz de aceptar el riesgo, perfecto;[5] pero tenga en cuenta que está jugando con fuego.

- *"Estoy dispuesto a trabajar duro para concluir esto a tiempo".* La disposición de trabajar horas extras es admirable pero también es peligrosa, como se sugirió anteriormente. Además, tenga en mente que el compromiso de dedicar muchas horas a menudo se contrae en un momento de gran emoción al principio del proyecto; seis meses más tarde, pudiera no parecer tan buena idea.

- *"He trabajado antes en sistemas como éste; esto es pan comido".* Bueno, tal vez, si de hecho ha trabajado en un proyecto exactamente igual a éste o muy parecido. Sin embargo, al inicio del proyecto existe una tendencia a ver similitudes superficiales con proyectos anteriores y suponer de manera optimista que se podrá terminar aún más rápido. Es probable que encuentre que el nuevo proyecto en realidad es muy diferente cuando vea los detalles; y es probable que olvide todos los problemas que tuvo con el proyecto anterior.

5 Dios mira encima de mi hombro al estar escribiendo esto, y dice, "No, no está perfecto". Tal vez esté dispuesto a tomar el riesgo de no entregar el proyecto en la fecha y presupuesto optimistas en que los está prometiendo, pero fallar probablemente ponga en peligro mucho más que su carrera. No es ético, profesional, ni intelectualmente honesto hacer estimaciones optimistamente irreales cuando su jefe, los usuarios y toda la organización pudieran sufrir pérdidas considerables porque no pudo entregar lo que prometió.

Por estos motivos, es muy importante que las estimaciones las haga otra persona y no el responsable del trabajo. También es muy deseable que si no se tiene una base de datos de estimación (que se discute en la sección B.1.4) o un paquete de estimación computarizado (que se discute en la sección B.4), se obtengan estimaciones de más de una persona. Por lo menos obtenga las de tres personas; esto dará una estimación del peor y del mejor de los casos, además de una intermedia.

B.2.4 La falta de una base de datos de estimación

Cuando uno se enfrenta con un nuevo proyecto le gustaría poder usar estadísticas de una docena de proyectos similares para producir estimaciones precisas. Algunas compañías consultoras y casas de software pueden hacerlo: cuando se les pide estimar el tiempo y dinero necesarios para, digamos, un sistema de ingreso de pedidos, pueden decir: "Bueno, esto es casi igual a los últimos 137 sistemas de ingreso de pedidos que hemos construido, así que debe tomar X meses-persona, Y dólares y Z personas".

Incluso dentro de su propia organización, es posible que se hayan desarrollado 137 sistemas de ingreso de pedidos durante las últimas décadas. Pero esto no significa necesariamente que será fácil estimar el presupuesto o el tiempo necesarios para el 138; si no se guardaron registros precisos, en todo lo que se puede basar es en chismes y rumores. En una organización típica de proceso de datos, que actúa como organización de servicio interno, sin tener que preocuparse sobre cifras de pérdida/ganancia o consideraciones de flujo de efectivo, no existe un buen incentivo para guardar registros cuidadosos como éstos.

Algunas organizaciones grandes de proceso de datos están comenzando a cambiar su actitud y desarrollan bases de datos que pueden usarse para generar estimaciones más precisas para proyectos futuros. Algunas compañías consultoras que se han especializado en esta área han desarrollado bases de datos de literalmente *miles* de proyectos, que se usan para proporcionar parámetros en los modelos de estimación computarizados que se discuten en la sección B.4.

Mientras tanto, existe una buena probabilidad de que se enfrentará con una estimación única en su género. Definitivamente debería buscar otros proyectos similares en su organización; pero esté consciente de que puede estar en una situación análoga a la del arquitecto a quien se le preguntó cuánto tiempo tardaría en construir una casa subterránea en un pantano.

B.2.5 La insistencia de la administración en estimaciones prematuras detalladas

Como regla general, es casi imposible producir estimaciones detalladas de costos, tiempos y requerimientos para un proyecto hasta que se hayan realizado una gran cantidad de análisis y diseños detallados de sistemas. Después de todo, ¿cómo le puede decir al usuario cuánto costará un sistema si no sabe lo que se desea?

No obstante, existe una gran presión, tanto por parte de los usuarios como por parte de la administración, para producir una estimación que ambas partes consideren precisa y detallada, y además en una etapa muy temprana del proyecto. ¿Por qué? Simplemente porque necesitan tomar una decisión de proceder o no a la inversión de tiempo, dinero y recursos humanos para construir el sistema.

Esta exigencia de una pronta estimación es fácilmente entendible; lo único que no es realista es suponer que una estimación temprana puede ser detallada y precisa. Resulta más apropiado dar a la administración una serie de estimaciones a lo largo de todo el proyecto, cada cual más precisa y detallada que la anterior. Así, si el equipo está desarrollando un sistema para una aplicación con la que están bastante familiarizados pueden proporcionar, por ejemplo, la siguiente serie de estimaciones:

- Al final de la encuesta o estudio de factibilidad: una estimación que puede variar en más/menos un 50 por ciento. Es decir, el equipo del proyecto puede decir a la administración que el sistema tardará un año y costará $200,000, más/menos un 50 por ciento. La administración debiera darse cuenta con esto de que en realidad puede tardarse 18 meses y costar hasta 300,000 dólares estadounidenses.
- Al final de la etapa de análisis: una estimación que puede variar en más/menos un 25 por ciento.
- Al final de la fase de diseño: una estimación revisada que puede variar en más/menos un 10 por ciento.
- Al final de la fase de programación (cuando aún falta hacer las pruebas): una estimación final que no deberá variar en más/menos un 5 por ciento.

B.2.6 La dificultad, común a toda la industria, de medir la unidad de trabajo.

Muchas industrias tienen formas estándar de medir la cantidad de trabajo de un proyecto. Quien construye una casa, por ejemplo, puede medir la unidad de trabajo en términos del número de ladrillos a ser colocados, o del número de habitaciones a construir. Pero en el área del desarrollo de sistemas no existe acuerdo sobre la forma de medir la unidad de trabajo a realizar.

El método más común es medir el número de instrucciones, o líneas de código, que se deberán escribir. Así, en algunos proyectos, probablemente se haga referencia a KLOC, que en inglés significa Kilo-líneas de código. Pero existen muchos problemas con esto:

- ¿Cuentan como línea de código los comentarios en un programa de computadora?

- ¿Contamos sólo el código que se le entrega al usuario o también el que se escribe para realizar pruebas, programas de utilerías y otras actividades de apoyo durante el proyecto? (Y, en mayor escala, ¿contamos el código asociado con proyectos cancelados en un intento de medir la productividad de la empresa?)
- ¿Qué tal si el programador ha escrito más de una instrucción en la misma línea de un listado de programa? ¿Y qué con las instrucciones complejas que requieren más de una línea?
- Y, lo más importante, ¿cómo tratamos el hecho de que algunos programadores usen más líneas de código que otros para lograr la misma función? Como se vio en la sección B.1.2, esto puede variar bastante.

Como señala Capers Jones en su obra *Programming Productivity* [Jones, 1985]), los resultados de productividad reportados se pueden distorsionar en dos órdenes de magnitud usando diferentes formas de medir la unidad de trabajo; tal vez ésta sea la razón por la cual algunos programadores argumenten ser 10 o 100 veces más productivos que sus colegas. Debido a estos problemas, algunas organizaciones están empezando a usar "puntos de función" como unidades de trabajo; esto corresponde a grandes rasgos a las burbujas atómicas de nivel inferior en un DFD.[6]

B.2.7 Estimaciones basadas en suposiciones de tiempo extra no pagado

Como se mencionó anteriormente, los usuarios y los administradores de proyecto pueden reaccionar a los conflictos de tiempo sugiriendo, implícita o explícitamente, que el equipo del proyecto invierta horas extra los fines de semana, se quede sin vacaciones o por lo menos las posponga. Esto usualmente se ve acompañado de apelaciones a su lealtad, profesionalismo, dedicación, devoción, orgullo, honor y patriotismo.

Le dejo la decisión respecto a si la voluntad de trabajar o no horas extras sea cuestión de patriotismo. En algunas organizaciones este pudiera ser el caso: todos los proyectos podrían estar organizados de tal forma que sólo tengan éxito si el equipo invierte 80 horas semanales de trabajo. Y algunos proyectos (por ejemplo, el proyecto Apollo de la NASA, que llevó al hombre a la luna por primera vez en 1969) pueden ser tan emocionantes que todo mundo estará más que contento de apuntarse para las horas extras requeridas. Y no es algo fuera de lo común encontrar que el

---

6 El término "punto de función" lo introdujo A.J. Albrecht para describir esto; véase "Measuring Application Development Productivity", *Proceedings of the Joint SHARE/GUIDE Application Development Symposium* (Chicago: GUIDE International Corp., 1979). Tom DeMarco utiliza el término "explosión de función" en una manera muy similar; para mayores detalles vea su libro, *Controlling Software Projects* (Nueva York: YOURDON Press, 1982). También véase la obra de Capers Jones, *Programming Productivity* (Nueva York: Mc Graw- Hill, 1986) para una discusión completa respecto a las dificultades de medir la productividad y los muchos factores que la afectan.

proyecto que parecía estar bajo control se atrasa durante el último mes, requiriendo así unas cuantas semanas de trabajar hasta tarde y los fines de semana.

Pero recuerde que trabajar es como correr: puede correr a toda velocidad unos 100 metros, pero no a lo largo de todo un maratón de 42 kilómetros. De manera similar, puede trabajar días de 14 horas durante algunas semanas, pero no es realista, en la mayoría de los casos, suponer que puede trabajar días de 14 horas durante un proyecto de tres años. Las personas con pareja, hijos, u otros intereses, simplemente se rehusarán a continuar trabajando así después de unos meses; de ser necesario, renunciarán y buscarán otro trabajo. La gente joven y soltera podría estar más dispuesta a dedicar al proyecto todo el tiempo que está despierta (al igual que sus sueños), sobre todo si sienten que les ayudará a avanzar en su carrera o en su conocimiento de la profesión.

Incluso si los miembros del personal están dispuestos a trabajar días de 14 horas, no hay garantía de que sean eficientes en su trabajo. Esto sucede sobre todo si el trabajo en tiempo extra continúa por varios meses: las horas extras a menudo se vuelven no productivas, y usualmente se cometen más errores cuando la gente trabaja en un estado de ánimo exhausto y presionado.

B.3 REGLAS PARA LA ESTIMACION

Existen cuatro reglas importantes cuando se desarrollan estimaciones de la cantidad de trabajo a realizarse en un proyecto de desarrollo de sistemas:

1. Haga las unidades de estimación lo más pequeñas posible.
2. Hágalas lo más independientes posible.
3. Tenga en cuenta el factor de comunicación.
4. Distinga entre trabajo nuevo y prestado.

Esto se discute a continuación.

B.3.1 Haga las unidades de estimación lo más pequeñas posible

Esto debería ser una sugerencia obvia, pero no se le hace caso tan a menudo como pudiera pensarse; también tiene algunas desventajas, como veremos. Pero, en general, es mucho mejor estimar el presupuesto y el tiempo requerido para una unidad de trabajo de una semana que para un "milenio-hombre".[7] Los proyectos grandes tienen complejidades grandes; si intenta estimar la cantidad de trabajo, es

7 Un "milenio-hombre" son mil años-hombre de trabajo. Uso la palabra "hombre" deliberadamente, porque estoy convencido de que las mujeres son demasiado inteligentes como para emplear estas enormes unidades. Este término lo sugirió originalmente uno de los clientes de mi compañía, que es una gran compañía eléctrica de California.

casi seguro que cometerá errores importantes. Es mejor basar sus estimaciones en cantidades pequeñas de trabajo.

Esto implica, desde luego, que el proyecto global se ha reducido a unidades pequeñas de trabajo, lo cual normalmente sucederá como resultado del análisis, el diseño y la programación estructurados; desafortunadamente, como se discutió en la sección B.1.5, puede requerirse una estimación detallada del presupuesto y los requerimientos de recursos *antes* de hacer esto. No existe una solución mágica para el problema; lo que puede hacer es tratar de convencer a los administradores y usuarios de que una estimación precisa y detallada requiere un esfuerzo inicial para determinar las unicades de trabajo a realizarse.

Pero, ¿qué tan pequeñas deben ser las unidades de trabajo? Algunas organizaciones las miden por mes; sin embargo, esto parece demasiado grande. En el lapso de un mes los proyectos pueden salir seriamente de control. Tal vez sea más razonable medirlo en unidades de una semana; como me dijo una vez un administrador de proyecto veterano: "Nunca se logra nada útil en menos de una semana." Sin embargo, tal vez la unidad más común de trabajo sea un día; esto se ajusta perfectamente a la manera en que organizamos nuestra vida de trabajo. Algunas organizaciones de hecho lo miden en horas; aunque sí existen muchas actividades que toman una hora o menos (por ejemplo, definir un dato en el diccionario de datos), parece una unidad demasiado microscópica como para trabajar con ella.

B.3.2 Hágalas tan independientes como sea posible

Un problema que afecta a muchos intentos de estimar es la interacción, o acoplamiento, entre una parte del trabajo y otra. Si un sistema se divide en porciones con muchas interacciones, entonces la cantidad total de trabajo necesario para desarrollar el sistema será mucho mayor que la suma lineal de la cantidad necesaria para cada porción. Si se hace algún cambio a la porción 13, por ejemplo, el cambio puede causar problemas en la porción 14, y un cambio a la 14 podría resultar en cambios a la 15, etc. El efecto de ola ha causado el caos en muchos proyectos.

La solución es dividir el sistema en porciones pequeñas e independientes que se acoplan de manera holgada con otras. Esto requiere de una labor cuidadosa; se discutió en el Capítulo 20 como la principal justificación de la agrupación de burbujas de bajo nivel en agregados de nivel superior dentro del modelo preliminar de comportamiento. La noción de independencia modular también es importante durante la fase de diseño del proyecto; se analizó en el Capítulo 22.

B.3.3 Tenga en cuenta el factor de comunicación

Incluso en un proyecto donde todos los módulos son independientes entre sí, las personas tienen que comunicarse. Si un individuo va a realizar el proyecto, entonces sólo se requiere comunicación con el usuario (y tal vez algunas discusiones con la administración). Pero un proyecto típico tiene muchos analistas, diseñadores,

especialistas en bases de datos y programadores; o peor aún, algunos de ellos pueden trabajar en distintas compañías o incluso hablar diferentes idiomas.

Por tanto, las estimaciones deben incluir tiempo para la comunicación entre todo el personal del proyecto. Esta comunicación tomará la forma de reuniones, memoranda, conversaciones telefónicas, etc. Tenga también en cuenta que la cantidad de comunicación aumenta drásticamente al aumentar el tamaño del equipo: el número de vías de comunicación entre miembros del equipo aumenta como el *factorial* del número de individuos.

B.3.4 Distinga entre trabajo nuevo y prestado

Si el equipo tiene suerte, podrá usar el trabajo hecho en proyectos anteriores. Muchas veces esto se logra mediante módulos en una biblioteca de software común.[8] Sin embargo, no debe suponer que los módulos reusables vienen gratuitamente; tomará tiempo 1) hallarlos, 2) investigar si realizan la función deseada y, 3) aprender lo suficiente sobre ellos para poder entender cómo usarlos. Es más apropiado estimar que los módulos prestados tomarán una fracción (tal vez un 25 por ciento, o tan poco como el 10 por ciento) del trabajo necesario para desarrollarlos desde un principio.

B.4 FORMULAS PARA LA ESTIMACION

Durante los últimos 20 años la industria de desarrollo de sistemas ha invertido una cantidad enorme de tiempo y esfuerzo en el desarrollo de modelos, o fórmulas, para predecir el tiempo, recursos y costos de un sistema. Algunos de estos modelos se usan mucho actualmente; tal vez el más conocido sea el modelo COCOMO, desarrollado por Barry Boehm en TRW.[9] Pero como señala Tom DeMarco en *Controllirg Software Projects,*

> *No hay modelos de costos transportables.* Si espera que alguien más desarrolle un conjunto de fórmulas que pueda usar para prever costos en su negocio, probablemente espere para siempre. Buena parte de nuestra industria concluyó, al percatarse de esto, que el modelado de costos era irrelevante. Creo que esa fue una conclusión errónea. Si se pueden usar modelos de costos desarrollados localmente para mejorar la precisión del proceso de predicción, y si la mejoría vale el gasto de desarrollarlos, entonces el concepto es viable.

8 Pero también puede ser posible reutilizar porciones de la parte de diseño de un modelo de requerimientos del usuario, o incluso porciones de un estudio de factibilidad. Antes esto normalmente no se hacía porque el modelo de diseño, los modelos de análisis y los estudios de factibilidad no se documentaban bien y jamás se mantenían. Ahora, con la proliferación de los productos de estación de trabajo para análisis, del tipo que se discute en el apéndice A, esto se está volviendo más práctico.

9 Para una discusión detallada acerca de este modelo véase la obra de Barry Boehm, *Software Engineering Economics* (Englewood Cliffs. N.J.: Prentice-Hall, 1981).

Sin embargo, es interesante ver algunas de las fórmulas que se utilizan en otras organizaciones; por lo menos le darán un punto de partida para desarrollar las propias. Algunas manejan hasta 40 factores o parámetros; pero, como verá, algunas usan sólo uno.

B.4.1 Fórmulas para estimar el tiempo de trabajo

Tres fórmulas comunes para estimar el esfuerzo (descrito en horas-persona) se basan en líneas de código. Walston y Felix desarrollaron un modelo en IBM (véase [Walston y Felix, 1977]), basándose en observaciones de unos sesenta proyectos, que se expresa de la siguiente manera:

$$E = 5.2 * L^{0.91}$$

donde E se midió en meses-persona, y L en miles de líneas de código.

De manera similar, Bailey y Basili desarrollaron una fórmula basada en 19 proyectos; se expresa como

$$E = 3.4 + 0.72 * DL^{1.17}$$ más o menos un 25 por ciento.

donde nuevamente se midió el esfuerzo en meses-persona, y DL en miles de líneas de código entregado; véase [Bailey y Basili, 1983].

Finalmente, el modelo COCOMO de Barry Boehm tiene una fórmula de esfuerzo para tres tipos distintos de sistemas: sistemas orgánicos, semi-separados, e integrados:

$$E = 2.4 * KDSI^{1.05}$$ (sistemas orgánicos)

$$E = 3.0 * KDSI^{1.12}$$ (sistemas semi-separados)

$$E = 3.6 * KDSI^{1.20}$$ (sistemas integrados)

donde KDSI representa "kilo instrucciones fuente entregadas"; véase [Boehm, 1981] para más detalles.

B.4.2 Fórmulas para estimar tiempo

Una vez desarrollada la estimación de cantidad de trabajo a realizar, pudiera pensarse que es fácil estimar la cantidad de tiempo necesario para el proyecto. Después de todo, si tiene un proyecto estimado en 10 meses-persona de trabajo, y hay 5 personas disponibles, entonces terminar el proyecto debiera tomar dos meses. ¿Pero qué tal si sólo hay disponibles 2? ¿Tomará entonces 5 meses?

En general, lo que interesa aquí son las relaciones entre tiempo y personas. Muchos años de experiencia dolorosa nos han enseñado que esto no es sencillo: doblar el número de personas que trabajan en el proyecto no necesariamente reduce a la mitad la duración. De hecho, Fred Brooks, el arquitecto del sistema operativo

OS/360 original, acuñó la frase "Añadir más gente a un proyecto de software retrasado, sólo lo retrasa más". Existen dos razones para esto: 1) añadir más personas aumenta la comunicación necesaria entre miembros del equipo, lo cual reduce la productividad y, 2) hay trabajo indivisible en un proyecto que sólo puede realizarlo una persona, por lo que añadir más no ayudará.

Aunque este es un concepto útil, no dice específicamente cuántas personas necesitará el proyecto ni cuánto tiempo tomará. Esta área también ha sido tema de investigación; Barry Boehm encontró, por ejemplo, que el tiempo necesario para un proyecto se puede expresar mediante la siguiente fórmula:

$$T = 2.5 * E^{0.33}$$

donde E es el esfuerzo medido en meses-persona; véase [Boehm, 1981] para mayores detalles.

Se han hecho también estudios de la "carga de personas" óptima para un proyecto; las tres fórmulas mejor conocidas se basan en el trabajo de Norden, Putnam y Parr; véase [Norden, 1963], [Putnam y Fitzsimmons, 1979] y [Parr, 1980]. Norden fue el primero en encontrar que la contratación de personal para un proyecto sigue una curva como la que se muestra en la figura B.1.

Se conoce la gráfica como una distribución de Rayleigh, basada en la fórmula matemática de la curva. Putnam proporciona una fórmula para describir el número de personas necesarias para un proyecto en función del tiempo:

$$\text{Personas}(t) = 2K * a * t * \exp(-at^2)$$

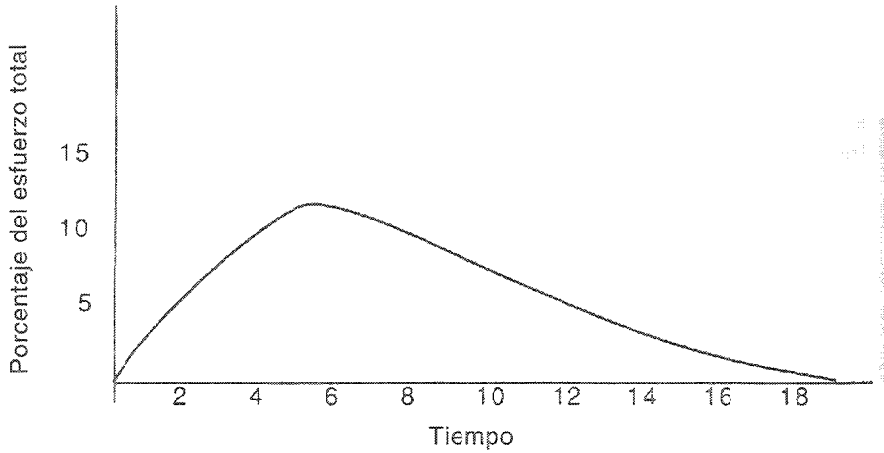


Figura B.1: **Contratación típica de personal para su proyecto**

donde K es el esfuerzo total del proyecto (expresado en meses-persona), y a es un factor de aceleración que establece la pendiente inicial de la curva. (Note que k representa el área total bajo la curva.)

Parr [Parr, 1980] desarrolló otro modelo; aunque la forma global es similar a la de la figura B.1, estima una mayor cantidad de personal en las etapas tempranas. El modelo de Parr se describe mediante la siguiente fórmula.

$$\text{Personas}(t) = 1/4\ \text{sech}^2((at + c)/2)$$

B.5 MODELOS DE ESTIMACION COMPUTARIZADOS

La idea de utilizar fórmulas con exponenciales y secantes hiperbólicas probablemente no sea muy atractiva; puede estar seguro que muchos analistas veteranos olvidaron hace mucho lo que es una secante hiperbólica y no tienen la menor idea de cómo calcular una exponencial. Pero no es necesario recordar ninguna de las fórmulas, ni es necesario realizar a mano los cálculos. Existen ahora muchos paquetes computarizados de estimación de proyectos; la mayoría son ejecutables en PC, y usan el modelo COCOMO de Boehm, lo mismo que el modelo de Putnam descrito anteriormente. Algunos incorporan también las gráficas PERT y GANTT que se discutieron en el Capítulo 16.

Si está trabajando en algo que no sea un sistema trivial, definitivamente debiera investigar tales paquetes. No sólo hacen cálculos sino también permiten jugar con simulaciones de tipo "qué-sucede-si" para observar el efecto de añadir personas al proyecto, o de perderlas debido a enfermedad u otras calamidades.

REFERENCIAS

1. Tom DEMarco, *Controlling Software Projects.* Nueva York: YOURDON Press, 1982.

2. Barry Boehm, *Software Engineering Economics.* Englewood Cliffs, N.J.: Prentice-Hall, 1981.

3. *Workshop on Quantitative software Models for Reliability, Complexity, and Cost: An Assessment of the State of the Art.* IEEE, catálogo nº TH0067-9. Nueva York: Institute of Electrical and Electronics Engineers, 1979.

4. Victor Basili, *Tutorial on Model and Metrics for Software Management and Engineering.* Nueva York: IEEE, 1980.

5. C. E. Walston y C. P. Felix, "A Method of Programming Measurement and Estimation", *IBM Systems Journal,* volumen 16, nº 1 (enero de 1977), pp. 54-73. Reimpreso en *Writings of the Revolution,* Edward Yourdon (editor). Nueva York: YOURDON Press, 1982, pp. 389-408.

6. J.W. Bailey y V.R. Basili, "A Meta-Model for Software Development and Resource Expenditures", *Proceedings of the 5th International Conference on Software Engineering.* Nueva York: Institute of Electrical and Electronic Engineers, 1983.

7. P.V. Norden, "Useful Tools for Project Management", *Operations Research in Research and Development.* Nueva York: Wiley, 1963.

8. Larry Putnam y A. Fitzsimmons, "Estimating Software Costs", *Datamation,* septiembre de 1979, pp. 89-98; octubre de 1979, pp. 171-178, noviembre de 1979, pp. 137-140.

9. F.N. Parr, "An Alternative to the Rayleigh Curve Model for Software Development Effort", *IEEE Transactions on Software Engineering,* volumen SE-6, número 3 (mayo de 1980), pp. 291-296.

10. T. Capers Jones, *Programming Productivity.* Nueva York: McGraw-Hill, 1985.

APENDICE

C CALCULOS DE COSTO/BENEFICIO

C.1 INTRODUCCION

Este apéndice se dedica a las técnicas de cálculos de costo/beneficio, que son una parte importante de todo análisis de sistemas. El propósito, desde luego, es mostrar a los usuarios del nuevo sistema, al igual que a otros grupos de administradores de la organización, que los beneficios que se espera obtener con el nuevo sistema superan a los costos esperados.

Como analista subordinado, pudiera ser que no se esté involucrado para nada en este intento, o se le podría dar la tarea de desarrollar un modelo de costo/beneficio para una pequeña porción del sistema global. Incluso como analista en jefe a cargo de todo el proyecto, pudiera ser que no esté involucrado en los cálculos de costo/beneficio porque podrían estar a cargo, por ejemplo, de un grupo de finanzas separado.

O tal vez ni siquiera se haga. Muchas organizaciones desarrollan sistemas simplemente para satisfacer requerimientos obligatorios de gobierno (por ejemplo, sistemas de reporte para la oficina del trabajo, o nuevos sistemas para manejar la cambiante legislación de impuestos). Desde luego, incluso en estos casos, cuando no existe beneficio derivable del sistema (más que el lujo de evitarse multas o poder permanecer en el negocio), la administración usualmente desea saber cuál será el costo del sistema; pero esto se puede realizar como parte de las actividades de estimación que se discuten en el apéndice B.

Existe otra razón por la cual puede no hacerse el estudio de costo/beneficio: porque el usuario no lo quiera. Así como un consumidor pudiera no tener manera de justificar el costo de un Cadillac (es decir, lo mismo le hubiera servido un VW pequeño o incluso una bicicleta), muchos usuarios no pueden justificar el costo del nuevo sistema que han pedido. A veces piden un nuevo sistema por las mismas razones

que un consumidor compra un automóvil caro: estar a la altura de los Fulanito.[1] En otros casos el usuario puede sentir que existe la necesidad legítima de un nuevo sistema, pero reconocer que todos los beneficios son un tanto intangibles o extremadamente difíciles de cuantificar; en un momento dado, puede justificar la adquisición del nuevo sistema atribuyéndolo a un "presentimiento" de que redituará.

Como analista, no está en posición de insistir en que se realice un estudio de costo/beneficio; después de todo, es el sistema del usuario, y si él quiere construirlo sin justificación alguna, es su prerrogativa. Sin embargo, es buena idea investigar si se ha hecho un estudio de costo/beneficio para el proyecto y, de ser así, si es razonable.[2] Si no lo hay, o si los beneficios son muy difusos (o si se subestimaron exageradamente los costos), debe estar consciente de que el proyecto es vulnerable.

En el peor de los casos encontrará que el usuario no es demasiado entusiasta en lo que se refiere al proyecto, pero la administración superior lo ha convencido basándose en los cálculos optimistas del administrador del proyecto, quien sí desea realmente construir el sistema (porque ayudará a su carrera, porque piensa que todos los usuarios deben estar totalmente computarizados, o por cientos de razones más). En el mejor de los casos, puede encontrar que el usuario ha autorizado el sistema y es muy entusiasta al respecto a pesar de la falta de un cálculo razonable de costo/beneficio. Pero los usuarios son volubles: el proyecto predilecto de hoy se vuelve el proyecto rechazado de mañana.

Y los usuarios van y vienen: el usuario que autorizó el proyecto ayer puede ser sustituido mañana por otro que tiene una visión muy diferente de la deseabilidad del proyecto. Así que, si no existe un cálculo de costo/beneficio (y es evidente que nadie quiere desarrollar uno), mi consejo es muy sencillo: mantenga al día su currículum, pues es probable que esté buscando un empleo nuevo antes de que concluya el proyecto.

En las secciones siguientes examinaremos diversos aspectos de los cálculos de costo/beneficio:

- Análisis de costos
- Análisis de beneficios

1 Esto se da sobre todo en algunas industrias altamente competitivas donde se desarrollan sistemas de cómputo nuevos para ofrecer tipos adicionales de servicios para el mercado (por ejemplo, nuevos sistemas bancarios, sistemas de tarjeta de crédito y sistemas de "viajero frecuente" de aerolíneas). Por tanto, su usuario podría no justificar el costo de un sistema así con base en méritos propios, pero puede sentir que tiene que desarrollarlo para mantenerse al día con la competencia.

2 Como analista en jefe, está en posición de saber esto de inmediato, desde luego. Pero para un analista subordinado, que entró al proyecto después de seis meses de iniciado puede no ser tan evidente. Para ese punto el proyecto tiene vida propia y luchará por su existencia independientemente del usuario y de cualquier otro proceso racional de toma de decisiones.

- Cómo expresar los ahorros
- Análisis de riesgos

C.2 ANALISIS DE COSTOS

El propósito de esta actividad, desde luego, es calcular todos los costos anticipados asociados con el sistema: no sólo el costo de *construirlo*, sino también el costo de *instalarlo*, de *operarlo* y de *mantenerlo*, además de los costos extras. Cada uno de éstos se discute a continuación.

C.2.1 El costo de construir el sistema

En el apéndice B discutimos las técnicas de estimación de la cantidad de tiempo necesaria para construir un sistema y la cantidad de personas que se requiere. Tenga presente que no sólo necesita calcular el costo de analistas y programadores, sino también de toda la demás gente involucrada con el desarrollo del sistema:

- Oficinistas
- Administradores
- Miembros de la comunidad usuaria
- Consultores y programadores bajo contrato
- Posiblemente miembros del personal de auditoría, de control de calidad o de operaciones.

En la mayoría de los casos, de la administración puede obtener el salario promedio de las distintas categorías de personas incluidas en su proyecto (o del departamento de contabilidad); esto se puede expresar en términos de costos por hora, por mes, o anuales. Asegúrese de tomar en cuenta los factores extras de su compañía; es decir, probablemente tenga que multiplicar cada salario por un factor de, digamos, 150 por ciento para cubrir el costo de seguros, beneficios a empleados y otros varios. Nuevamente, esto se puede obtener de la administración o del departamento de contabilidad.

Tenga presente también que quienes trabajan en el proyecto no siempre están disponibles el cien por ciento del tiempo: necesariamente se perderá algo de tiempo por concepto de enfermedades, vacaciones, licencias, etc. Su compañía podría no tener un factor estándar a aplicar a este tiempo perdido; de ser así, suponga por lo menos un factor el 10 por ciento. Tampoco está muy lejos de lo realista un 20 por ciento o un 25 por ciento. (Es posible que esto ya se haya tomado en cuenta al hacerse la estimación de recursos que se describe en el apéndice B.) Revise para asegurar que el factor de pérdida de tiempo no se haya omitido ni se haya aplicado dos veces.

En muchos proyectos deberá incluir también el costo de preparar al personal de desarrollo. Puede ser que los miembros del equipo necesiten prepararse en el área de las nuevas metodologías de desarrollo, los nuevos lenguajes de programación o las diversas habilidades sobre el software y hardware asociados con el equipo comercial que se esté usando. Otro costo que hay que tener en cuenta es el de tiempo de computadora, terminales o estaciones de trabajo y herramientas de desarrollo (editores, paquetes de prueba, etc.) que se ocupan para el desarrollo del sistema. En algunos casos, las terminales y las herramientas pueden existir ya y por tanto el proyecto no incurrirá en gastos adicionales; en casi todos los casos, sin embargo, el proyecto tendrá que incluir los costos del tiempo de computadora. (Note que esto también puede incluir costos de almacenamiento en disco y costos de telecomunicaciones, al igual que costos de papel, formas y otros materiales extras.)

Algunos proyectos nuevos se desarrollan con gente nueva, es decir, personas que no trabajaban para la organización antes de comenzar el proyecto, y para la cual, sin duda, no existía espacio de oficina. Por tanto puede tener que incluir costos de reclutamiento (gastos de viaje para los candidatos al trabajo, pagos de agencias de empleo, etc.), además de gastos de empleados asociados con la preparación inicial que debe tener un nuevo empleado. Puede ser necesario incluir el costo de espacio de oficina, escritorios, teléfonos y otro tipo de equipo para el personal nuevo.

En algunos proyectos, también surgen gastos de viaje por visitas que se deben hacer a alguna sede lejana para poder entrevistar usuarios. Obviamente, éste no es un factor a tomar en cuenta en un proyecto donde todos los usuarios estén localizados en la misma zona geográfica que el equipo de desarrollo; pero en proyectos con diversos grupos de usuarios en diferentes localidades (a veces en distintos países), esto puede representar un gasto importante. Por cierto, a menudo la administración supondrá que toda la información necesaria se puede recabar en un solo viaje; en proyectos reales suelen requerirse viajes subsecuentes para solucionar incógnitas y malos entendidos.[3]

Así, los costos de desarrollo de un sistema pueden ser variados y múltiples. La siguiente lista resume la discusión anterior; puede no ser completa, pero cubre lo principal:

- Los salarios y gastos extra para todo el personal relacionado con el proyecto.
- Costos de capacitación
- Tiempo de computadora y herramientas de desarrollo para el personal
- Costos de reclutamiento del personal nuevo

3 Esto se puede minimizar a veces si su organización tiene extenso acceso a correo electrónico u otras formas de comunicación además del teléfono.

- Espacio de oficina y equipo para el personal nuevo
- Gastos de viaje para visitar a usuarios lejanos

C.2.2 El costo de instalar el sistema

En un proyecto sencillo, pudiera ser suficiente llamar por teléfono al usuario y decirle que se terminó de desarrollar el sistema; puede entregarse en un disco flexible y dejar que él mismo lo instale en su computadora personal. Pero en un proyecto grande y complejo, el proceso de instalación es más difícil e incluye muchos gastos. Entre ellos tenemos los siguientes:

- Gastos de capacitación de usuarios
- Gastos de conversión de bases de datos
- Gastos de instalación comercial
- Gastos de aprobación reglamentaria
- Gastos de ejecuciones paralelas
- Gastos del equipo de desarrollo durante la instalación

Típicamente, toda la comunidad usuaria necesitará de cierta capacitación para familiarizarse con el uso del sistema. Puede requerirse también capacitación adicional para los usuarios supervisores y también para el personal de operaciones y otros miembros extra del personal. Note que esto significa que también debe incluirse el costo de desarrollar cursos de preparación para los usuarios, el costo de manuales o papelería de los cursos, y el costo de lugares de capacitación para los usuarios (aulas, etc.). Finalmente, no olvide el costo del *tiempo* del usuario durante este proceso de capacitación; pueden pedirle calcularlo en términos de los salarios de los usuarios, o puede calcularse en términos del costo de los *sustitutos* que realizan la labor de los usuarios mientras éstos se están preparando.

Los costos de conversión de bases de datos pueden ignorarse si se está instalando un sistema nuevo para el cual no existe precedente. Pero si el sistema nuevo reemplaza a uno anterior, seguramente existirá una base de datos que necesite incorporarse. Si la base de datos existente no tiene forma computarizada (por ejemplo, archivos llenos de papeles), entonces habrá un gasto substancial asociado con el ingreso de datos; es decir, alguien (o posiblemente un grupo grande de personas) probablemente se tendrá que sentar frente a una terminal a teclear todos los datos relevantes para el nuevo sistema. Si los datos existentes ya están computarizados, puede haber un pequeño costo para traducir mecánicamente esos archivos al nuevo formato.[4]

4 Existe un costo oculto del que debe estar consciente: durante la conversión de la base de datos anterior a la nueva, es inevitable que se encuentren errores. Esto sucede sobre todo, como se po-

Los costos de instalacion comercial no se deben ignorar, sobre todo si el sistema incluye hardware nuevo, equipo nuevo de telecomunicaciones y/o software nuevo. Los proveedores generalmente le darán una buena estimación de los costos de instalación, y debe poder lograr una cotización fija.

Para algunos sistemas, puede haber un gasto en licencias u otras formas de aprobación reglamentaria, por parte de diversas autoridades gubernamentales locales, estatales o federales. Esto también puede incluir cosas tales como pruebas ambientales contra emisiones de radiación de las pantallas de video que usan los operadores de ingreso de datos. Si la aprobación es un procedimiento burocrático que simplemente involucra el llenado de formas, debe poderse estimar el costo de manera bastante precisa; si por el contrario involucra un proceso de prueba, su estimación puede ser mucho más aproximada.

El costo de ejecuciones en paralelo, de haberlas, también debe incluirse en la estimación de los costos de instalación. Para muchos tipos de sistemas, el usuario insistirá en que el sistema anterior se use en paralelo con el nuevo durante algún periodo de tiempo. Esto puede involucrar una duplicación temporal de personal usuario y otros gastos relacionados. Investigue (con suerte lo encontrará en la especificación del sistema) cuánto durará la ejecución paralela; esto debe ayudarle a realizar una estimación apropiada.

Asegúrese que no se le vaya a olvidar el costo del personal de desarrollo que participa en la instalación. Típicamente, los programadores y analistas involucrados con el desarrollo del sistema lo estarán también con su instalación. Obviamente, además de sus sueldos (y posible tiempo extra) y beneficios adicionales, debe tomar en cuenta también gastos de viajes que se pudieran requerir para instalar el sistema en alguna sede lejana del usuario.

Finalmente, tenga presente que para los sistemas grandes, la instalación no se hará de golpe; por ejemplo, un nuevo sistema bancario puede instalarse en una sucursal tras otra, durante un periodo de algunos meses. En general, esto significa que el costo de instalación de las sucursales iniciales (o áreas de usuarios) será mayor que el de las subsecuentes, porque el equipo de instalación tendrá cada vez más experiencia y (con suerte) los problemas iniciales que pudo haber con el sistema se habrán erradicado tras las primeras instalaciones. Por otro lado, si el proceso de instalación dura varios meses (o incluso años), debe tomar en cuenta la posibilidad

drá imaginar, si la base de datos existente es manual y las entradas se han hecho a mano; encontrará datos faltantes, incompletos y datos obviamente incorrectos. Entre más históricos sean los datos, más errores es probable encontrar. Además, la conversión misma puede dar lugar al surgimiento de errores, sobre todo si es un proceso manual. Por ello, es probable que exista un gasto asociado con la *corrección* de datos. Probablemente sea buena idea tomar una muestra al azar de la base de datos existente para estimar el número de errores que se encuentren; luego, hay que estimar el costo de corregirlos (en la mayoría de los casos, las correcciones se tendrán que hacer manualmente).

de cambio de personal: la gente que ha adquirido experiencia en la instalación del sistema y la preparación de los usuarios se puede aburrir del proceso y cambiarse a un nuevo trabajo en otra parte.

C.2.3 El costo del dinero

El dinero que se requiere para desarrollar e instalar un sistema no se da en los árboles; la organización tiene que pedirlo prestado, o bien debe sacarse de sus fondos actuales. Por tanto, existe un costo asociado con el uso del dinero. Dependiendo de su organización, se le puede pedir que exprese esto en términos del costo del dinero prestado, o de los intereses que ganaría si se tuviera invertido en lugar de estarse usando para el proyecto.

Esta es un área en la que debe recurrir al consejo del departamento de contabilidad. Ellos tendrán casi seguramente una regla estándar acerca de cómo tratar tales cuestiones, y es importante que su proyecto utilice el mismo enfoque.

C.2.4 Costos operacionales

Una vez instalado el sistema, al usuario le costará dinero continuar operándolo. Sin embargo, esto también debe representar un área en la que el nuevo sistema ahorrará dinero, dado que es de suponerse que será más económico que el que actualmente tiene el usuario (a menos que haya añadido mayor funcionalidad). Algunos ejemplos típicos de costos operacionales son:

- Costos de hardware y materiales y equipo relacionados
- Costos de software
- Costos de personas
- Costos de mantenimiento
- Costos de facilidades

Aquí el término *hardware* es muy amplio. Abarca desde el costo del equipo de cómputo (suponiendo que no se haya comprado ya, pero vea la sección C.2.6 también), del equipo de telecomunicaciones, de terminales, de estaciones de trabajo, y materiales (como papel, formas, discos flexibles, gavetas para discos, cintas de impresora, etc.). Tenga presente también que parte del hardware se puede considerar consumible en el sentido de que se desgasta o necesita reemplazarse. Esto incluye terminales, algunas impresoras y tal vez otros tipos de equipo. Asegúrese de que se coticen costos de refacciones de manera adecuada.

Los *costos de software* en esta discusión significan los gastos continuos de renta de sistemas operativos, paquetes de administración de bases de datos y otros tipos de software que se puede haber rentado de algún proveedor.

Los *costos de personas* incluyen al personal de operaciones, personal de apoyo técnico, programadores de mantenimiento y el costo de los usuarios directamente involucrados con la operación cotidiana del sistema. Como se discutió anteriormente, probablemente tenga que expresar esto como un costo aumentado para poder tomar en consideración seguros, beneficios y gastos extras de la corporación.

Los *costos de mantenimiento* incluyen el costo mensual (o anual) previsto para el equipo de cómputo; su estimación debe incluir los costos no sólo del mantenimiento preventivo que proporciona el proveedor, sino también costo extra de reparaciones en caso de que se descomponga el equipo. Incluya también costos de mantenimiento de software comercial, si se necesita; el contrato de mantenimiento que ofrecen los proveedores usualmente incluye un número telefónico de consulta para apoyo técnico, además de actualizaciones gratuitas (o de costo reducido) a nuevas versiones de sus paquetes.

Además, sus costos de mantenimiento deben incluir una estimación del costo de mantenimiento para reparación y mejoramiento de software de aplicación. Puede ser un factor importante en el costo, como lo muestra el hecho de que muchas organizaciones gastan más del 50 por ciento de su presupuesto de proceso de datos en mantenimiento. Hay varias maneras de estimar los costos del nuevo sistema:

- Si el sistema reemplaza a uno anterior, puede estimar que el nuevo requerirá la misma cantidad de esfuerzo de mantenimiento. Este es un esfuerzo conservador, ya que supone que el anterior se desarrolló usando técnicas relativamente modernas de ingeniería de software y que el nuevo no usará técnicas más modernas o eficientes que éstas. Esto es poco probable si el sistema actual tiene de 10 a 15 años de antigüedad, pero por lo menos representa una especie de estimación del peor de los casos.
- Una estimación optimista puede basarse en el ahorro esperado respecto al mantenimiento del sistema actual. Muchas organizaciones han encontrado que, por ejemplo, sus costos de mantenimiento se han reducido en un factor de cinco o más gracias al uso cuidadoso del análisis, diseño, y programación estructurados.[5] Investigue otros proyectos similares dentro de su propia organización para ver si se han logrado ahorros de este tipo; de ser así, puede ser razonable esperarlos en su proyecto. Sea cauteloso, sin embargo, en cuanto a la tentación de mostrar reducciones considerables de personal basadas en la instalación de su nuevo sistema; rara vez sucede, por las razones que se discuten en la sección C.3.1.
- Si actualmente no se está utilizando algún sistema que pueda servir de comparación (o si desea evitar estimaciones demasiado optimistas o pesimistas), trate de determinar el costo promedio de mantenimiento para sistemas similares dentro de su organización. Esto probablemente se basará en alguna unidad normalizada (por ejemplo, costos de mantenimiento por línea de código por año, o costos de mantenimiento por punto de función por año), pero las estimaciones realizadas en el apéndice B deben permitirle hacer estimaciones de mantenimiento apropiadas para su proyecto.

5 Para cálculos detallados de estos ahorros, consulte el libro de Capers Jones, *Programming Productivity*. (Nueva York: McGraw-Hill, 1985).

Un costo final que debe estimarse cuando se calcula el costo operacional del nuevo sistema es el de la planta física (por ejemplo, el cuarto de computadoras y las oficinas para el personal de operaciones, la gente de mantenimiento del proveedor y el personal usuario). Si el nuevo sistema va a operar en una computadora principal centralizada que ya está instalada, estos costos pueden estar incluidos en el costo global de hardware que se discutió anteriormente. Sin embargo, si está desarrollando un sistema completamente nuevo que tendrá su propio local de operaciones, éste pudiera ser un gasto importante.

C.2.5 El costo del fracaso

Existe otro gasto más que debe tener en cuenta: el costo de posibles fallas del nuevo sistema. Es conveniente sepultar esto en la categoría de costos operacionales, pero tiende a ocultar lo que se convertirá en un aspecto importante de los sistemas de información en un futuro: su confiabilidad.

Existen varias formas de falla de sistemas, como se podrá imaginar; en algunos casos, el sistema queda totalmente fuera de operación hasta que se corrige la falla, mientras que en otros, continúa operando, pero una o varias de sus salidas son incorrectas. En algunos casos, algunas funciones pueden ser inoperables y en otros puede ser que los usuarios no puedan tener acceso al sistema. Todas estas formas de falla tienen un costo asociado: costos de hardware, costos de software, costos de personal relacionado con la corrección del error, costos legales en el caso de que la falla del sistema haya ocasionado pérdidas financieras u otras pérdidas lamentables, y costos asociados con la posible pérdida de ganancias o pérdida de clientes.

¿Cómo se debe estimar ésto? Sería ingenuo ignorarlo por completo pues aún no hemos logrado la capacidad de construir sistemas perfectos; por otro lado, si pregunta a los programadores o analistas que laboran en el proyecto cuántas fallas preveen que el sistema nuevo puede tener, lo mirarán como si le acabara de dar una nueva forma de senilidad.

Tal vez lo más responsable que puede hacer es 1) estudiar la relación de fallas del sistema actual, si está construyendo uno nuevo para reemplazarlo y, 2) ver

6 Esto supone, desde luego, que alguien de su organización está llevando un registro de estas cosas. Una encuesta de cas 500 instalaciones de proceso de datos en los EUA conducida por Lientz y Swanson en 1980 sugiere que aproximadamente un 50% de las organizaciones no llevaron un registro de las fallas operacionales en sus sistemas; véase el libro *Software Maintenance Management,* de Lientz y Swanson, Reading Mass.: Addison Wesley, 1980.

la relación de fallas de todos los demás sistemas de su organización.[6] Así puede hacer algunas suposiciones razonables sobre la relación de fallas que tendrá el sistema nuevo. Con suerte, su proyecto se podrá construir con una cantidad substancialmente menor de errores que la del actual, o incluso que la del sistema promedio de su organización; de hecho, intente lograr una mejoría del 10 por ciento si no más.

Si no existen estadísticas disponibles para la confiabilidad del software del sistema actual de su organización, y no tiene una base sobre la cual hacer una estimación para el nuevo, entonces por lo menos debe considerar este hecho en su documento de análisis de riesgos; para más información al respecto vea la sección C.5. Si está construyendo un sistema grande y complejo, en el cual una falla tendría consecuencias potencialmente desastrosas y de gran alcance, no es profesionalmente aceptable no tener un modelo de confiabilidad de software, a pesar del hecho de que la mayoría de las organizaciones no se preocupan de hacerlo por el momento. Para obtener más información al respecto, vea el libro *Software Reliability* de Glen Myers (Reading, Mass.: Addison-Wesley, 1979).

C.2.6 Distinga entre costos de capital y costos de operación

Algunos costos asociados con el nuevo sistema, se desembolsarán durante el año en el que se presentan; es decir, su organización los reconocerá en su declaración de impuestos durante el año en el cual ocurren. Otros se capitalizan; es decir, se reparten a lo largo de un periodo de varios años. Aunque esto no afecta al costo global del sistema, su clasificación como costo capitalizable o costo de operación puede tener un impacto enorme sobre los impuestos de la organización. De manera similar, la decisión de realizar algunos gastos sobre una base de compra en lugar de renta puede tener un impacto tremendo sobre el flujo de efectivo de la organización, aun cuando el costo total siga siendo el mismo.

Típicamente, las compras de hardware se consideran gastos de capital, y su costo se reparte a lo largo de un periodo de 5 a 7 años (dependiendo de los reglamentos de impuestos prevalecientes). El costo de desarrollar software puede capitalizarse o no. Y los costos de instalación y operación normalmente se contabilizan cuando ocurren, aunque pueden existir pequeñas variantes en esta área.

Obviamente, aquí no puede inventar su propia política de contabilidad. Es importante investigar qué parámetros financieros se usan en su organización y seguirlos de manera consistente.

C.3 ANALISIS DE BENEFICIOS

Es mucho más difícil calcular los beneficios de un nuevo sistema de información que calcular su costo. En algunos casos, como se mencionó al principio de este apéndice, puede ser imposible. Debido tal vez a que el sistema es obligatorio, o porque el usuario decidió que quiere el sistema sin importar si se pueden identificar beneficios tangibles o no.

El intento de calcular beneficios *tangibles* es el que ocasiona tantos problemas. Los usuarios probablemente hablarán entusiastamente acerca de "mejor control" o "información más oportuna" o "mejores marcos para toma de decisiones", pero si les pregunta cuánto dinero va a ahorrar o cuántas ganancias reportará, es probable que contesten: "Pues... mucho ...sencillamente, sé que será magnífico..." De hecho, probablemente lo será, pero términos como "magnífico" no tienen mucha cabida en hojas de cálculo que muestran comparaciones numéricas de costo/beneficio.

Por ello, su más grande labor al llevar a cabo un cálculo de costo-beneficio será acorralar a los usuarios y hacer que identifiquen beneficios tangibles que puedan medirse y calcularse de manera cuantitativa. Si no lo puede lograr (lo cual suele ser el caso para muchos proyectos y analistas), trate de lograr que comparen su nuevo sistema con algún otro con beneficios conocidos. Así, puede decir al usuario: "Suponga que tuviera que elegir entre el sistema nuevo del que estamos hablando y el sistema X. ¿Cuál consideraría más importante? Si sólo se pudiera implantar uno de ellos, ¿cuál escogería?" Suponiendo que el sistema X tiene algunos beneficios tangibles asociados, esto debiera darle por lo menos una manera burda de determinar el valor aproximado del nuevo sistema.

En las secciones siguientes, distinguiremos entre beneficios tácticos y estratégicos de un nuevo sistema. En este contexto, un beneficio táctico es aquél que permite que la organización continúe realizando la misma actividad de negocios, pero a menor costo (o mayor ganancia); un beneficio estratégico es el que permite comenzar a realizar un tipo de negocio totalmente nuevo, o a hacerlo en un área totalmente nueva o con clientes nuevos.

C.3.1 Beneficios tácticos

Los beneficios tácticos suelen asociarse con reducciones en el personal administrativo o de oficina. Aunque esto no es música para los oídos de los oficinistas, es una realidad. Un nuevo sistema de información puede permitir que se realice la misma función con la mitad o menos del número de usuarios que se ocupaban antes. Esto generalmente se debe a que los usuarios actualmente están realizando cálculos o actividades de registro de datos a mano, cuando pudieran computarizarse; o se ven forzados a realizar las mismas actividades (o registrar los mismos datos) múltiples veces, cuando puede hacerse una vez con una computadora; o les toma una gran cantidad de tiempo recuperar los datos, siendo que puede hacerse rápidamente por computadora.

Aunque éste sea un beneficio obvio del nuevo sistema, tenga cuidado de no sobreestimar el efecto. En algunos casos, habrá menos ahorro de lo que puede haber estimado; los reglamentos del sindicato y la bondad de algunos administradores intermedios de la organización usuaria pueden evitar el despido de algunos de esos usuarios oficinistas. Además, es igualmente importante darse cuenta que los niveles superiores de la administración se impresionan cada vez menos con el ahorro de

uno o dos oficinistas; buscan beneficios mayores y mejores con la introducción de un nuevo sistema.

Un tipo de beneficio táctico mucho más interesante es el ahorro que resulta de poder procesar transacciones de negocios más rápidamente. Poder manejar más transacciones por segundo no sólo permite reducir costos de oficina, sino que puede llevar a un mejor flujo de efectivo para la organización (convirtiendo el pedido del cliente en efectivo más rápidamente, acelerando el tiempo de entrega, etc.). Nuevamente, el secreto radica en en cuantificar esto y expresarlo como una cantidad en dinero. Pero, como ejemplo, considere el siguiente diálogo entre el usuario y el analista:

Analista: "Bueno, ¿cuántos pedidos diarios procesa?"

Usuario: "Bueno, procesamos 10,000 pedidos diarios. Pero siempre hay un retraso, así que toma una semana, en promedio surtir el pedido de un cliente."

Analista: "Y la factura se le manda al cliente junto con la mercancía ¿no es así? Por lo que no se espera que el cliente pague hasta haberla recibido, ¿o no?"

Usuario: "Cierto."

Analista: "Así que si podemos reducir la tardanza en el proceso del pedido de cinco días a uno, obtendríamos el dinero del cliente, en promedio, cuatro días antes. Y si el pedido promedio es de $1,000 y procesamos 10,000 pedidos diarios, eso significa que estamos hablando de alrededor de $10'000,000 diarios. El tener en nuestras manos $10'000,000 cuatro días antes de lo que se esperaba antes, por concepto de los pedidos diarios, valdría..."

Un nuevo sistema también puede reportar ahorros en equipo de cómputo; el sistema anterior puede estar funcionando en una computadora principal cara, mientras que el nuevo funciona en una pequeña PC colocada en el escritorio del usuario. Un cambio así no sólo ahorra costos de hardware, sino tambien representa un ahorro en el área de costos de local, de operadores, etc. Y si el nuevo sistema reduce la cantidad de papel y de formas impresas, también ahí debe reflejarse un ahorro. Asegúrese de que sus cálculos en cuanto a esto estén completos; tenga en cuenta que pueden requerirse menos archiveros, menos espacio de oficina, menos máquinas de escribir, y posiblemente menos llamadas telefónicas entre su organización y los clientes, etc.

Los costos de mantenimiento del nuevo sistema también deben proporcionar un beneficio, como se discutió en la sección anterior. Los costos de mantenimiento de hardware deben reducirse (a menos que esté ejecutando su nuevo sistema en el mismo equipo de cómputo que ya está instalado en la organización), y los costos de mantenimiento de software es de suponerse que serán inferiores a los del sistema actual.

C.3.2 Beneficios estratégicos del nuevo sistema

En cada vez más casos, los beneficios realmente interesantes e importantes de un nuevo sistema son beneficios *estratégicos*. No sólo se trata de la oportunidad de ahorrarse unos cuantos oficinistas o unas cuantas hojas de papel, sino de la posibilidad de permitirle a la organización hacer cosas que le serían imposibles con el sistema actual. Existen varios ejemplos de beneficios estratégicos potenciales:

- Identificar o atraer nuevos clientes que de otra manera no podría identificar la organización.
- Entrar a nuevos mercados o proporcionar nuevos productos que previamente no estaban disponibles
- Capturar, reproducir o distribuir conocimientos y experiencia a los que previamente sólo tenían acceso una o dos personas dentro de la organización.

En una economía tan competitiva como parece ser la actual, un sistema de información que puede atraer nuevos clientes o evitar la pérdida de los actuales por la competencia es realmente muy valioso. En algunos casos, esto es posible gracias a la funcionalidad que ofrece el nuevo sistema, que anteriormente no estaba disponible; en otros, puede resultar de la capacidad del sistema para identificar clientes potenciales nuevos que anteriormente ignoraba la organización. Sea cual sea la situación, trate de cuantificar este beneficio en términos del aumento de clientes o de mercado y, de ahí, en términos de ganancias.

Una forma más difícil de beneficio estratégico es la capacidad del sistema para proporcionar información que anteriormente no se tenía. El ejemplo típico es la capacidad de identificar tendencias y patrones (por ejemplo, tendencias de ventas por territorio, por estación, o por preferencias del cliente hacia ciertos productos). Esto es posible en casi cualquier sistema automatizado que reemplace a uno manual; y usualmente cualquier sistema en línea o de tiempo real presenta tales tendencias de una manera más oportuna que la lograda con un sistema por lotes. De manera similar, un sistema con capacidades gráficas puede proporcionar información de una manera más efectiva que uno actual que produzca información en forma de tablas y salidas numéricas. Un sistema construido con un lenguaje de programación de cuarta generación y un sistema de administración de bases de datos moderno puede permitir consultas ad hoc.

Una forma relativamente nueva de beneficio que se logró con los sistemas expertos comerciales es la difusión de conocimientos que previamente sólo estaban al alcance de una o dos personas. Este conocimiento típicamente es de naturaleza evaluativa, diagnóstica o de juicio, y el experto humano que posee esa capacidad

usualmente se considera un bien valioso para la organización; de aquí que la capacidad del nuevo sistema de duplicar esto sea un beneficio cuyo valor debe poderse calcular.

En tanto que las técnicas de inteligencia artificial continúan creciendo, identificaremos como beneficio la capacidad de un sistema de *extender* el conocimiento que alguna vez fue sólo de unos cuantos expertos humanos de la organización. Así, un experto humano dentro de la organización puede tener reglas de juicio que utiliza para diagnosticar la posible causa de fallas en algún sistema mecánico (en una plataforma de extracción de petróleo, por ejemplo). Obviamente, sería de beneficio estratégico capturar el conocimiento de dicho experto humano y duplicarlo para el uso de otros; pero podría ser aún más valioso extender la experiencia e incrementar la capacidad de diagnóstico para tratar con las fallas del sistema.

C. 4 COMO EXPRESAR LOS COSTOS Y BENEFICIOS DEL SISTEMA

Si se incurriera de manera instantánea en todos los costos del sistema, y todos los beneficios se observaran de manera instantánea, sería relativamente sencillo representar el valor del sistema como la diferencia entre costos y beneficios. Pero como se mencionó anteriormente, los gastos usualmente se hacen en el transcurso de algunos años; y si el gasto se llegara a hacer en un momento (por ejemplo, una compra de hardware), las políticas de contabilidad de la organización pudieran dictaminar que se reparta a lo largo de un periodo de varios años.

Así, probablemente tendrá que demostrar los costos y beneficios que del sistema a lo largo de cierto período de tiempo. Existen cuatro métodos comunes para hacer esto:

- Flujo de efectivo
- Rendimientos de inversiones (en inglés, ROI)
- Tasa interna de rendimiento (IRR)
- Valor neto actual (NPV)

Cada uno se discute a continuación.

C.4.1 Flujo de efectivo

Ya sea que el sistema muestre algún día ganancias o no (ganancias que excedan a los costos), la administración desea saber cuánto efectivo se invertirá antes de esperar un flujo positivo; obviamente, se preocuparán más acerca de esto para proyectos grandes que para proyectos chicos. Note que el flujo de efectivo del proyecto puede ser muy distinto al de la información que originalmente se reportó como pérdidas y pérdidas para la organización. Por ejemplo, el equipo del proyecto puede gastar $100,000 en sueldos durante un esfuerzo de un año en el desarrollo de sistemas;

pero las leyes sobre impuestos pueden permitir que el costo se amorticen a lo largo de un periodo de, digamos, 5 años. Así, la organización puede reportar gastos de sólo $20,000 en sus formas de impuestos para un año, pero los $100,000 en efectivo definitivamente ya se erogaron. De manera similar, los beneficios del nuevo sistema pueden parecer muy distintos desde un punto de vista de flujo de efectivo que desde el punto de vista de las pérdidas y ganancias de la organización que se reportan a Hacienda.

En la mayoría de los casos, es apropiado mostrar tanto un flujo anual como uno agregado. El estudio de costos y beneficios pudiera producir para la administración una tabla como la siguiente:

Proyecciones del flujo de efectivo del proyecto X

	Año 1	Año 2	Año 3	Año 4	TOTAL
Ahorros	0	10,000	50,000	100,000	160,000
Gastos	50,000	30,000	20,000	10,000	110,000
Efectivo neto	-50,000	-20,000	30,000	90,000	50,000
Efectivo agregado	-50,000	-70,000	-40,000	50,000	50,000

C.4.2 Rendimiento sobre inversiones

Otra manera de evaluar los costos y beneficios del sistema es calcular el *rendimiento de la inversión.* Suponga, por ejemplo, que invirtió $110,000 en bienes raíces o en valores y que luego lo vendió todo por $160,000. (Note que éstas son las mismas cifras que se usan en el ejemplo de flujo de efectivo anterior.) Esto significaría que logró una ganancia de $50,000 sobre una inversión de $110,000; expresado en términos de porcentaje, esto significa que su rendimiento de la inversión fue del 45 por ciento.

Esto suena mejor que invertir dinero en una cuenta de ahorros. Pero, espere ...en el ejemplo anterior, la ganancia no se obtuvo al final del primer año; de hecho, tardó cuatro años. Así que esto hace que el rendimiento de la inversión sea aproximadamente del 11 por ciento anual. Aun esto resulta engañoso, sin embargo, porque no toma en cuenta el valor actual del dinero futuro. Esto se discutirá a continuación.

C.4.3 Valor actual neto

Si alguien le diera $100 hoy, sabría cual es su valor: tendría una buena idea de cuánto puede comprar con esa cantidad. Pero ¿cuánto valdrán $100 si sabe que no los recibirá sino hasta dentro de un año? Esto se conoce como el valor actual o el valor descontado. El valor actual del dinero que recibirá en un futuro se define como la cantidad que tiene que invertir *hoy,* con los intereses actuales, para poder lle-

gar a la cantidad especificada. Así, el valor actual de los $100 del próximo año es de aproximadamente $95.24, con intereses del 5%.

En general, si deseamos calcular el valor actual de alguna cantidad de dinero (que llamaremos F), dentro de n años, podemos usar la siguiente fórmula:

$$P = F / (1 + i)^n$$

donde i es el interés. Así, en el ejemplo anterior, el valor actual de los beneficios se puede calcular como sigue (suponiendo un interés del 5%):

Cálculos de valor actual para el proyecto X

	Año 1	Año 2	Año 3	Año 4	TOTAL
Ahorros	0	10,000	50,000	100,000	160,000
Valor actual	0	9,070	43,192	82,270	134,532

Como puede ver, esto hace mucho menos impresionante el rendimiento financiero del proyecto, pero es más realista. Para ser aún más realistas, sin embargo, debemos darnos cuenta que los costos a futuro se deben descontar de la misma manera que los beneficios. Así como un beneficio de $10,000 al final del segundo año sólo valdrá $9,070 hoy, de la misma manera un gasto de $10,000 que se haga al final del segundo año representa un costo actual de solamente $9,070.

Esto nos lleva a la definición de *valor actual neto* de un proyecto: la diferencia entre el valor actual de los beneficios y el valor actual de los costos. Para nuestro proyecto de muestra, esto llevaría a los siguientes cálculos:

Cálculos de valor actual neto del proyecto X

	Año 1	Año 2	año 3	Año 4	TOTAL
Ahorros	0	10,000	50,000	100,000	160,000
Valor actual de los beneficios	0	9,070	43,192	82,270	134,532
Gastos en efectivo	50,000	30,000	20,000	10,000	110,000
Valor actual de los costos	47,619	27,211	17,277	8,227	100,334
Valor actual neto acumulativo	-47,619	-65,760	-39,845	34,198	34,198

Así, el valor neto actual del sistema, es decir, el valor *hoy* de la ganancia que esperamos recibir del sistema al cabo de 4 años es de $34,198.

C.4.4 Tasa interna de rendimiento

La *tasa interna de rendimiento* es análoga al porcentaje que anuncian los bancos, los fondos de mercado y otras instituciones financieras acerca de sus cuentas de ahorro y demás oportunidades de inversión. La tasa interna de rendimiento (IRR) se define como la tasa de interés que se ocuparía para generar los ahorros cada año (es decir, los beneficios del sistema, que ya hemos identificado) dada una inversión igual a los gastos en efectivo que hemos identificado. En el ejemplo anterior, piense que se invirtió un total de $110,000 en una cuenta imaginaria de ahorros durante un período de 4 años. La pregunta es: ¿Qué tasa de interés habría que tener para retirar un total de $160,000 para el final del cuarto año, *sin dejar dinero en el banco?* Comparando esto con la tasa primaria y otras tasas de inversiones diversas, la administración puede determinar si el nuevo sistema es, de hecho, una buena inversión.

Suponga que se describen los beneficios futuros que se lograrán al cabo del año 1, el año 2, ..., y el año N como B1, B2, ..., BN; suponga también que los gastos futuros se describen como C1, C2, ..., CN. Así, debe resolverse la siguiente fórmula polinomial para la tasa de interés i:

$$C1/(1+i) + C_2/(1+i)^2 + ... + C_N/(1+i)^N = B_1/(1+i)^1 + B_2/(1+i)^2 + ... + B_N/(1+i)^N$$

Este no es el tipo de fórmula que se pueda resolver fácilmente en una servilleta o con una calculadora sencilla de cuatro funciones. Sin embargo, hay calculadoras especializadas con funciones de IRR integradas; además, existe una gran diversidad de programas de aplicación especial para este tipo de análisis financiero, basadas en PC o en computadoras grandes. Si no tiene tales herramientas a la mano, pida ayuda al departamento de finanzas o de contabilidad.

C.5 ANALISIS DE RIESGOS

Como parte de los cálculos de costo-beneficio, debe estar dispuesto a llevar a cabo un análisis de riesgos del nuevo sistema. Si la administración no le pide uno, debe hacerlo de todos modos. La razón de esto es que no podemos suponer con certeza que se lograrán los beneficios estimados, ni que se incurrirá en los gastos estimados. Las cosas pueden ser mejores: pero resulta de mayor preocupación la posibilidad de que sean peores.

La administración generalmente deseará saber las consecuencias de que las cosas salgan mal durante el proyecto; y desearán saber *cuáles* pueden ir mal. Específicamente, querrán saber bajo qué condiciones los costos estimados podrían ser significativamente mayores, así como las condiciones bajo las cuales los beneficios fueran bastante menores de lo estimado.

¿Cómo pueden los costos ser mayores de lo estimado? He aquí algunas posibilidades; a usted le toca identificar los riesgos específicos inherentes a su propio proyecto:

- El vendedor del hardware podría quebrar.
- El equipo del proyecto pudiera sufrir un cambio extremo, enfermedad u otros problemas.
- La tecnología usada para el proyecto pudiera no funcionar como se anunció, sobre todo si jamás se había usado antes.
- Podría perderse la oportunidad; por ejemplo, el sistema podría no estar listo para su instalación sino hasta el 2 de enero, y los reglamentos gubernamentales impiden su instalación hasta el *siguiente* 1º de enero.
- Pueden surgir problemas políticos o de contrato con sindicatos, contratistas externos, etc.
- El equipo del proyecto podría no tener el conocimiento necesario de la aplicación, u otras deficiencias (preparación o experiencia inadecuadas) que lleven a una productividad menor a la esperada.
- Circunstancias económicas o de negocios turbulentas podrían obligar a cancelar el contrato.
- Puede surgir una diversidad de gastos ocultos, por ejemplo, costos extras o trámites que no fueron necesarios en el sistema anterior.

De manera similar, los beneficios estimados podrían no materializarse. He aquí algunas razones posibles:

- Los usuarios operacionales podrían encontrar el uso del nuevo sistema más difícil de lo esperado, llevando a demoras e interrupciones. (Esto es de particular importancia si los beneficios del sistema se habían previsto en el sentido de una mayor productividad del usuario.)
- Podrían no ocurrir mejoras esperadas en la participación en el mercado. Tal vez el sistema no produzca más clientes, más pedidos, más negocios o más rendimientos.
- El sistema podría no comportarse como se espera; por ejemplo, no procesar tantas transacciones por segundo como se esperaba.
- El valor de la nueva información disponible gracias al sistema podría resultar no tener beneficios tangibles.

Para tratar estos riesgos, a menudo resulta ser buena idea considerar el peor escenario posible, el mejor y el esperado. Será mejor entre más preciso y realista sea esto: no tiene caso engañarse a sí mismo y a la administración con suposiciones innecesariamente optimistas respecto a los costos y beneficios. De manera similar, aunque nadie espera que la situación del peor caso ocurra, es importante que la administración entienda cuan malas se podrían poner las cosas.

Como nota final acerca del análisis de riesgos: muchas veces la administración sabe de muchos más riesgos que usted (por ejemplo, una potencial fusión entre su compañía y otra, que volverá inútil al sistema). Ellos puede evaluar dichos riesgos, y a menudo ni siquiera se los comunicarán; a usted lo necesitan para evaluar los riesgos técnicos del proyecto.

APENDICE

D AUDITORIAS E INSPECCIONES

D.1 INTRODUCCION

Este apéndice proporciona una visión global breve de una técnica conocida como *inspección* (*Walkthrough*). Puede encontrar útil hacer auditorías de las especificaciones que se desarrollaron durante el proyecto de análisis del sistema. Sin embargo, para usar este concepto necesita saber qué es una inspección, por qué se hacen, quién participa en ellas, y cuáles son los procedimientos.

Después de terminar de leer este apéndice puede necesitar información adicional. Dos referencias posibles son: *Structured Walkthroughs*, 3ª edición, de Edward Yourdon (Nueva York: YOURDON Press, 1985) y *Technical Inspections and Reviews*, de Daniel Freedman y Gerald Weinberg (Boston: Little, Brown, 1977).

D.2 ¿QUE ES UNA INSPECCION?

Como se usa el término en la industria de desarrollo de sistemas, una inspección es una *revisión de algún producto técnico hecha por un grupo de colegas*. Esto significa que la revisión implica a otros analistas que estén trabajando con usted, además de usuarios, programadores, diseñadores, personal de operaciones y otros que puedan estar involucrados en diversos aspectos del proyecto en el que está trabajando. Pero una inspección, bajo condiciones normales, no incluye a su jefe, al jefe del departamento, o al vicepresidente de la organización usuaria.

Obviamente, esta gente eminente querrá la oportunidad de revisar diversos aspectos del proyecto, incluyendo las especificaciones con las que está trabajando. Pero generalmente están menos involucrados con los detalles técnicos que los colegas de usted, y tal vez no tengan la posibilidad de ofrecer sugerencias detalladas. La política usualmente es un factor importante en tales revisiones de alto poder. Esto no significa que estas revisiones sean malas ni irrelevantes, sino simplemente que

son *diferentes* a las auditorías que se discuten en este apéndice. El peligro de permitir el acceso de la administración a un grupo de este tipo es que generalmente se interpondrá la política, o la inspección resultará ser una evaluación del rendimiento de *una persona*, más que la revisión técnica de un *producto*.

Note que puede haber distintos tipos de auditorías o inspecciones en un proyecto típico:

- Inspecciones de análisis
- Inspecciones de diseño
- Inspecciones de código
- Inspecciones de pruebas

Dado que el tema de este libro es el análisis de sistemas, nos concentraremos en las inspecciones de análisis. Para fines prácticos, esto significa que un grupo de analistas, junto con otras personas interesadas, se reúnen para revisar diagramas de flujo de datos, diagramas de entidad-relación, diagramas de transición de estados, entradas del diccionario de datos y especificaciones de procesos, es decir, todos los productos técnicos desarrollados por el analista.

D.3 ¿POR QUE HACEMOS INSPECCIONES?

La principal razón es encontrar errores tan rápida y económicamente como sea posible. Como se mencionó anteriormente en este libro, generalmente es mucho más barato encontrar y eliminar errores tan pronto como sea posible, en lugar de esperar hasta que el producto se haya terminado y enviado a la siguiente etapa de desarrollo.

Existen otras maneras de encontrar errores además de las inspecciones: la persona que produce el producto (por ejemplo, un DFD) puede revisarlo él mismo para tratar de encontrar errores. Pero el sentido común y años de experiencia en el campo del proceso de datos dictan que esto suele ser una manera bastante cara de hacer las cosas. Muchas veces las personas no pueden encontrar sus propios errores, sin importar cuánto estudien su trabajo. Esto sucede ya sea que se esté leyendo un documento textual para encontrar errores tipográficos, o un programa de computadora, o un DFD para encontrar errores. Un grupo de colegas interesados y conocedores a menudo pueden encontrarlos más rápido.

Otra manera de encontrar errores es usar una estación de trabajo para analista del tipo que se discute en el apéndice A; a grandes rasgos esto es análogo a usar un compilador para hallar errores de sintaxis en un programa, en lugar de revisar a mano el listado del programa. Si tiene una estación de trabajo de analista, por supuesto úsela para identificar todos los errores de sintaxis que sea capaz de encontrar. Pero así como un compilador no encuentra todos los errores en un programa de

computadora, (por ejemplo, no encuentra los errores lógicos o de tiempo de ejecución), de la misma manera una estación de trabajo de analista no pretende encontrar todos los errores en un conjunto de modelos de especificaciones. La inspección sigue siendo un complemento útil para las herramientas mecánicas disponibles.

De hecho, una de las cosas que es poco probable que una estación de trabajo para analista pueda hacer es comentar sobre el estilo de los productos: esto es algo que las personas pueden hacer muy bien. Así, cuando los revisores humanos examinan un DFD pueden plantear preguntas como:

- ¿Hay demasiadas burbujas en este diagrama?
- ¿Se escogieron de manera significativa los nombres de los procesos?
- ¿Se dibujó el diagrama de manera estética? ¿Es probable que el usuario realmente lea el diagrama, o es probable que se confunda debido a él?
- ¿Se agruparon los flujos de datos de manera significativa de un nivel a otro?
- ¿Se agruparon las actividades elementales de manera inteligente para formar burbujas de nivel superior?

Además, existen otros beneficios que normalmente obtienen las organizaciones con el enfoque de las inspecciones: capacitación y seguro. Un proceso de revisión por un grupo de colegas es una excelente manera de enseñar a los miembros novatos del equipo (al igual que a los antiguos y viejos) detalles sobre la aplicación, sobre el análisis de sistemas, o sobre la notación de diagramas de flujo de datos. Y puesto que todos los miembros del grupo de revisión llegan a familiarizarse más o menos con el producto (a menudo íntimamente), la inspección se convierte en un seguro contra una partida inesperada e inoportuna del productor; alguien debe poder continuar su labor.

El gran peligro de esto es que el productor puede no estar de acuerdo con los beneficios, y considerar que todo el proceso de inspección es una invasión de su privacidad. Si considera los DFD como de su propiedad (y no como un bien de la corporación) puede ser renuente a mostrarlos a otras personas. Si su sentido del estilo difiere del de sus colegas, puede haber discusiones violentas durante la inspección. Y si el productor se opone a la noción de capacitación y seguro, puede rechazar el concepto de inspección.

En general, las inspecciones tienen éxito en un ambiente donde la noción de equipo ya está aceptada y operando; en un proyecto típico, esto significa que cada individuo entiende que una falla seria o un error en su trabajo puede poner en peligro el éxito de todo el proyecto, lo cual significa que el potencial de errores de su trabajo es motivo válido de preocupación para los demás miembros del equipo. Para

más acerca de la noción de equipos, sobre todo los llamados "equipos sin ego", consulte el clásico libro *The Psychology of Computer Programming*, de Gerald Weinberg (Nueva york: Van Nostrand Reinhold, 1971).

D.4 CUANDO CONDUCIR UNA INSPECCION

Una inspección se puede llevar a cabo en casi cualquier momento durante el desarrolo de un producto técnico; es decir desde el momento en que por primera vez se le ocurre al productor, hasta el momento en que por fin está absolutamente convencido de que el producto está terminado y listo para ser entregado al consumidor (o a la siguiente etapa en el proceso de desarrollo). En general, es preferible hacer una inspección tan pronto como sea posible, pero no tanto como para que el producto esté incompleto o lleno de errores triviales que el autor pudo haber eliminado.

Tómese por ejemplo un diagrama de flujo de datos para ilustrar esto. El productor típicamente hará varias iteraciones sobre 1) discutir el área relevante del sistema con el usuario; 2) visualizar mentalmente un DFD; 3) trazar varias versiones incompletas de un DFD en servilletas o en sobres viejos; 4) dibujar una versión relativamente limpia en una hoja de papel; 5) meter los detalles del DFD en una estación de trabajo de analista del tipo discutido en el apéndice A; 6) eliminar errores de sintaxis con la herramienta disponible en la estación de trabajo; 7) imprimir la versión final del DFD con una impresora laser o un graficador y; 8) entregar el DFD final al jefe junto con una exclamación triunfante de que se concluyó la tarea antes de lo previsto.

En este caso es demasiado pronto como para tener una inspección que sirva de algo en las etapas 1), 2) y 3); la inspección se puede hacer de manera efectiva en las etapas 4), 5) o 6); y en las etapas 7) y 8) es demasiado tarde. Precisamente cuándo hay que hacer una inspección dependerá de cuánto apoyo automatizado se tenga, de qué tan rápido se pueda tener dicho apoyo (es decir, ¿el analista tiene que esperar cuatro días para tener acceso a la estación de trabajo de analista que está compartiendo con otros 27 analistas?), y de cuánto cuesta usarlo.

La principal razón para evitar una inspección en una etapa tardía es que el productor habrá invertido demasiado de su ego en el producto como para querer hacer cambios, a menos de que se trate de la corrección de errores gruesos (e incluso a veces ni esos). También, el productor puede haber dedicado demasiado tiempo a la eliminación de errores del producto, siendo que el equipo de revisión lo pudo haber hecho de manera más rápida y económica si lo hubiera visto antes.

Y, finalmente, se debe recordar la psicología de las personas mismas que revisan: invierten su propio tiempo en la búsqueda de los errores de otro, y sentirán esto hasta cierto grado, independientemente de lo altruistas que digan ser. Debido a ello no se les debe mostrar un producto incompleto y mal hecho; pero tampoco se les debe mostrar un producto terminado, congelado y perfecto.

Si va a invertir una hora de su tiempo en la revisión del DFD de un colega, es bueno saber que está haciendo algo útil al encontrar un error que el productor mismo no encontró. Si, por otro lado, pasa media hora viendo un proyecto sin hallar nada qué comentar, es natural que vea el esfuerzo como un desperdicio de tiempo y la próxima vez no esté tan dispuesto a participar en una inspección.

D.5 ROLES EN UNA INSPECCION

Muchas organizaciones hacen inspecciones sin mayor formalismo o preparación que los recién descritos. Pero muchos han encontrado que conviene introducir algún formalismo o estructura en la revisión; de aquí proviene el término común "inspección estructurada". Una característica común de una inspección estructurada es el conjunto de roles formales que juegan los que revisan. Cada uno juega distintos papeles en distintas inspecciones; y en algunos casos uno puede desempeñar más de un papel.

He aquí algunos de los roles comunes que se encuentran en una inspección:

- El *presentador.* Es la persona que explica al grupo de revisión lo que el producto hace, qué suposiciones se hicieron cuando se creó, etc. En muchos casos, el presentador es el productor, pero no siempre. Algunas organizaciones sienten que si el productor presenta su propio producto, entonces 1) el producto puede ser tan críptico u oscuro que no se pueda sostener por sí solo no estando el productor presente para explicarlo y, 2) el productor pudiera lavarle el cerebro de manera sutil (y, es de suponerse, de manera inocente) a su público, induciéndoles los mismos errores, descuidos y errores de omisión y comisión que él mismo cometió.

- El *moderador.* Es la persona que organiza y conduce la reunión. Su propósito es que la discusión continúe de una manera ordenada y constructiva, para evitar salidas por la tangente en las discusiones, además de críticas al presentador. Por razones obvias, existe la tentación de permitir al administrador del proyecto hacer este papel; pero por las razones descritas anteriormente en este apéndice, su presencia en el grupo a menudo cambia el tono de la revisión de una manera muy negativa.

- El *secretario.* Es quien toma nota por escrito de los eventos importantes de la revisión. Además de cosas triviales, como la fecha de realización de la auditoría, de quién era el producto que se estaba revisando, y quiénes estuvieron presentes en la revisión. El secretario toma nota de las preguntas técnicas de importancia que hayan surgido, los errores que se encontraron, y las sugerencias de mejora o modificación que hicieron los miembros del grupo de revisión.

- *Oráculo de mantenimiento.* Es un revisor cuya principal preocupación es el mantenimiento a largo plazo del producto. Generalmente se preocupará de que éste no sea demasiado idiosincrático y que esté lo suficientemente bien documentado. Es probable que juegue un papel mayor en las revisiones de diseño y de código que en las de análisis de sistemas.

- El *verificador de estándares.* El rol de esta persona es obvio: asegurar la consistencia del producto con los estándares globales que adoptaron el proyecto y/o la organización. En ocasiones su papel principal es aconsejar al productor (y a otros miembros del equipo) sobre si el el grupo de control de calidad finalmente juzgará aceptable el producto (en términos de la adhesión a los estándares).

Existen dos comentarios finales respecto a estos roles: primero, tenga en cuenta que pueden cambiar de una revisión a otra. Segundo, si el usuario participa activamente en el proyecto, recuerde incluirlo en uno de ellos.

D. 6 PROCEDIMIENTOS DE REVISION

Como se indicó en la sección anterior, las revisiones exitosas usualmente se caracterizan por un conjunto de roles y procedimientos formales. Estos procedimientos varían de una organización a otra, pero la siguiente es una lista típica:

1. Programe la revisión uno o dos días antes y distribuya material apropiado a los miembros del grupo. Si la auditoría se programa sin suficiente anticipación, los que revisan no tendrán oportunidad de estudiar el producto con anticipación.

2. Asegúrese que quienes revisan hayan realmente dedicado un tiempo a la revisión del producto. Una manera fácil de hacerlo es pedirle a cada uno que llegue a la auditoría aportando por lo menos un comentario positivo y por lo menos uno negativo sobre el producto. Aquí el peligro radica en que los que revisan pueden estar tan ocupados o tan carentes de interés por el producto que no hagan el trabajo por adelantado y simplemente se sienten y se queden callados durante toda la auditoría sin aportar nada.

3. Pídale al presentador que haga una presentación breve del producto. Esto a menudo se hace proyectando acetatos, empleando rotafolios, etc. Aquí es donde literalmente el grupo hace la revisión del producto.

4. Solicite comentarios a los que revisan. Esto normalmente lo hace el moderador, quien puede decidir dar la vuelta a todo el salón, pidiendo a cada uno que señale un error o que haga un comentario acerca del producto.

5. Asegúrese que las cuestiones se *presenten* pero no se *resuelvan* durante la auditoría. Esto sobre todo importa si se detecta un error no trivial; deje que el productor se las arregle para resolverlo en su propio tiempo, en lugar de permitir que se desarrolle una sesión no estructurada de ideas. Este procedimiento también es importante cuando surgen comentarios

acerca del estilo: el productor puede estar en desacuerdo, y es mejor que los reconsidere después de la revisión (o que hable aparte con quien hizo la sugerencia sobre el estilo).

6. Cuide que la auditoría sea breve. No más de una hora. No se puede esperar que la gente mantenga un alto nivel de concentración durante más de una hora; se empezarán a distraer, y existe un gran peligro de que la revisión degenere hasta convertirse en pelea.

7. Haga una votación acerca de los resultados de la auditoría. Las recomendaciones típicas que los revisores hacen son: 1) "Creemos que el producto está bien tal como está", o 2) "Creemos que se deben corregir algunos errores y que se deben atender algunos detalles de estilo, pero confiamos en que el productor hará los cambios apropiados sin necesidad de otra revisión", y "Hemos encontrado tantos errores y detalles respecto al estilo, que quisiéramos hacer otra revisión después de que el productor haya hecho las modificaciones apropiadas". Dependiendo de la naturaleza del equipo y de la forma en que las personas asumen la responsabilidad de su trabajo en la organización, este voto puede ser limitante o bien representar simplemente una sugerencia opcional hecha por los que revisan.

D.7 RESUMEN

Aunque el enfoque de la auditoría es un proceso sencillo y directo de revisión, no se usa tanto como pudiera pensarse. Una de las razones de esto es el aumento del tiempo requerido para hacerlas: puede tomar hasta un 5 o un 10 por ciento del tiempo total del proyecto. Por otro lado, muchas organizaciones que las han usado han reportado reducciones dramáticas en cuanto al número de errores que quedan sin detectarse.

Posiblemente la razón más importante de no usar auditorías sea que algunos programadores y analistas siguen considerando a sus programas y diagramas de flujo de datos como propiedad personal, en lugar de considerarlos como un bien de la organización. Por ello, prefieren no mostrar a otros su trabajo y se oponen marcadamente a cualquier crítica o sugerencia de mejora. Este es un punto de vista peligroso; cada vez más organizaciones se dan cuenta de que deben realizar algún tipo de revisión por grupos de colegas si desean mejorar la calidad de los sistemas que producen.

APENDICE

E TECNICAS DE ENTREVISTA Y RECOLECCION DE DATOS

E.1 INTRODUCCION

Este apéndice discute reglas para las *entrevistas* que usted realizará durante la fase de análisis de un proyecto de desarrollo de sistemas. Probablemente entrevistará a usuarios, administradores, auditores, programadores que mantienen los sistemas de cómputo existentes, y varios tipos más de personas.

¿Por qué se hacen encuestas durante el análisis de sistemas? Las razones son las siguientes:

- Necesitamos reunir información sobre el comportamiento de un sistema actual o de los requerimientos del nuevo, a partir de las personas que lo tienen.
- Necesitamos verificar lo que como analistas entendimos del comportamiento del sistema actual o de los requerimientos del nuevo. Este entendimiento probablemente se adquirió mediante entrevistas anteriores, junto con información reunida en forma independiente.
- Necesitamos reunir información acerca del sistema o sistemas actuales para poder realizar cálculos de costo-beneficio (vea el Apéndice C para más información al respecto).

Este Apéndice cubre los siguientes tópicos referentes al proceso de la entrevista:

- Tipos de entrevista

- Problemas fundamentales de los que hay que preocuparse en la entrevista
- Reglas generales para hacer entrevistas

E.2 TIPOS DE ENTREVISTAS

Tal vez la forma de entrevista más común sea la de un encuentro en vivo, frente a frente, entre usted (posiblemente acompañado de uno o más colegas analistas de los mismos proyectos) y uno o más sujetos (entrevistados). Típicamente, uno de los entrevistadores tomará notas con papel y lápiz. Menos comúnmente, la entrevista se grabará o una secretaria tomará notas formales. Durante todo este apéndice, supondré que su entrevista será de naturaleza general, pero no haré suposiciones acerca de grabadoras o estenógrafos.

Tome en cuenta que el tipo de información obtenida durante una entrevista también puede obtenerse por otros medios, por ejemplo, pidiendo al sujeto o sujetos de la entrevista que respondan a un cuestionario formal por escrito, o pidiéndoles que le den un escrito con una descripción de los requerimientos de su nuevo sistema. También es posible que las entrevistas se puedan complementar con la presencia de algunos especialistas (quienes incluso pudieran llevar a cabo la entrevista mientras el analista juega un papel pasivo), tales como expertos en la industria, psicólogos del comportamiento y negociadores del sindicato. Y, por último, debe tener en cuenta que se puede usar otro medio de captura de datos (por ejemplo, videograbadora) para grabar el contenido de una entrevista.

Durante los años 80 se hizo popular una manera especializada de hacer entrevistas en algunas organizaciones de administración de sistemas de información; se conoce como JAD (siglas en inglés de Desarrollo Conjunto de Aplicaciones), o diseño acelerado, o análisis de equipo, o por varios términos más. Consiste en un proceso acelerado de entrevista y de recolección de datos en el cual todos los usuarios clave y el personal de análisis de sistemas se reúnen en una sola junta intensiva (que puede durar desde un día hasta una semana) para documentar los requerimientos del usuario. Usualmente un especialista preparado supervisa la junta y actúa como mediador para promover una mejor comunicación entre los analistas y los usuarios.

Aunque todas estas variantes se han usado, de hecho, son relativamente raras y no se discutirán más a fondo en este Apéndice. La entrevista más común es todavía el enfrentamiento uno a uno entre un analista y un usuario final.

E.3 PROBLEMAS FUNDAMENTALES DE LOS QUE HAY QUE PREOCUPARSE

A primera vista, pudiera parecer que el proceso de entrevistar a un usuario es sencillo y directo. Después de todo, usted es una persona inteligente y capaz de comunicarse, y el usuario lo es también. Ambos son personas racionales y ambos desean lograr el mismo objetivo: transferir información sobre un nuevo sistema propuesto de la mente del usuario a la suya. ¿Cuál es el problema?

De hecho, existen *muchos* problemas que pueden surgir. En muchos proyectos de alta tecnología, se ha observado que los problemas más difíciles no involucran hardware ni software sino "genteware". Estos asuntos en el análisis de sistemas se aprecian más por lo regular en la situación de la entrevista: es aquí donde el analista y el usuario verdaderamente entran en contacto. Los problemas más comunes de los que se debe cuidar son los siguientes:

- *Entrevistar a las personas equivocadas en el momento equivocado.* Es muy fácil, debido a los problemas de organización y de política, que se encuentre hablando con la persona que oficialmente es la experta en política del usuario, y que resulta no saber nada de los verdaderos requerimientos del sistema; también es posible perder la oportunidad de hablar con el usuario desconocido que realmente *sí* los maneja. Aun si encuentra a la persona correcta, puede encontrar que está tratando de hacerle la entrevista en un momento en el que no está disponible o en el que está sometido a muchas presiones y emergencias.
- *Hacer las preguntas equivocadas y obtener las respuestas equivocadas.* El análisis de sistemas es, como le gusta decir a Tom DeMarco, una forma de comunicación entre extraterrestres. Los usuarios y los analistas tienen distinto vocabulario, distinta experiencia, y a menudo distintos conjuntos de suposiciones, percepciones, valores y prioridades. Por ello, es fácil que le haga una pregunta razonable al usuario acerca de los requerimientos de su sistema, y que éste le malentienda por completo, sin que ninguno se percate de ello. Y es fácil que el usuario le dé información respecto a los requerimientos y que usted malentienda dicha información, nuevamente sin que ninguno de los dos se percate. Las herramientas de modelado que se presentaron anteriormente en este libro son un intento de proporcionar un lenguaje común, no ambiguo, a manera de poder disminuir estos malos entendidos. Pero las entrevistas se llevan a cabo en gran medida en un lenguaje común hablado (inglés, español, francés, etc.), así que el problema es verdadero. Es por esto que es tan importante programar entrevistas de seguimiento para verificar que ambas partes hayan entendido tanto las preguntas como las respuestas.
- *Crear fricciones entre ambas partes.* Como veremos en la Sección E.6, existen razones por las cuales un usuario se puede sentir incómodo o incluso antagonista en una entrevista con un analista de sistemas (por ejemplo, porque siente que el propósito del nuevo sistema que el analista está especificando le va a quitar su empleo). Y el analista puede sentirse molesto por la forma en la que el usuario está respondiendo (por ejemplo,

puede sentir que el usuario lo está insultando al sugerir que es demasiado joven e inexperto para estar dando sugerencias sobre los requerimientos del nuevo sistema). En cualquier caso, pueden surgir fricciones entre las partes, haciendo más difícil la comunicación.

No existe alguna manera mágica de garantizar que estos problemas no ocurran; son el resultado de las interacciones entre personas, y cada una de tales interacciones es única. Sin embargo, las siguientes sugerencias le pueden ayudar a reducir la posibilidad de que surjan estos problemas; aparte de eso sólo queda depender de la práctica y mejorar con cada entrevista.

E.4 REGLAS PARA HACER ENTREVISTAS

Las siguientes reglas le pueden ser útiles para hacer una entrevista exitosa con su usuario.

E.4.1 Desarrolle un plan global de la entrevista

Antes de comenzar, es muy importante que se entere de a quién debe entrevistar. De otro modo, desperdiciará el tiempo de todos, y creará problemas políticos enormes, al hablar con la persona equivocada de cosas equivocadas.

Esto requiere que obtenga un organigrama que muestre los distintos puestos. Si no se ha publicado uno, encuentre a alguien que sepa cómo trabaja la organización y pida ayuda. Si sí existe uno, asegúrese que esté actualizado; a menudo las organizaciones cambian de manera mucho más rápida que el ciclo anual de las publicaciones en las que se producen los organigramas.

Incluso un organigrama no le dirá necesariamente con quién necesita hablar; en ocasiones la persona de más conocimientos en lo que se refiere a algún aspecto del sistema será un oficinista o algún administrador que ni siquiera aparece allí. Como se discutió en el Capítulo 3, muchas veces existen tres niveles de usuarios en una organización grande y compleja: el verdadero usuario, el usuario supervisor operacional, y el usuario ejecutivo supervisor. A menudo resulta importante hablar con todos ellos.

También es importante en muchos casos hablar con ellos en la secuencia apropiada y con la combinación correcta. Es decir, puede encontrarse entrevistando a Marta, quien dice: "Pues, desde luego, Jorge me proporciona todos mis datos de entrada, él le puede decir cómo es. Y entonces yo..." En tal caso resulta útil hablar primero con Jorge y luego con Marta. O puede estar entrevistando a Francisco, quien dice: "Pues, en realidad, Susana y yo trabajamos juntos en esto; ella hace una parte y yo el resto..." En este caso, obviamente sería más útil hablar con ambos a la vez. A veces puede distinguir qué usuarios se necesita entrevistar y en qué secuencia, partiendo tan sólo de su conocimiento general de la organización; a veces los usuarios mismos le dirán en cuanto sepan que los va a entrevistar.

E.4.2 Asegúrese de contar con aprobación para hablar con los usuarios

En algunas organizaciones informales no hay restricciones en cuanto a quién entreviste o cuándo. Pero en una organización grande esto es poco usual; es políticamente peligroso andar por toda la organización usuaria haciendo entrevistas sin alguna aprobación anterior.

En la mayoría de los casos, esta aprobación vendrá del administrador de un área usuaria (un departamento, división o grupo) o bien de algún representante de los usuarios previamente designado, vinculado con el proyecto de desarrollo de sistemas. En cualquier caso, los usuarios tienen motivos legítimos para desear aprobar, por adelantado, a quién se vaya a entrevistar:

- Pueden sentir que algunos usuarios no son capaces de entender o describir bien los requerimientos del sistema.
- Pueden preocuparse de que algunos de sus usuarios de nivel operacional sean "renegados", que den requerimientos falsos (o en cualquier caso, requerimientos que la administración no apruebe).
- Pueden preccuparse de que las entrevistas interfieran con las labores normales de los usuarios. Por ello, querrán programarlas.
- Pueden preocuparse de que las entrevistas se perciban como el comienzo de un esfuerzo por reemplazar a los usuarios humanos con un sistema computarizado, causando despidos, etc.
- Pueden pensar que ellos mismos (los administradores) saben mucho más sobre los requerimientos del sistema que cualquier otra persona). Por tanto podrían no querer que usted hable con usuarios de nivel operacional.
- Puede existir algún conflicto político, a un nivel mucho más alto de la administración, entre la organización usuaria y su organización de desarrollo de sistemas. Así, el administrador puede no tener verdadera objeción a sus entrevistas, pero al evitarlas puede mandar un mensaje político al jefe del jefe de su jefe.

Por todas estas razones, es buena idea lograr la aprobación por adelantado. En muchos casos, basta la aprobación verbal; si la organización es terriblemente burocrática o paranoica, puede ser que incluso la necesite por escrito. Esto también significa, por cierto, que debe estar al tanto de la política de la organización si siente que necesita hablar con un usuario (típicamente un usuario de nivel operacional) a quien se le haya indicado no entrevistar. Puede desear programar alguna reunión clandestina fuera del lugar de trabajo, pero usualmente resulta menos peligroso pasar la solicitud a sus superiores para que suba por toda la cadena jerárquica de la

organización, para finalmente ser pasada a los niveles superiores de la organización usuaria, para luego descender por toda esa cadena jerárquica.[1]

E.4.3 Planee la entrevista para usar de manera efectiva su tiempo

El motivo principal de esta sugerencia es que se dé cuenta de que está quitándole tiempo al usuario y que él (o su jefe) pueden sentir que se está *desperdiciando*. Por eso es importante que planee y se prepare lo más posible con anticipación para que pueda hacer uso efectivo de la entrevista.

Lo primero que hay que hacer es asegurarse de que el usuario conozca el tema de la entrevista. En algunos casos esto se puede hacer por teléfono; en otros, puede ser apropiado escribir una lista de las preguntas que vaya a hacer, o los tópicos que va a abordar, o los DFD que quiere revisar, y mandársela al usuario con uno o dos días de anticipación. Si no lo puede hacer, es un indicio de que en realidad no está preparado para la entrevista. Y si el usuario no ha leído el material que le mandó, significa o bien que 1) está muy ocupado, 2) que no le interesa, 3) que siente hostilidad hacia todo el concepto de la entrevista o, 4) que no es capaz de entender sus preguntas.

Además: reúna toda la información posible antes de la entrevista. Si existen formas o reportes relacionados con la discusión, generalmente puede obtenerlos por adelantado. Si existen otros documentos que describen el nuevo sistema o el anterior, asegúrese de haberlos obtenido y estudiado antes de la entrevista.

Si preparó sus preguntas por adelantado, debe poder restringir la entrevista a una hora o menos. Esto es importante, pues no sólo resulta generalmente que el usuario no puede dedicarle más que una hora o dos cada vez, sino también (como además lo señalé en el Apéndice D) las personas generalmente no se pueden enfocar y concentrar completamente (sobre todo si están contemplando diagramas extraños) por más de una hora. Esto significa que, desde luego, deberá organizar su entrevista para que cubra un temario relativamente limitado, concentrándose típicamente en una parte pequeña del sistema. También puede significar que debe programar varias entrevistas con el mismo usuario para cubrir por completo el área con la que se encuentra involucrado.

Finalmente, programe una reunión de seguimiento para revisar el material que recopiló. Generalmente, deseará retirarse a su escritorio con toda la información que reunió de la entrevista, asumir su papel de analista, y ponerse a trabajar. Puede ser que se necesite dibujar DFD, o escribir entradas del diccionario de datos; tal vez se ocupe hacer cálculos de costo-beneficio; puede ser necesario correlacionar la información que obtuvo con los datos obtenidos de otras entrevistas, etc. De cualquier forma, los datos de dicha entrevista se manipularán, documentarán, analizarán y se *transformarán a una forma que posiblemente el usuario jamás haya visto antes.* Por ello necesita programar una entrevista de seguimiento para verificar: 1) que no haya entendido mal lo que el usuario le dijo, 2) que éste no haya cambiado de opinión desde la entrevista[2], y 3) que él entienda la notación o representación gráfica de dicha información.

1 Todo esto involucra políticas organizativas que rebasan el alcance de este libro. Para mayor información lea alguno de los textos estándar de administración y de teoría organizacional, o consulte el agradable libro de Robert Block, *The Politics of Projects* (Nueva York: YOURDON Press, 1981).

E.4.4 Use herramientas automatizadas cuando sea apropiado, pero no abuse

Durante la entrevista, puede resultarle conveniente usar herramientas de prototipos, sobre todo si su propósito es discutir el punto de vista del usuario en cuanto a la interfase humano-máquina. De manera similar, si está revisando un diagrama de flujo de datos y discutiendo posibles cambios, puede resultarle conveniente usar una de las herramientas CASE que se discuten en el Apéndice A.

Recuerde, sin embargo, que el propósito de dichas herramientas es *facilitar* la entrevista, no estorbarle; deben permitir a usted y al usuario explorar alternativas y hacer cambios rápida y fácilmente; puede serle útil para anotar lo que entiende de los requerimientos del usuario en el acto y corregir de inmediato errores que haya podido cometer.

Sin embargo, si la tecnología estorba, debe excluirse de la entrevista. Si se necesita que el usuario salga de su ambiente normal de trabajo (por ejemplo, que vaya a otro edificio, o al cuarto de computadoras), puede ser que vea la herramienta como algo molesto. Si no está familiarizado con la tecnología de las computadoras y se le pide que use una herramienta, es posible que la rechace. Y si usted no está familiarizado con la herramienta, (o si la herramienta es lenta, susceptible de cometer errores, o limitada en su uso) entonces será un gran estorbo para la entrevista. En cualquiera de estos casos, probablemente sea aconsejable usar la herramienta *después* de la entrevista; luego puede mostrar al usuario los resultados obtenidos con la herramienta sin causar problemas innecesarios.

E.4.5 Trate de juzgar qué información le interesa más al usuario

Si tiene que desarrollar un modelo completo para alguna porción de un sistema, tarde o temprano tendrá que determinar entradas, salidas, funciones, características dependientes del tiempo y memoria del sistema. El orden en el cual obtenga la información realmente no importa mucho o, por lo menos, probablemente a usted no le importe mucho.

2 ¿Por qué razón podrá el usuario cambiar de opinión de una entrevista a la siguiente? Normalmente porque la entrevista hace que dirija su atención a algo que tan sólo ha considerado "desde lejos" hasta ahora. Las preguntas durante la entrevista pueden hacer que vea sus requerimientos en forma diferente.

Pero puede importarle mucho al usuario, y debe permitirle empezar la entrevista donde él quiera. Algunos usuarios desearán comenzar por las salidas, es decir, los reportes o datos que quieren que el sistema produzca (de hecho, puede ser que ni siquiera sepan qué entradas se ocupan para producir las salidas deseadas). Otros usuarios pueden estar más interesados en las entradas o en los detalles de una transformación funcional. Otros querrán hablar acerca de los detalles de los datos de algún almacén. Sea lo que sea, trate de visualizar los requerimientos del sistema lo mejor posible desde el punto de vista de ellos, y tenga dicha perspectiva en mente cuando les haga las preguntas necesarias durante la entrevista.

E.4.6 Use un estilo apropiado de entrevista

Como lo señala William Davis (Davis, 1983):

> La actitud que sobre la entrevista tome será determinante en el éxito o fracaso. Una entrevista no es un concurso. Evite ataques; evite uso excesivo de vocabulario técnico; haga una entrevista, no un interrogatorio. Hable con las personas, no contra ellas, y háblelas poniéndolas en su mismo nivel, no arriba o abajo. Una entrevista no es un juicio. Haga preguntas de sondeo pero no interrogue. Recuerde que el entrevistado es el experto, y que usted es el que busca respuestas. Finalmente, haga lo que haga, evite atacar la credibilidad de la otra persona. No diga: "Tal persona me dijo otra cosa", o "No sabe de qué está hablando."

Hacer preguntas de sondeo no siempre es fácil; dependiendo de la personalidad del entrevistado y el tema de la entrevista, puede requerir una variedad de estilos para lograr la información deseada. He aquí algunos estilos que pueden resultar útiles:

- *Relaciones:* Pídale al usuario que explique la relación entre lo que se está discutiendo y otras partes del sistema. Si el usuario está hablando acerca de un objeto (por ejemplo, un cliente), pídale que explique su relación con otros objetos; si está describiendo una función (es decir, una burbuja del DFD), pídale que le explique su relación con otras funciones. Esto no sólo le ayudará a descubrir más sobre lo que se está discutiendo, sino también a descubrir interfases (por ejemplo, flujos de datos de una burbuja a otra en el DFD) y relaciones formales.

- *Puntos de vista alternativos:* Pídale al usuario que describa el punto de vista de otros usuarios sobre lo que se está discutiendo. Por ejemplo, pregúntele lo que opina su jefe acerca de una burbuja del DFD, o un tipo de objeto en el DER; o pregúntele lo que opinan sus subordinados.

- *Sondeo:* Pida al usuario una descripción narrativa e informal de lo que le interesa saber. Puede decir: "Háblame de la manera en que calcula los costos de envío". O si le está hablando acerca de un tipo de objeto del DER, puede decirle: "Háblame del cliente. ¿Qué sabe (o necesita saber) acerca de un cliente?"

- *Dependencias:* Pregunte al usuario si lo que se está discutiendo depende de alguna otra cosa para su existencia. Esto es particularmente útil cuando se discuten tipos de objetos posibles y relaciones posibles dentro del DER. Por ejemplo, en un sistema de ingreso de pedidos, puede preguntar al usuario si es posible tener un pedido (si eso es lo que está discutiendo en ese momento) sin tener cliente.

- *Repetición:* Dígale al usuario lo que cree haberle oído decir; use sus propias palabras en lugar de las de él y pídale que lo confirme. Así, puede decir: "A ver si le entendí: cuando una pieza entra al sistema, siempre le hace un proceso y le manda un mensaje de status al departamento de auditoría."

E.5 POSIBLES FORMAS DE RESISTENCIA A SER ENTREVISTADO

Como se mencionó antes, debe estar preparado para el hecho de que algunos usuarios se opongan a la idea misma de una entrevista; ésta es una de las razones para asegurar que su administrador o alguien de autoridad en su departamento esté enterado de la entrevista y la haya aprobado. Algunas de las objeciones más comunes (y sus posibles respuestas) se citan a continuación:

- *Está ocupando demasiado de mi tiempo.* La respuesta a esto es que le diga que lo comprende, y se disculpe por ello, pero le comunique que la ha preparado de antemano lo más posible, con lo cual espera reducir la entrevista al mínimo. Esto requiere, desde luego, que sea puntual, que no se salga del objetivo, y que termine cuando lo prometió.

- *Está amenazando mi empleo.* Esta es a menudo una reacción muy emocional, y puede estar o no bien fundada. Aunque se le pueda ocurrir un gran número de respuestas, recuerde que usted no es el administrador de esta persona, y que no tiene autoridad para asegurarle que su empleo no peligra, ni para prevenirle de que sí. Puede tratar de deslindar la responsabilidad diciendo: "No tengo nada qué ver con esto; sólo estoy documentando los requerimientos del sistema por orden de la administración", pero el usuario no aceptará eso. Lo verá como el "experto en eficiencia" cuya labor es aconsejar a la administración cómo eliminar su empleo mediante la computadora. La solución de este problema, si se presenta, es comunicarlo a los niveles superiores de la administración y obtener el veredicto oficial de *ellos*, en persona o por escrito si es posible.

- *No conoce nuestro negocio, así que ¿cómo propone decirnos lo que el nuevo sistema debe ser?* La respuesta a esto es: "Tiene razón: por eso lo estoy entrevistando para averiguar lo que usted opina sobre los reque-

rimientos". Por otro lado, si usted es un analista astuto, probablemente sugiera varias maneras de "mejorar" las cosas (particularmente si todo, o parte, del trabajo que el usuario hace actualmente es resultado de una implantación vieja e ineficiente del sistema); por tanto, tal vez este tipo de comentario sea inevitable. Sin embargo, lo adecuado es continuar siendo lo más humilde posible, y constantemente reconocer lo experto que el usuario es en su área de trabajo, a la vez que continúa pidiéndole que tenga la amabilidad de explicarle (contribuyendo de esta manera a su entendimiento) por qué no funciona su idea.

- *Está tratando de cambiar la forma en que hacemos las cosas aquí.* Esta es una variante del comentario anterior. Debe mostrarle que aunque proponga algunos cambios (radicales) en la *implantación* de su sistema actual, no está tratando de cambiar sus características *esenciales,* excepto en las áreas en las que ellos mismos lo solicitaron. Sin embargo, tenga en mente que algunas de las características de la implantación del sistema actual podrían tener que conservarse, porque tiene alguna interfase con otros sistemas externos que requieren entradas o salidas de forma predefinida.

- *No queremos este sistema.* Esta es una variante de la queja: "Me está quitando el empleo." La verdadera respuesta a esto es que está ahí, haciendo la entrevista, porque el administrador de los usuarios quiere el nuevo sistema. No es asunto suyo tratar de convencer al usuario operacional que deben querer el sistema (no importa lo grandioso que usted opine que es); hacer esto es asumir la responsabilidad usted mismo, y no le corresponde.

- *¿Por qué está desperdiciando nuestro tiempo con esta entrevista?* "Sabemos lo que queremos, y si fuera competente, lo entendería de inmediato. ¿Por qué no pone manos a la obra, y hace de una vez el sistema?". Esta es una queja difícil de tratar, porque tiene que ver con el hecho básico de que los usuarios y los analistas hablan lenguajes distintos; si el usuario no reconoce este hecho, le avecinan grandes problemas. Una solución posible es hacer una analogía. Pregúntele al usuario si le permitiría a un arquitecto comenzar a construir su casa sin discusiones detalladas y planos, seguidos de una amplia comunicación durante la construcción. Pregúntele si estaría dispuesto a decirle al arquitecto "Constrúyame una casita bonita de tres recámaras. Sabe a lo que me refiero, ¿verdad?" Sin embargo, tenga en cuenta que con la amplia disponibilidad de lenguajes de cuarta generación y computadoras personales, el usuario puede sentir que él mismo puede construir el sistema; tal vez el haber tenido éxito con proyectos sencillos (por ejemplo, hojas de cálculo) le haya dado la impresión de que todos los sistemas son fáciles de hacer. Esto puede explicar el porqué de su impaciencia con usted.

E.6 OTROS PROBLEMAS DE LOS QUE HAY QUE CUIDARSE

Las reglas que se dieron anteriormente previenen sobre muchos problemas políticos a los que se puede enfrentar durante una entrevista, y las muchas razones por las cuales un usuario podría resistirse a ser entrevistado. Pero aún quedan algunos otros problemas a los que se debe anticipar:

- *Una discusión que se enfoca más a cuestiones de implantación que a cuestiones de requerimientos.* Esto suele suceder cuando el usuario dice: "Así es como me gustaría que construyera el sistema...". Sucede bastante a menudo cuando el usuario piensa en términos de la implantación de su sistema actual; y puede suceder si el usuario está familiarizado con la tecnología de las computadoras (por ejemplo, si tiene su propia PC o es un ex- programador). Recuerde que no es su tarea en una entrevista de análisis describir las características de implantación del sistema, a menos que sean tan importantes que realmente tengan cabida dentro del modelo de implantación del usuario que se discutió en el Capítulo 21.

- *Confundir síntomas con problemas.* Este es un problema que se presenta en muchos campos, no sólo en el de las computadoras. Imagine un paciente hablando a su médico y diciendo: "Doctor: mi problema consiste en siento la cara realmente caliente. ¿Me puede resolver el problema?" Es de suponer que se trata de un síntoma, es decir, alguna especie de fiebre, que indica un problema médico. Lo que hay que hacer es darse cuenta de que se trata de un síntoma, no del problema mismo, y luego hay que encontrar el verdadero problema. Lo mismo sucede una y otra vez en las entrevistas de análisis de sistemas. Sin embargo, en gran medida depende de dónde se coloca la *frontera* en el diagrama de contexto: el que la queja de un usuario sea síntoma o problema depende de si está relacionada con algo que esté dentro o fuera de la frontera del sistema. Por tanto, ponga especial atención al desarrollo del modelo ambiental; esto se discute con detalle en el Capítulo 18.

- *El usuario podría no ser capaz de decir qué quiere que el sistema haga, o podría cambiar de opinión.* Este es un problema común, y el analista debe estar preparado para enfrentarlo. Entre más extremo sea el problema, más importante se vuelve el prototipo. Para más información sobre esto, consulte el Capítulo 5.

- *Hay desacuerdo entre colegas, subordinados y administradores.* Desafortunadamente, esto pone al analista en el papel de negociador entre las diversas partes contendientes. Como analista no puede abdicar y decir "Cuando se hayan puesto de acuerdo sobre lo que quieren, regresen y díganmelo." En lugar de ello, debe actuar como mediador y tratar de reunir a las partes para trabajar, para llegar a un consenso. Desafortunadamen-

te, esto involucra habilidades y procedimientos que rebasan el alcance de este libro.

E.7 FORMAS ALTERNATIVAS DE RECOPILACION DE DATOS

Las entrevistas no son la única manera de recopilar información sobre los requerimientos de un sistema. De hecho, entre más información pueda reunir de otras fuentes, probablemente sean más productivas sus entrevistas. Entre las alternativas a las entrevistas tenemos:

- *Cuestionarios:* Puede enviar cuestionarios por escrito a los usuarios de su organización, a las personas (u organizaciones) que interactúan con el sistema, a los administradores que aprobaron el proyecto, y a otros.
- *Presentaciones de proveedores.* Los proveedores de software y hardware pueden haber desarrollado ya sistemas para la aplicación que le interesa. El pedirles que hagan una presentación de las características de su sistema puede no sólo ayudarle a determinar si el de ellos ofrece una buena solución, sino también a señalar requerimientos de funciones y datos almacenados que de otra manera hubieran pasado desapercibidos.
- *Visitas a otras instalaciones:* Busque organizaciones que tengan la misma línea de negocios o que tengan sistemas similares al que pretende construir. Organice una visita a la instalación para obtener información de primera mano sobre las características y capacidades del sistema.
- *Recolección de datos:* Busque formas, reportes, manuales, procedimientos escritos, despliegues de pantallas y listados de programas de la organización usuaria. Sin embargo, tenga en mente que normalmente se asocian con la *implantación actual* del sistema. Como se discutió en el Capítulo 18, esto usualmente incluirá información redundante, contradictoria u obsoleta. No obstante, suele ser un buen punto de partida para familiarizarse con el terreno antes de realizar entrevistas personales con el usuario.
- *Investigación externa:* Si está construyendo un sistema para una aplicación nueva, sobre la cual el usuario no tiene experiencia para poderle comunicar sus requerimientos, entonces debe buscar información de sociedades profesionales (por ejemplo, la ACM, la IEEE, o la DPMA), o de revistas técnicas, libros de texto o artículos de investigación.

E.8 RESUMEN

Las habilidades de comunicación, la diplomacia y otras cuestiones humanas involucradas en la entrevista no son fáciles de exponer en un libro. Es algo que tiene que aprender con la práctica o la observación: como analista novato, es buena idea acompañar a un veterano y observar algunas entrevistas. Además, obtenga retroalimentación: pídale a su jefe que investigue lo que los usuarios opinan sobre cómo hace sus entrevistas. Y deles retroalimentación a los usuarios: dígales para qué se usarán los resultados de su entrevista, para que no piensen que todo fue un desperdicio de tiempo.

REFERENCIAS

1. Abraham Maslow, *Motivation and Personality.* Nueva York: Harper & Row, 1954.
2. Charles J. Stewart y Cash Stewart, *Interviewing Principles and Practices,* 2a. edición, Dubuque, Iowa: William C. Brown, 1978.
3. William S. Davis, *Systems Analysis and Design: A Structured Approach.* Reading, Mass.: Addison Wesley, 1983.

APENDICE

F CASO DE ESTUDIO: YOURDON PRESS

F.1 INTRODUCCION

Ninguna discusión sobre análisis de sistemas estaría completa sin por lo menos un ejemplo que ilustre las diversas herramientas y técnicas de modelado que se discuten en este libro. Desafortunadamente, es probable que casi cualquier caso de estudio sea completamente ficticio, o bien sea una versión excesivamente simplificada y "esterilizada" de la situación real. Además, sería difícil encontrar un ejemplo que ilustre una aplicación orientada tanto a los negocios como a la ciencia.

Este caso de estudio describe los requerimientos de computarización de las actividades de proceso de información de YOURDON Press. Por un lado, es muy característico de la actividad editorial real vigente durante alrededor de 10 años. De hecho, una de las cosas que quiero mostrar en este caso de estudio es que no siempre se hacen las cosas por razones racionales (incluyendo la formación de compañías y la iniciación de muchos proyectos de desarrollo de sistemas), y que la mayoría de los sistemas tienen que lidiar con muchos detalles desagradables en la vida real.

Por otro lado, YOURDON Press ya se incorporó a las filas de los ejemplos ficticios, pues Prentice-Hall la compró en 1986, y sus actividades de proceso de información se han incorporado a las de ésta.[1] Por ello, este caso de estudio describe lo que hubieran sido los requerimientos de proceso de información de YOURDON Press de haber continuado como editorial independiente.

Las siguientes secciones ofrecen una breve reseña de la operación de YOURDON Press, el modelo ambiental del sistema, el modelo de comportamiento y el modelo de implantación del usuario.

1 Entretanto, las actividades de procesamiento de información de Prentice-Hall las está asumiendo su nueva compañía matriz, Simon & Schuster. Y esta última es parte de una empresa aún mayor, Gulf + Western, lo cual demuestra que los sistemas casi siempre son parte de sistemas mayores.

F.2 ANTECEDENTES

Para entender cómo trabajaba YOURDON Press, es necesario pasar un poco de tiempo explicando el contexto más amplio de la corporación dentro de la cual existía: YOURDON inc. Sin YOURDON inc., no hubiera existido YOURDON Press; aunque, sin YOURDON Press, está claro que YOURDON inc. no hubiera logrado el éxito del que goza.

YOURDON inc. se formó como resultado del crecimiento de las actividades independientes de consulta y de enseñanza que yo había estado realizando durante varios años a fines de los años 60 y comienzos de los 70. La compañía se formó en abril de 1973 porque mi contador me informó de las ventajas que en el área de impuestos ofrecía tener una corporación en lugar de ser un consultor autoempleado. A pesar de estos consejos prácticos, la corporación no funcionó realmente sino hasta el día de los inocentes, en abril de 1974.

Como suele suceder en la mayoría de las compañías (y la mayoría de los proyectos de proceso de datos), una de mis primeras actividades fue pensar en un nombre adecuado para la compañía. A mi esposa y a mí, quienes éramos los únicos accionistas, directores, oficiales y empleados de la compañía, nos gustó el nombre de "Alcachofas y Otros Animales Peludos, Inc.", pero decidimos que el membrete no cabría en nuestros documentos. Finalmente nos decidimos por el nombre "Superprogramadores, Ltd.", e hicimos las solicitudes necesarias, en el estado de Nueva York, para establecer el nombre. Como dos semanas después, cuando justo íbamos a poner algunos anuncios para nuestra primera serie de seminarios sobre programación estructurada, el estado nos informó que no se había aprobado el nombre de nuestra compañía: se parecía demasiado al nombre de una compañía ya existente. Cuando investigamos, encontramos que el nombre de la otra compañía era "Productos de Supermercado, Inc."[2] Desesperados, rápidamente escogimos un nombre del cual estábamos razonablemente seguros que ninguna otra compañía tendría: mi propio apellido. Así, nació YOURDON inc.

Las primeras actividades de la compañía fueron seminarios profesionales sobre técnicas avanzadas de programación y diseño de sistemas en línea, dirigidas a programadores y analistas veteranos de grandes organizaciones y agencias gubernamentales. Los seminarios eran de unas 20 horas de clase-pizarrón, acompañados de unas 100 páginas de notas; las notas del seminario sobre técnicas avanzadas de programación finalmente se convirtieron en un libro de texto: *Techniques of Program Structure and Design,* publicado por Prentice-Hall en 1975.

2 Cuando lo investigamos, descubrimos que Supermarket Products se encontraba en alguna parte de la periferia de la ciudad de Nueva York y que se ocupaba principalmente de importar plátanos de Guatemala. No veíamos qué tenía que ver esto con las computadoras ni por qué nuestro nombre había de interferir con el pobre Supermarket Products original, pero optamos por eludir el choque con la burocracia.

Debido al gran número de participantes en los seminarios, convino imprimir las notas en volumen moderado y encuadernarlas; así que a grandes rasgos se parecían a un libro, aunque algunas de las páginas quedaran de cabeza y otras se desprendieran del libro a la menor provocación. No obstante, algunos de los participantes de los seminarios pedían comprar copias adicionales de las notas y, por ello, como negocio paralelo, YOURDON inc. incursionó en la venta de "libros".

Sin embargo, YOURDON inc. se concentraba sobre todo en las actividades de enseñanza: el número de cursos distintos aumentó a aproximadamente 50 para mediados de los años 80, y la compañía ya ha preparado a unos 250,000 profesionales de proceso de datos en los EUA y más de otros 30 países. También comenzaron a crecer las actividades de consultoría profesional, y muchos de los miembros del personal técnico de la compañía ahora son consultores, jefes de proyecto y analistas en proyectos importantes de desarrollo de sistemas de compañías clientes en Norteamérica y Europa. A mediados de los años 80, la compañía ingresó al mercado de CASE, con un producto de paquete de herramientas del tipo que se describe en el apéndice A. En 1987, YOURDON inc. tenía sucursales en 8 ciudades, con alrededor de 150 empleados.

YOURDON Press surgió como una división de YOURDON inc. en 1976, con la publicación de tres libros: *Structured Design,* de Yourdon y Constantine; *Learning to Program in Structured COBOL,* de Yourdon, Gane y Sarson; y *How To Manage Structured Programming,* de Yourdon. Como sucedió con muchas organizaciones reales de negocios, esto resultó sin mucha planeación o pensamiento organizado: los libros parecían ser una buena manera de popularizar los conceptos de técnicas estructuradas que se estaban desarrollando y vendiendo en los seminarios de enseñanza de YOURDON inc.

Los tres primeros libros se produjeron con ayuda de una máquina de escribir IBM Selectric y se encuadernaron en hojas de 8.5 X 11 pulgadas; todo esto fue antes de los días de la selección de tipos y edición de escritorio fácil. La publicidad era modesta, y consistía tan sólo de unos cuantos anuncios en *Computerworld* y de comunicados a los integrantes de la lista de clientes de los seminarios de YOURDON inc. Las ventas eran igualmente modestas; de hecho, durante los primeros años de su existencia, YOURDON Press representaba sólo una pequeña fracción de los ingresos globales de la compañía.

En consecuencia, el sistema de información de las primeras actividades de YOURDON Press era modesto y completamente manual. Se tomaban pedidos por teléfono o por correo, pero no se aceptaban pedidos con tarjeta de crédito. Las facturas se mecanografiaban en formas de factura de cuatro tantos, y los pedidos se empacaban individualmente a mano. El inventario se almacenaba en uno de los espacios de bodega más elegantes del mundo: oficinas en el piso 38 de la sucursal de YOURDON inc., situada en el 1133 de Avenue of the Americas, con vista a todo Manhattan.

En el verano de 1976 llegó la automatización, en la forma de una minicomputadora PDP-11/45 y un misterioso sistema operativo llamado UNIX.[3] Unos meses más tarde, se le añadieron una máquina de fotocomposición, dos docenas de terminales y el paquete TROFF para creación de tipos. Esto inmediatamente facilitó la producción de libros de texto de YOURDON Press y finalmente llevó a la automatización de diversos aspectos del negocio de enseñanza y actividades generales de contabilidad de YOURDON inc. Pero las actividades operacionales de YOURDON Press, es decir, las que se considerarían como un "sistema de información", siguieron en forma manual durante varios años más.

En 1980 se desarrolló un número limitado de aplicaciones para YOURDON Press, usando las convenientes características de interconexión de procesos del sistema operativo UNIX. Entre 1980 y 1985 se usaron gradualmente el lenguaje C y varios programas del Shell de UNIX para añadir unos cuantos programas sencillos para proceso de órdenes, reportes de ventas, etiquetas de envío y reportes varios de contabilidad. Aunque estos programas fueron fáciles de desarrollar y razonablemente confiables de operar, se habían desarrollado en forma fraccionada, como los que se pueden ver hoy en día en organizaciones de proceso electrónico de datos donde los usuarios finales tienen acceso a hojas de cálculo, generadores de reportes y lenguajes de programación de cuarta generación. También tenían ciertas limitaciones; por ejemplo, no se podían modificar los detalles de un pedido después de ingresarlo al sistema, sino que se usaba el editor de textos estándar de UNIX para modificar el pedido, que se almacenaba en la computadora en forma de un simple texto en ASCII, terminado con un carácter de final de línea.

Una de las actividades más difíciles de la operación diaria de YOURDON Press era la tarea de producir una declaración actualizada que mostrara todos los pedidos, pagos, devoluciones de libros y créditos de clientes durante un periodo. El proceso de reconciliar dichas actividades (que se daba en forma de interacciones entre los clientes y el personal administrativo de YOURDON Press) con los registros financieros que conservaba el departamento de contabilidad de YOURDON inc. era igualmente difícil. Por diversas razones, YOURDON Press y el departamento de contabilidad siempre parecían estar "fuera de sincronía". Esto se complicó más por el hecho de que la sucursal de YOURDON inc. en Londres tenía su propio inventario de libros y realizaban sus envíos y su facturación en forma independiente a los de la sucursal en Nueva York; los precios se daban en libras esterlinas en lugar de en dó-

---

3 Desde luego, UNIX no es tan misterioso ahora, pero a mediados de los años 70 difícilmente alguien fuera de Bell Laboratories y de unas cuantas universidades lo habían oído mencionar. Ni yo ni la mayoría de mis colegas de YOURDON teníamos tal grado de precognición. Le debimos nuestra decisión -lo que más tarde hubimos que agradecer profundamente- a los apremios del Dr. J. Plauger, que había entrado a nuestra empresa procedente de Bell Labs en 1975. Plauger es ampliamente conocido por los libros que escribió en colaboración con Brian Kernighan, entre los que se cuentan The Elements of Programming Style (Elementos del estilo de programación) (Reading, Mass.: Adisson-Wesley, 1973) y Software Tools (Herramientas de programas y lenguajes) (Nueva York: McGraw-Hill, 1976).

lares y eran por lo general un tanto más caros que los de la sucursal neoyorquina.[4] Una vez al trimestre, cuando se tenían que preparar declaraciones financieras, se convocaba a largas y frustrantes juntas en las cuales se comparaban manualmente las salidas impresas de computadora producidas por el departamento de contabilidad, con las producidas por YOURDON Press en un intento de reconciliar las diferencias. La gente se enojaba; se gritaban insultos, obscenidades y adjetivos; en ocasiones se lanzaban objetos de un lado a otro del salón. No era una actividad grata ni deseable cada trimestre.

Por eso, para 1986, era evidente que se tendría que desarrollar un sistema completo si YOURDON Press iba a continuar creciendo; se comenzó la planeación inicial del nuevo sistema. Sin embargo, también era evidente que se requerirían cantidades substanciales para continuar haciendo crecer la organización, no sólo para equipo de cómputo adicional, sino también para modernizar la máquina de fotocomposición (ya obsoleta) y crecer las actividades editoriales y de mercadeo de la división. Finalmente se decidió que sería mejor que una organización más grande adquiriera la operación de publicaciones, y esto fue lo que llevó a la fusión con Prentice-Hall. Por tanto, los modelos de sistema que se describen a continuación corresponden a lo que hubieran sido los requerimientos si YOURDON Press hubiera continuado su operación como negocio independiente.

La planeación de un nuevo sistema de información también coincidió con una serie de cambios en la organización de YOURDON Press y el resto de YOURDON inc. Desde su inicio en 1974 hasta aproximadamente 1983, la compañía tenía la estructura organizacional que se muestra en la figura F.1.

Entre 1984 y 1986, la compañía se volvió una organización más regional, y añadió una nueva división para sus productos de software, como muestra la figura F.2.

Y durante este periodo, YOURDON Press gradualmente desarrolló la estructura organizacional que se muestra en la figura F.3.

4 La cuestión de los inventarios separados y de las ventas de las oficinas separadas se cernía en el horizonte como un problema cada vez mayor. En cada una de las diversas oficinas de YOURDON se insistía en que necesitaban tener un pequeño inventario local para dar servicio a los clientes que las visitaban y que querían adquirir un libro inmediatamente, en vez de esperar varios días (o semanas) a que se los enviara Galactic Headquarters. Y en la oficina canadiense se reclamaba que necesitaban su propia estructura de precios (esto es, precios formulados en dólares canadienses, en vez de estadounicenses) y su propia campaña de comercialización y publicidad para atraer al mercado canadiense de manera diferente que al de EUA En algunos casos, las oficinas lejanas simplemente le daban el libro al cliente y le pedían a la oficina central de Nueva York que generara la factura. En otros casos, el cliente pagaba el libro en el sitio mismo y pedía un comprobante. Las ventas de la oficina de Londres dieron cuenta de aproximadamente 10 por ciento del total de ingresos de YOURDON Press, mientras que las ventas de las otras oficinas dieron cuenta de menos del 1 por ciento del total de ingresos de YOURDON Press.

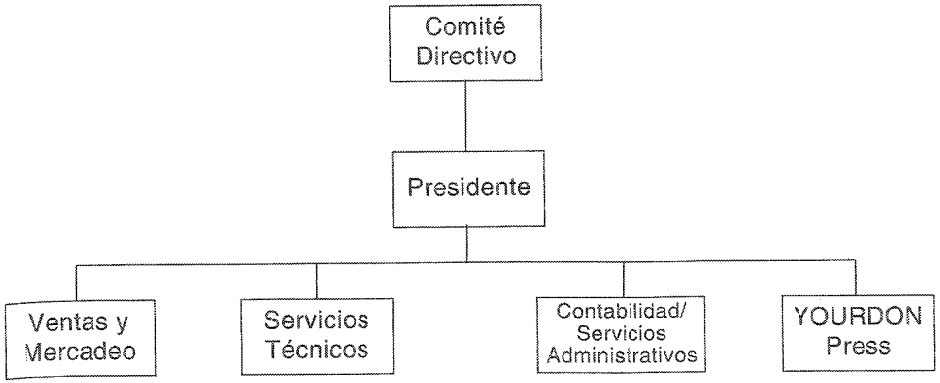


Figura F.1: **Estructura organizacional de YOURDON inc., 1974-1983**

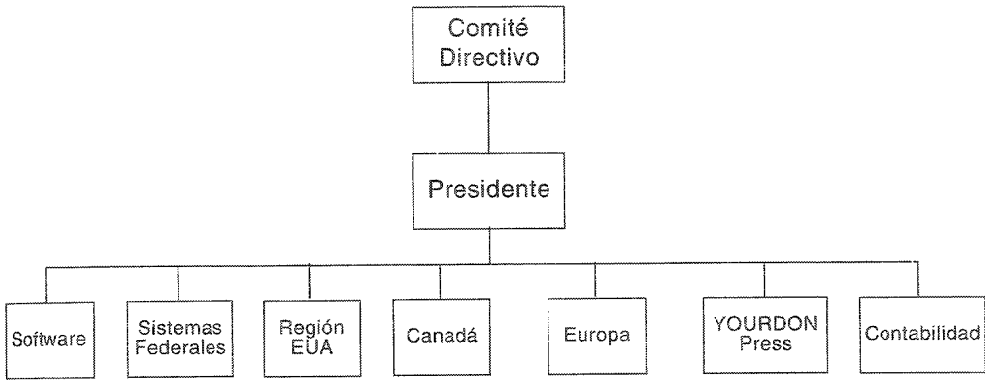


Figura F.2: **Estructura organizacional de YOURDON inc.,1984-1986**

Como parte de su reorganización, las operaciones de envío de YOURDON Press se mudaron del elegante espacio de oficinas que ocupaba el resto del personal a una bodega en una bella sección del centro de Yonkers, Nueva York. Así, existe una separación física de unos 30 kilómetros entre quienes ingresan los pedidos y quienes empacan los libros y los mandan a los clientes.

Los cuatro grupos principales dentro de YOURDON Press tenían las siguientes responsabilidades:

- *Servicios administrativos,* se hacían responsables de la mayoría de las interacciones cotidianas entre YOURDON Press y los clientes. Por tanto,

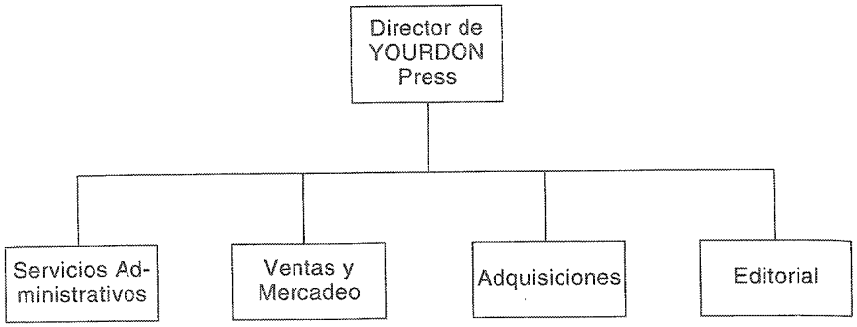


Figura F.3: **Estructura organizacional de YOURDON Press**

este grupo aceptaba pedidos, producía facturas, recibía pagos, discutía devoluciones y crédito con los clientes, interactuaba con la bodega para el envío de libros e interactuaba con el departamento de contabilidad, como se discutió anteriormente.

- *Ventas y mercadeo*, responsable de producir catálogos de los diversos libros de YOURDON Press, de colocar anuncios en revistas de computación y boletines de comercio, de mandar folletos de promoción a listas de clientes, y de lograr ventas por teléfono a grandes corporaciones que compran libros técnicos de computación.
- *Adquisiciones*, este grupo se hacía cargo de encontrar nuevos autores y nuevos libros. Esta parte de YOURDON Press llevaba a cabo todas las discusiones con los autores hasta el momento de la entrega del manuscito final.
- *Editorial*, grupo responsable de convertir el manuscrito entregado en un libro publicado. Esto no solo involucraba la edición sino también la interacción con los impresores para obtener propuestas para la impresión inicial. El grupo editorial también se hacía responsable de las ilustraciones y la producción de la portada, además de su contenido.

Debe tenerse en cuenta que YOURDON Press era una operación relativamente pequeña comparada con operaciones tan bien conocidas como McGraw-Hill, Prentice-Hall, y Random House. De las siguientes estadísticas se puede formar una idea de la escala de la operación:

- YOURDON Press ofrecía una lista de aproximadamente 50 libros; típicamente se añadían a la lista de 4 a 6 títulos nuevos anualmente.

- Los libros estaban escritos por aproximadamente dos docenas de autores, y el grupo de adquisición interactuaba con aproximadamente 200 autores potenciales, es decir, con aproximadamente 200 individuos que expresaban algún interés por escribir un libro, pero que no habían de hecho terminado de escribir algo.
- YOURDON Press procesaba aproximadamente 50 pedidos diarios.
- El pedido promedio ascendía a aproximadamente US $100, lo cual típicamente representaba tres o cuatro libros. Algunos pedidos, desde luego, eran de un solo libro; otros eran mayores. Se celebraba cada vez que se recibía un pedido con valor de más de 5000 dólares estadounidenses
- Se enviaban aproximadamente 50,000 libros anualmente.

Aparte de su pequeña escala, sin embargo, YOURDON Press operaba de una manera muy similar a la forma en que lo hacen otras editoriales más grandes. Las ventas se hacían por medio de pedidos por escrito, por teléfono o en persona (es decir, una persona que llegara a las oficinas de YOURDON inc./YOURDON Press para comprar algún libro). Los pagos se podían hacer en efectivo (lo cual era raro), por cheque o con tarjeta de crédito. Como política de la empresa, los pedidos inferiores a 100 dólares tenían que pagarse por adelantado; los pedidos mayores, sobre todo los de librerías y de compañías, por lo general requerían de facturas.

Para comprender el negocio de las publicaciones, también se debe estar familiarizado con el concepto de devoluciones. Si un cliente individual o una corporación sentían que un libro dado no satisfacía sus necesidades, o lo recibían dañado, generalmente lo regresaban y pedían una devolución de su dinero. Esto normalmente ocurría días después de que el cliente recibía el envío. Por otro lado, las librerías gozaban del privilegio de devolver hasta la mitad de los libros de un pedido en el transcurso de un año a partir de la fecha de recibo del pedido; esto es bastante común en la industria de las publicaciones, porque muchas veces las librerías no saben de antemano la demanda de cierto libro y evitan quedarse con un inventario que no pueden vender.

F.3 EL MODELO AMBIENTAL

F.3.1 La declaración de propósitos

El propósito del Sistema de Información de YOURDON Press (YPIS) es almacenar la información necesaria para vender libros a los clientes. Esto incluye ingreso de pedidos, facturación, generación de documentos de envío, control de inventarios y producción de reportes de regalías por derechos de autor y reportes de contabilidad.

F.3.2 El diagrama de contexto

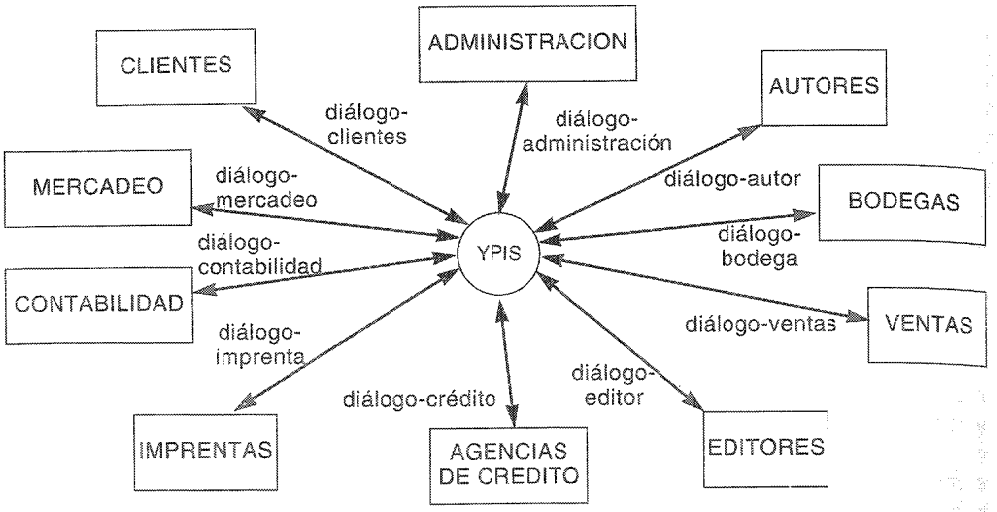


Figura F.4: **Diagrama de Contexto para el sistema YPIS**

F.3.3 La lista de acontecimientos

La lista de acontecimientos del sistema YPIS consiste en 40 acontecimientos. La mayoría están dirigidos por flujo, aunque la mayoría de los que involucran al departamento de contabilidad son temporales. Los acontecimientos se listan a continuación; los temporales se marcan con una "T" luego de su descripción.

1. El cliente pide un libro (esto también incluye pedidos urgentes especiales).
2. El cliente envía su pago.
3. El cliente pide información sobre algún libro (precio, etc.).
4. El cliente pide permiso de devolver un libro.
5. El cliente pregunta sobre el status de algún pedido.
6. El cliente pregunta sobre el status de alguna factura.
7. El cliente requiere de una declaración (mensual). (T)
8. El cliente pide un recordatorio de crédito.
9. El cliente desea un cheque de reembolso.

10. El departamento de contabilidad requiere de recibos (diarios) de efectivo. (T)
11. El departamento de contabilidad requiere de reportes de ventas (diarios). (T)
12. El departamento de contabilidad requiere de un reporte (mensual) de ventas netas. (T)
13. El departamento de contabilidad requiere de un reporte (trimestral) de regalías por derechos de autor. (T)
14. El departamento de contabilidad requiere de datos (mensuales) de inventario. (T)
15. El departamento de contabilidad requiere de un reporte (mensual) de comisiones sobre ventas. (T)
16. La administración fija un nuevo límite de crédito para un cliente.
17. La administración requiere de un reporte de cuentas por cobrar vencidas (mensual). (T)
18. La imprenta da una cotización de pedido de impresión (o de reimpresión).
19. La administración autoriza un pedido de impresión.
20. La imprenta notifica la cantidad exacta de impresos y fecha de entrega.
21. La imprenta envía factura por concepto de trabajo de impresión.
22. La administración solicita cotización de un pedido de impresión.
23. El departamento de mercadeo pide etiquetas de envío de la base de datos de clientes.
24. El departamento de mercadeo requiere de estadísticas sobre las ventas de libros.
25. El departamento de mercadeo necesita una fecha de disponibilidad de nuevos títulos.
26. Los editores anuncian un nuevo título (fecha en la que está listo para imprenta)
27. Los autores necesitan un reporte trimestral de utilidades por concepto de derechos de autor. (T)
28. La bodega necesita datos de envío y etiquetas. (T)
29. La bodega recibe libros de la imprenta.

30. La bodega recibe devoluciones de libros de un cliente.
31. La bodega hace un inventario físico (mensual).
32. La bodega envía un pedido a un cliente.
33. La bodega anuncia que un libro se ha agotado.
34. El departamento de adquisiciones anuncia el proyecto de un nuevo libro.
35. Un vendedor ingresa un pedido de parte de un cliente.
36. El departamento de mercadeo declara que un libro está agotado.
37. El cliente notifica un cambio de domicilio.
38. El autor notifica un cambio de domicilio.
39. El cliente elige participar en el plan de agencia.
40. Se necesita enviar facturas a un cliente. (T)

F.4 EL MODELO DE COMPORTAMIENTO

F.4.1 El modelo preliminar de comportamiento: diagramas de flujo de datos

Cada uno de los 40 acontecimientos listados en la sección F.3.3 tiene un diagrama de flujo de datos asociado. Desde luego, la logística de la impresión de un libro vuelve poco práctico, por decir lo menos, conectar los 40 diagramas en un solo diagrama compuesto que represente todo el sistema. Como se señaló en el capítulo 19, éste es el tipo de ejercicio que requiere de una hoja de papel muy grande, o incluso varias hojas unidas. Dejo esto como ejercicio al lector.

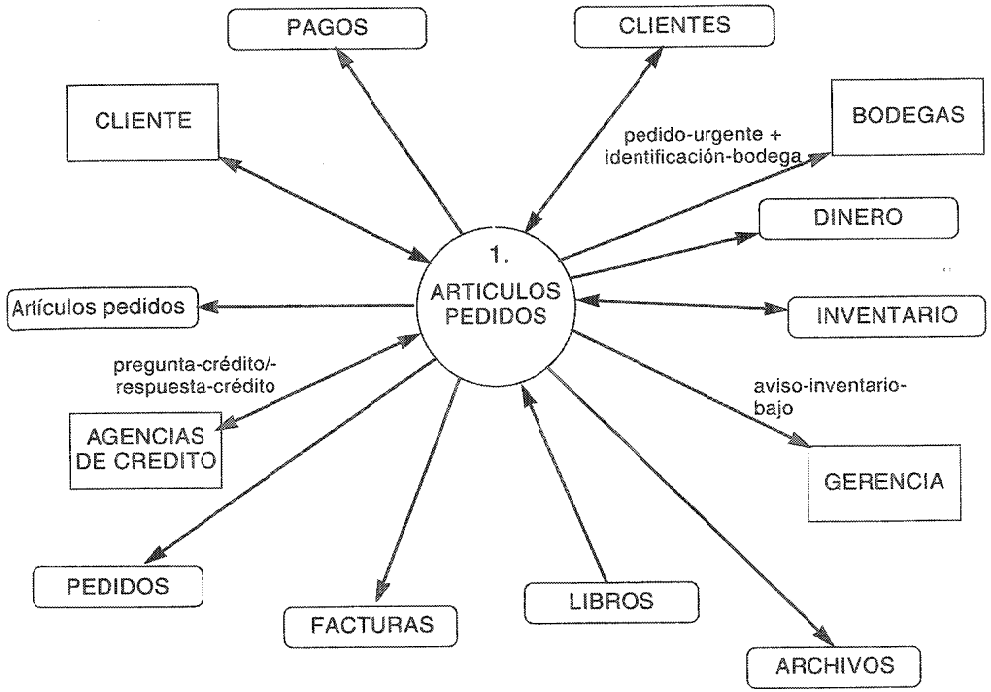
Los diagramas se dibujaron con la versión 2.0 del paquete **Design**, de Meta Systems Inc., Cambridge, Mass. Aunque no representa un paquete completo de herramientas CASE, es más elaborado que la mayoría de los paquetes sencillos de gráficas, y tiene la ventaja de ser ejecutable en una computadora Macintosh, que se usó para la preparación de este libro. Para acoplarse al programa **Design**, se muestran los almacenes del DFD con la notación de la figura F.5.

Figura F.5: **Notación para los almacenes en el caso de estudio de YOURDON Press**

Al dibujar los DFD preliminares tomé nota de los errores que encontré y los cambios que vi que tenía que hacer en otras partes del modelo; estas notas se listan debajo de cada DFD. La razón de esto es enfatizar que en un proyecto real el analista raramente dibuja un DFD perfecto al primer intento; después de pensar sobre el sistema, y luego de entrevistas de seguimiento con el usuario, es inevitable que se encuentren errores en el DFD que se está examinando o en alguna otra parte del modelo del sistema.

No se hizo ningún intento de crear un diccionario de datos organizado al desarrollar el modelo preliminar del comportamiento. Después de crear el modelo inicial del DFD se trazaron especificaciones de proceso para determinar si existían errores obvios. Muchos de esos errores se muestran como comentarios en las páginas siguientes. A continuación se creó un conjunto de DFD por niveles y se desarrolló el diccionario de datos.

Acontecimiento 1: El cliente pide un libro



Notas

1. Tras dibujar la primera versión de este diagrama, recordé que los pedidos con tarjeta de crédito normalmente requieren autorización si la cantidad sobrepasa algún límite prefijado. YOURDON Press aceptaba pedidos pagados con Mas-

tercard, Visa y American Express; por ello la interfase con el terminador etiquetado como "AGENCIAS DE CREDITO".

2. Pensar un poco más sobre la situación del crédito hizo obvio el hecho de que la definición de **cliente** en el almacén de **CLIENTES** tendría que incluir **límite-crédito** como campo. También se volvió evidente que se requería un acontecimiento para cambiar el límite de crédito del cliente (acontecimiento 16).

3. Observe que los pedidos *no* se envían uno por uno, con la excepción de pedidos urgentes. Los detalles de los pedidos urgentes se envían de inmediato a la bodega; todos los demás pedidos se almacenan simplemente en **PEDIDOS.** Como acontecimiento aparte (acontecimiento 28), la bodega recibe etiquetas de envío (normalmente correspondientes a un día). Olvidé los pedidos urgentes en la versión inicial del programa.

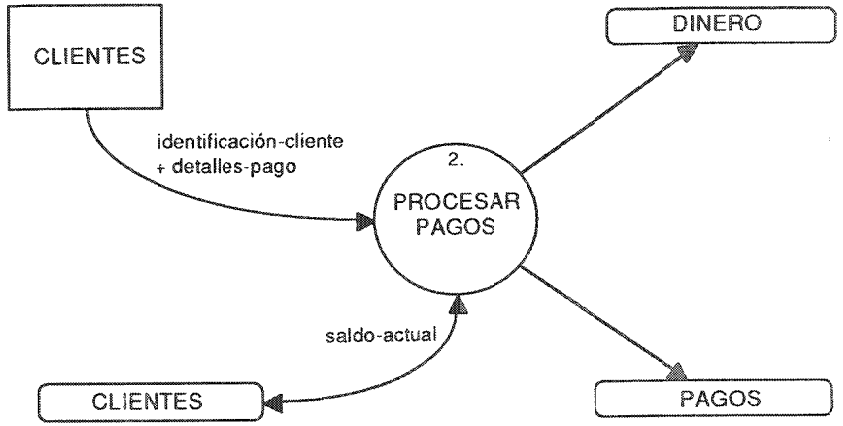
4. Cuando dibujé este DFD, también me di cuenta de la necesidad de un almacén **ARCHIVOS,** que es una copia del pedido por escrito original del cliente (o, en el caso de un pedido por teléfono, la forma de pedido del vendedor), más una copia de la factura que se generó para el pedido. No se necesita la copia de la factura en un modelo esencial (dado que se puede regenerar), pero los demás documentos sí, por si surgiera una disputa con el cliente, y en caso de auditorías o investigaciones de las autoridades de impuestos, etc.

5. Observe que se puede recibir un pedido por correo, por teléfono o en persona. No mostramos esto en el DFD anterior, porque todas éstas son funciones de transporte.

6. Observe que el sistema no vuelve automáticamente a pedir libros a la imprenta. En lugar de eso, repetidas veces se le informa a la administración que un inventario ha bajado más allá del umbral actual. Esto puede ocurrir como resultado del acontecimiento 1, al igual que como resultado de diversos acontecimientos más.

7. Se pueden recibir pedidos de nuevos clientes (sobre todo de librerías nuevas o de compañías que continuarán haciendo negocios con YOURDON Press). Por eso se tendrá que crear un nuevo registro en **CLIENTES** con la tasa estándar de descuento, etc. Esta es la razón de la flecha de dos cabezas entre la burbuja 1 y el almacén **CLIENTES.**

Acontecimiento 2: El cliente envía su pago.



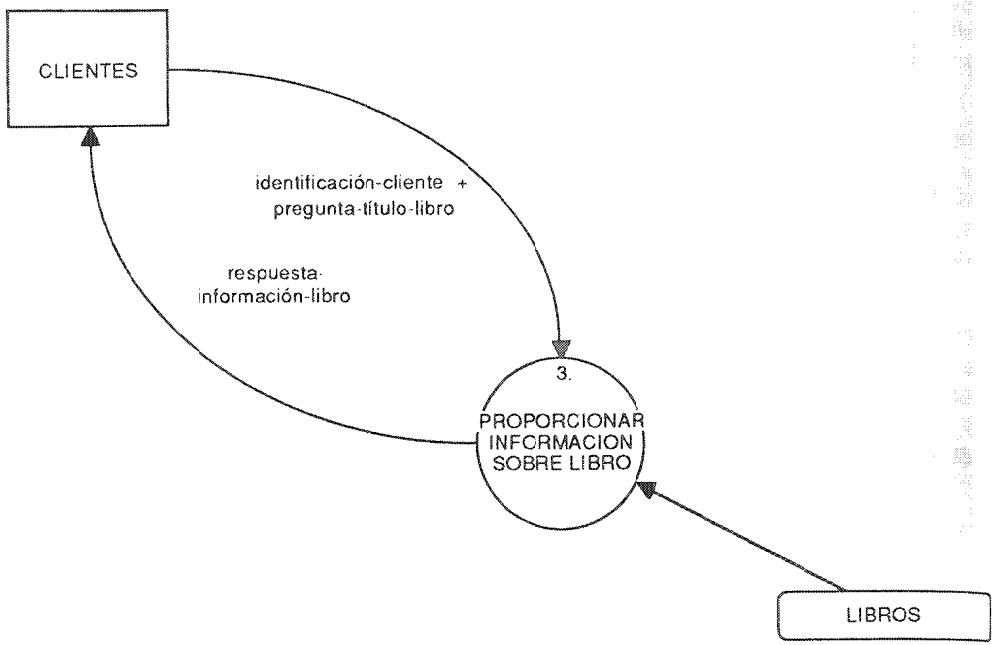
Notas:

1. El pago puede cubrir diversas facturas distintas, pero no siempre queda claro de cuáles facturas se trata. A veces los clientes no identifican la factura que están pagando; a veces identifican facturas que ya se pagaron; en ocasiones dan como referencia números de factura inexistentes.

2. A veces no queda claro incluso de dónde viene el pago. Esto sucede sobre todo en el caso de cadenas de librerías: la librería XYZ de la ciudad A puede ser propiedad de un conglomerado, PQR, de la ciudad B. Si llega un cheque cualquiera de la Corporación PQR a nombre de YOURDON Press, puede ser que no podamos determinar la factura o incluso la compañía involucrada. Los pagos de este tipo se almacenan en una categoría de contabilidad llamada efectivo no asentado. La suposición es que si continuamos mandando facturas retrasadas a la librería XYZ nos llamarán y nos dirán que la factura la pagó PQR.

3. No existe garantía de que el pago sea por la cantidad exacta de la factura. Algunos pagos son altos o bajos por una pequeña cantidad al azar. Algunos clientes tratan de evitar el pago del impuesto sobre la venta y los gastos de envío; esto usualmente da como resultado pagos uno o dos dólares por abajo de lo correcto.

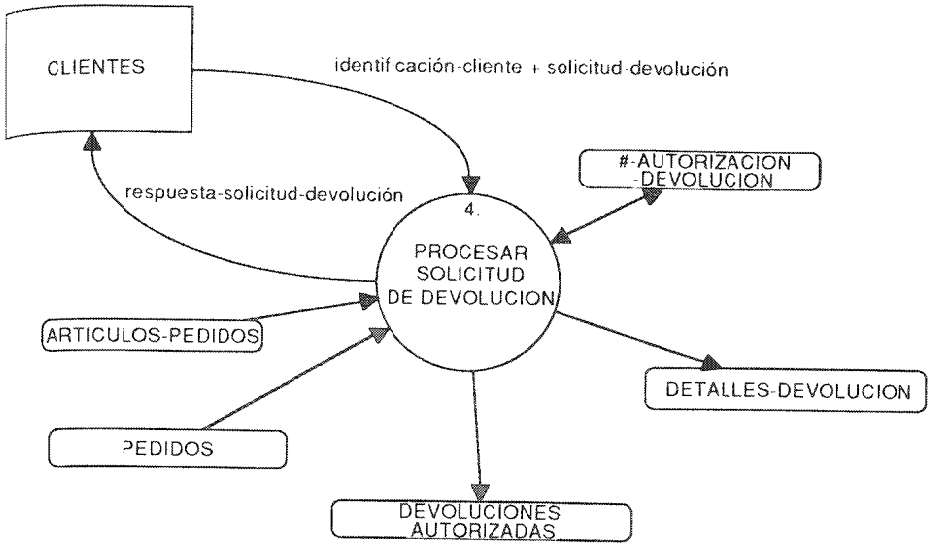
Acontecimiento 3: El cliente pide información sobre un libro.



Notas:

1. El cliente generalmente pregunta cosas tales como el precio del libro, o cuándo se espera tener en existencia cierto libro, o el programa de descuentos por volumen.

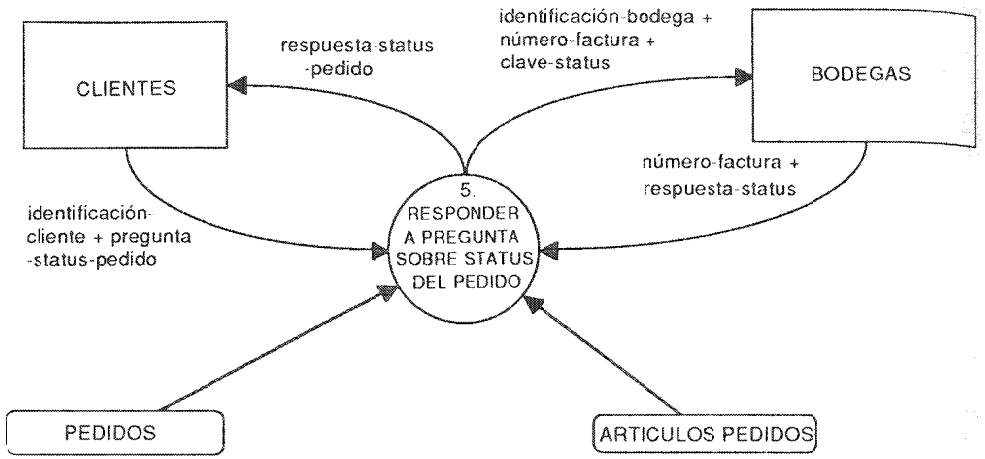
Acontecimiento 4: El cliente pide permiso de devolver un libro.



Notas:

1. Se supone que los clientes deben obtener la aprobación de YOURDON Press antes de devolver los libros. No siempre lo hacen.
2. Los libros devueltos llegan más tarde (acontecimiento 30) y pueden corresponder o no a la devolución solicitada que se autorizó aquí.
3. Observe que una devolución autorizada tiene que corresponder con el pedido original.

Acontecimiento 5: El cliente pregunta sobre el status de un pedido.



Notas:

1. Puede haberse retrasado el envío del pedido de algún cliente debido a lista de espera en la bodega o porque no hay libros del título requerido en existencia. Este retraso potencial es el que típicamente lleva a una pregunta del cliente.

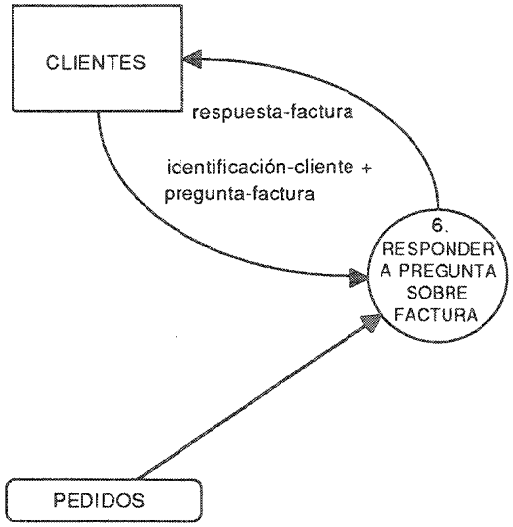
2. Si el cliente decide cancelar a estas alturas el pedido, se trata como un acontecimiento aparte (acontecimiento 8).

3. Otra posibilidad es que YOURDON Press no haya recibido el pedido (porque se perdió en el correo o en el departamento de correo de la oficina del cliente o de la oficina de YOURDON).

4. YOURDON Press puede haber recibido y procesado el pedido; de hecho, es posible que la bodega incluso haya enviado el pedido, pero que éste se haya perdido en el correo (o en otra agencia de transporte) en camino al cliente. Esto se trata de la misma manera (por ejemplo, el cliente puede decidir cancelar el pedido en este punto, o puede pedir crédito y volver a hacer el pedido).

5. Al desarrollar este DFD, me di cuenta de que no se han enviado facturas a los clientes (el envío de los libros por la bodega, y el envío de la factura por la oficina central de YOURDON Press son dos acontecimientos separados). Por ello, necesitamos un almacén aparte para las facturas, y un acontecimiento temporal que ocasione el envío de las facturas.

Acontecimiento 6: El cliente pregunta acerca del status de una factura.



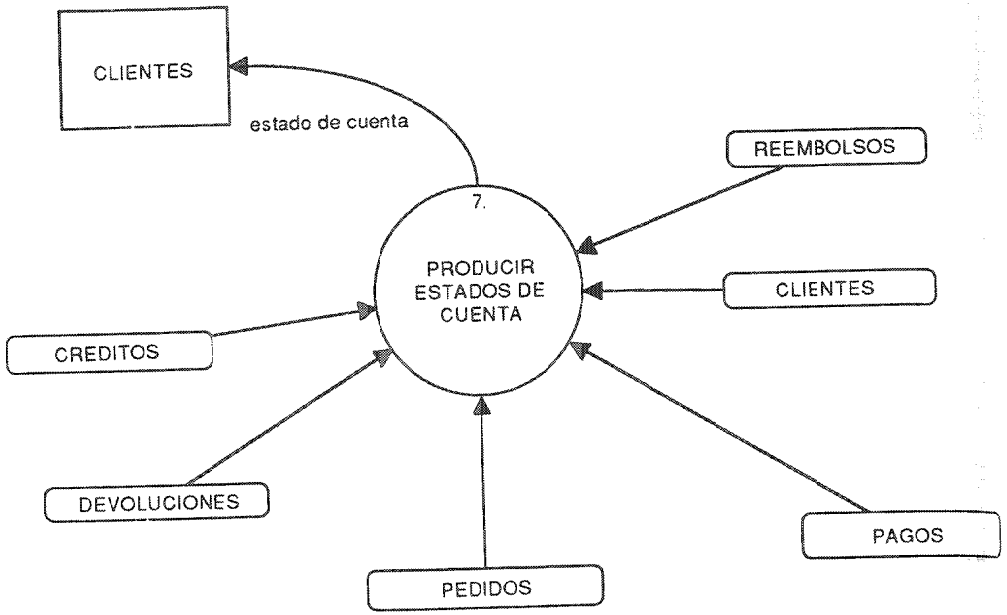
Notas:

1. La pregunta del cliente puede tener algo que ver con la tasa de descuento que se cotizó en la factura, o con los impuestos de envío, impuestos sobre la venta u otros aspectos de la factura.

2. Si el cliente hace una pregunta más general sobre todos sus pedidos, pagos, etc., la parte del sistema que lo maneja es el acontecimiento 7.

3. Para este acontecimiento, se necesita un **número-factura** para poder recuperar la información sobre el pedido. (**número-factura** es un componente de **pregunta-factura,** como se verá en el diccionario de datos.)

Acontecimiento 7: El cliente requiere un estado de cuenta (mensual).

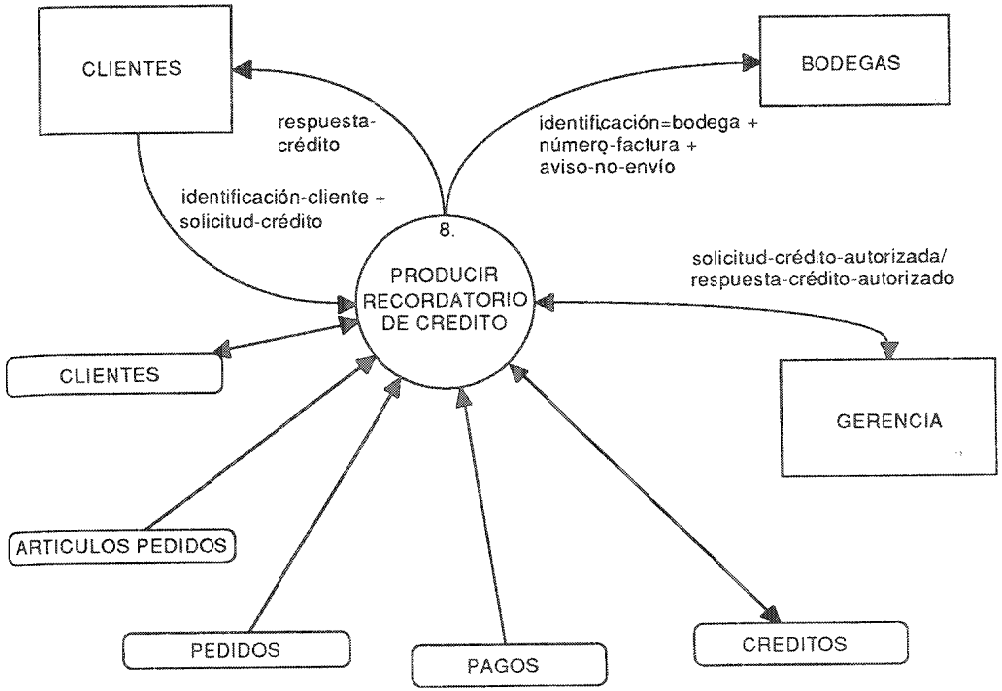


Notas:

1. Al trabajar sobre este DFD, descubrí la necesidad de un acontecimiento que permitiera al cliente solicitar crédito (acontecimiento 8).

2. Observe que este acontecimiento es temporal (por ejemplo, las declaraciones se generan regularmente, una vez al mes).

Acontecimiento 8: El cliente pide un recordatorio de crédito.



Notas:

1. Se le puede otorgar un crédito a un cliente por muchas razones distintas:

   - Un error en el pedido original (tal vez se le dio el descuento equivocado, o se le cobró mal, etc.)
   - El cliente recibió mercancía dañada (por ejemplo, la oficina de correos maltrató los libros).
   - El cliente recibió menos artículos de los solicitados en su pedido debido a un error de la bodega. En este punto solicita crédito por los libros que no recibió, en lugar de que le surtan el resto del pedido.
   - El cliente pagó en exceso una o más facturas anteriores y acaba de descubrirlo (normalmente esto sería obvio en cuanto recibiera la siguiente declaración).

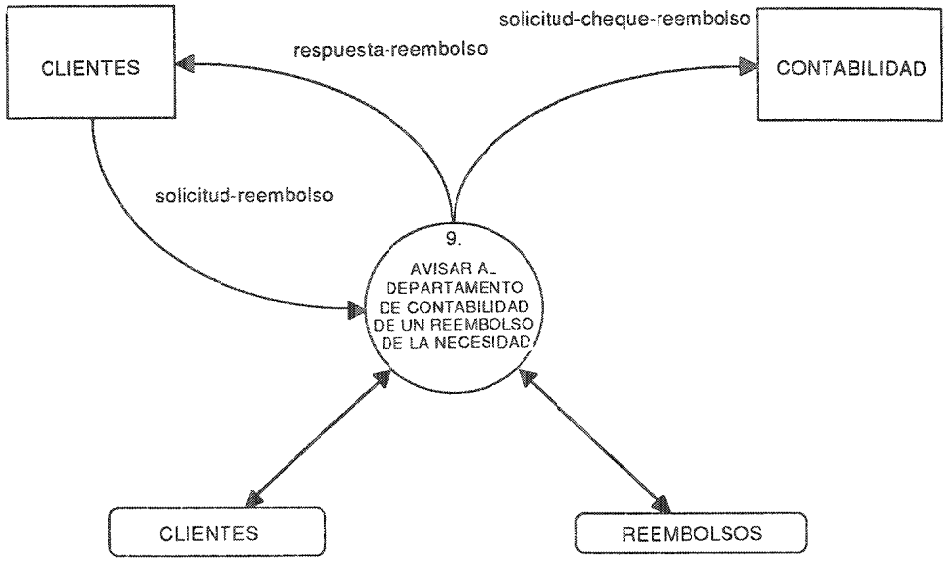
- Hubo un retraso excesivo en el envío, así que el cliente decidió cancelar el pedido.

2. Lo principal que esta burbuja tiene que hacer es actualizar el saldo de crédito del cliente.

3. No obstante, observe que la gerencia debe autorizar el crédito. Este acontecimiento se dibujó para mostrar una respuesta inmediata de la gerencia, para que se le pueda responder al cliente. Esto evita tener un almacén "pendiente" de solicitudes de autorización de crédito, que sería necesario de otra manera.

4. Observe que esta actividad nada tiene que ver con devoluciones de libros; éstas se manejan por separado.

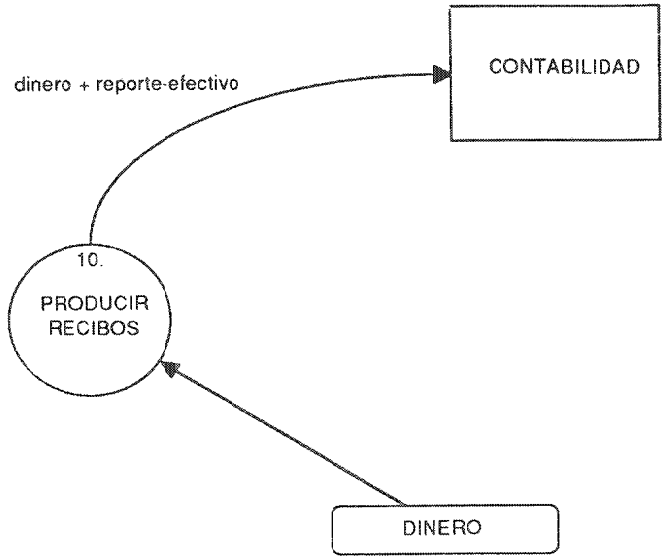
Acontecimiento 9: Un cliente desea un cheque de reembolso.



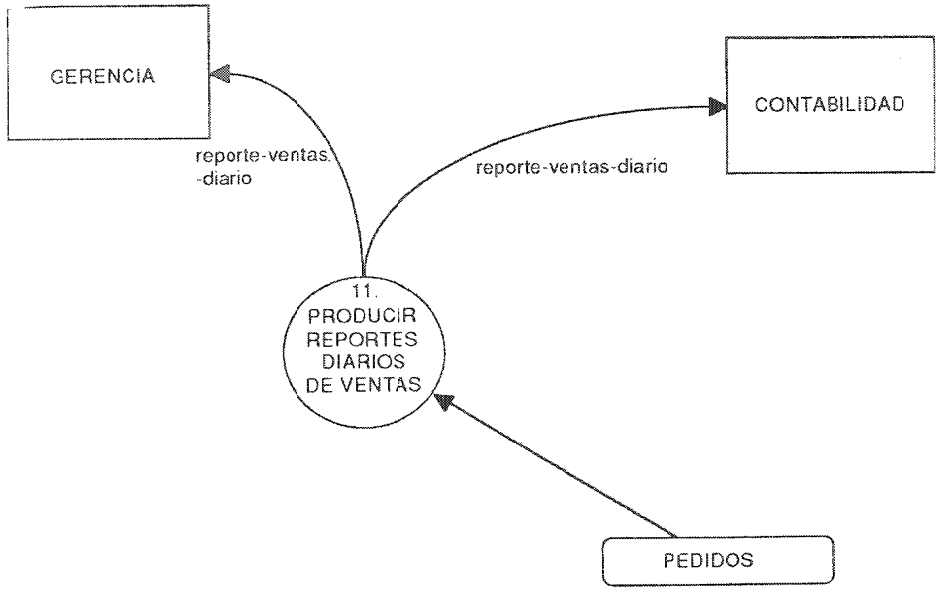
Notas:

1. Esto está bien si el cliente tiene saldo de crédito. En la mayoría de los casos, el cliente lo aplicará a compras futuras. Sin embargo, en ocasiones desea un cheque, porque no planea compras futuras o por alguna otra razón.

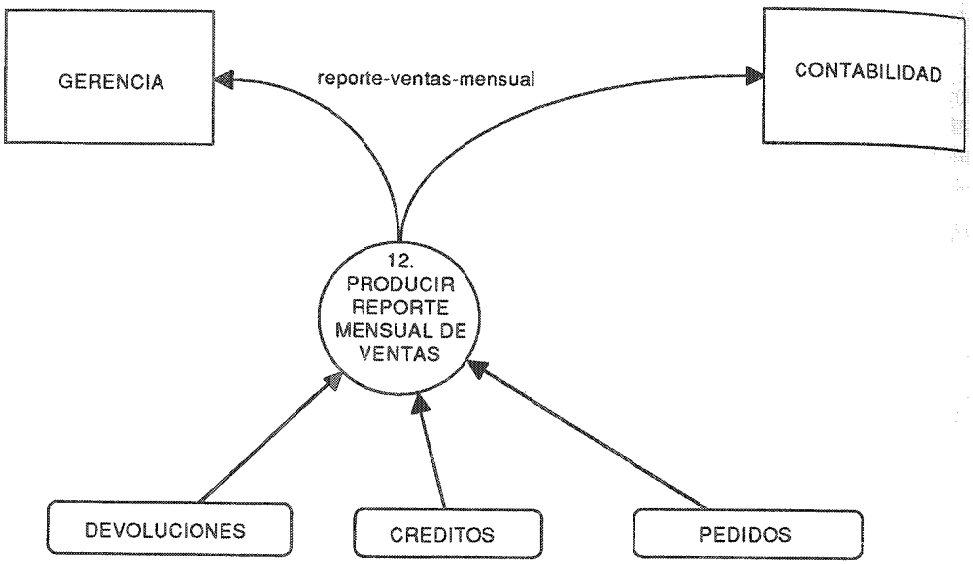
Acontecimiento 10: La contabilidad requiere recibos (diarios) de efectivo.



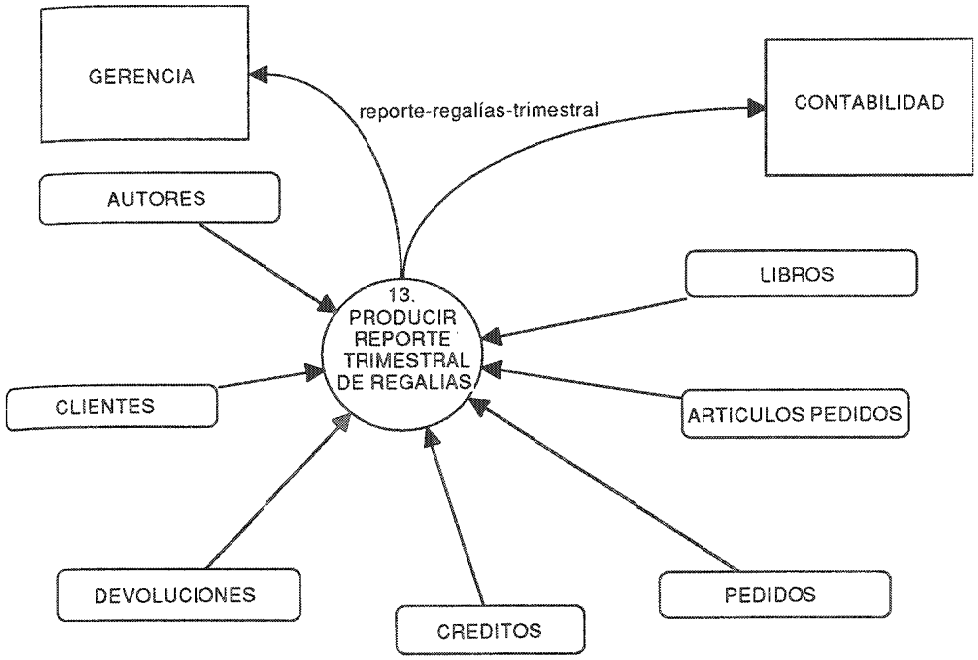
Acontecimiento 11: El departamento de contabilidad necesita reportes (diarios) de ventas.



Acontecimiento 12: El departamento de contabilidad necesita un reporte de ventas "netas" (mensual).



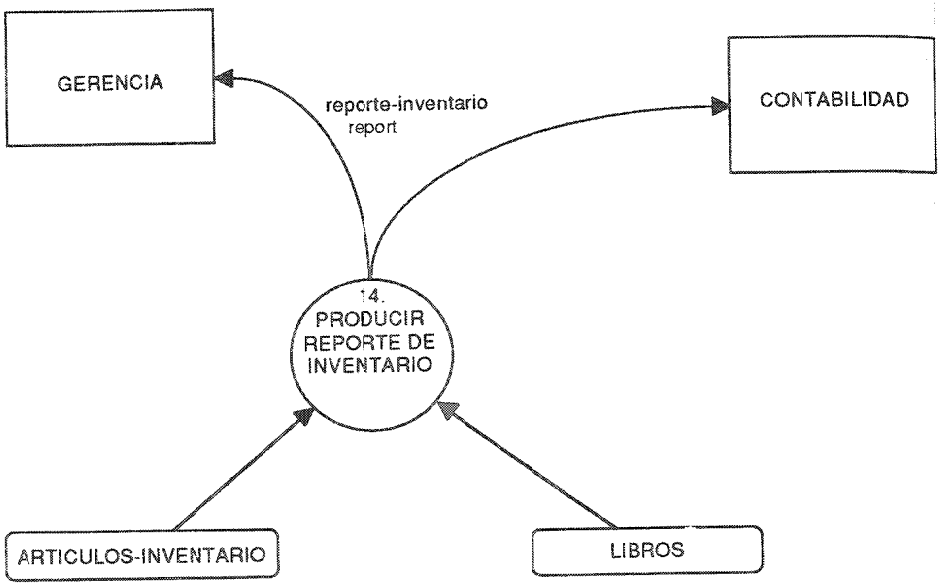
Acontecimiento 13: El departamento de contabilidad requiere un reporte (trimestral) de regalías de autor.



Notas:

1. Necesitamos tener acceso al almacén **LIBROS** para obtener la tasa de regalías del autor para el libro (el mismo autor puede tener diferentes regalías para libros distintos).
2. Necesitamos tener acceso al almacén **AUTORES** para obtener el número de Seguro Social, domicilio, etc.
3. Necesitamos tener acceso al almacén **LIBROS** para determinar si existen adelantos (que pueden haberse garantizado como resultado del acontecimiento 34) y actualizarlos para reflejar las regalías acumulativas actuales que se deben al autor.
4. Observe que las regalías de algún periodo dado pueden ser negativas si las devoluciones de libros para un libro dado exceden al número de libros pedidos.
5. Necesitamos el almacén **PEDIDOS** porque el departamento de contabilidad requiere detalles sobre quién compró los libros. No necesitamos esto en el acontecimiento 27.

Acontecimiento 14: El departamento de contabilidad requiere datos de inventario (mensuales).

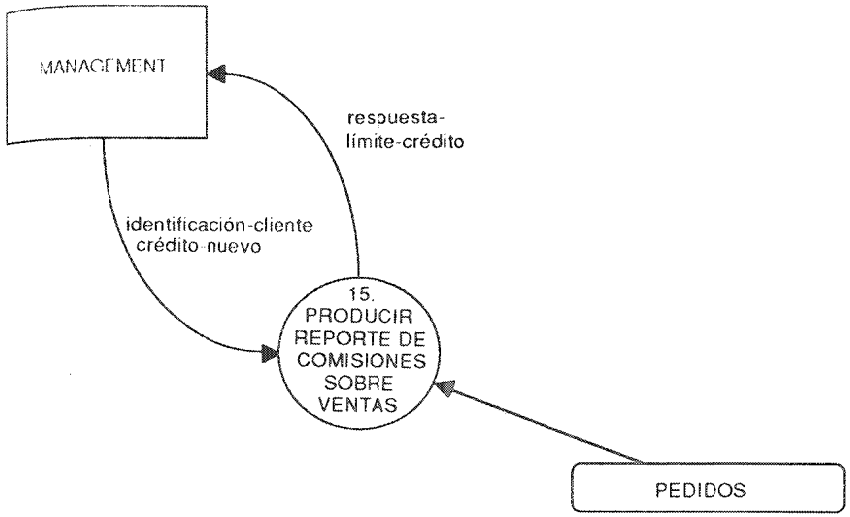


Notas:

1. El inventario se actualiza en forma no sincronizada como resultado de pedidos, devoluciones, recibo de envíos nuevos de la imprenta e inventario físico.

2. Observe que este reporte muestra libros que se han pedido, pero pueden no haber sido enviados por las bodegas. No necesariamente corresponderá con un inventario físico hecho en el mismo instante debido a los pedidos que ya se procesaron pero que aún no se han enviado.

Acontecimiento 15: El departamento de contabilidad requiere un reporte (mensual) de comisiones sobre ventas.



Notas:

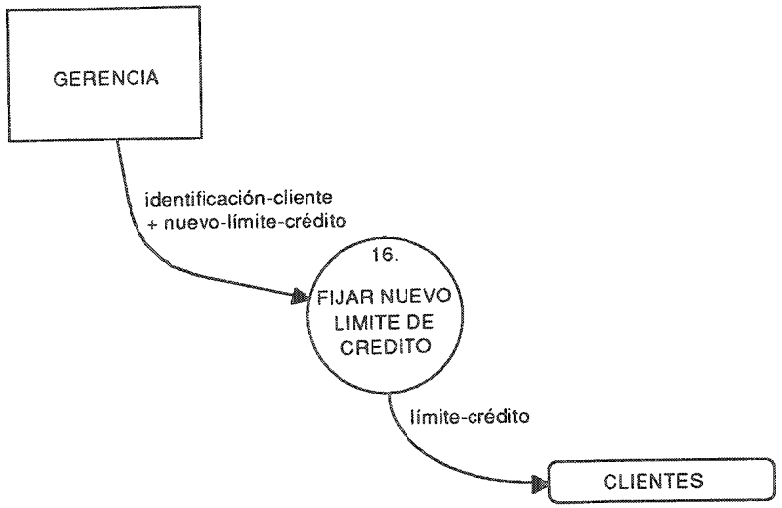
1. Esto supone que a los vendedores se les paga una comisión aun si el cliente no ha pagado. Ignora el suceso real de revertir una comisión si el cliente jamás paga y se tiene que invalidar la factura asociada.

2. Observe que muchas ventas no se asocian con un vendedor individual; se reciben, sin ser solicitadas, como resultado de campañas directas por correo, reseñas en periódicos y revistas computacionales, etc.

3. Este modelo también supone que a todos los vencedores se les paga la misma tasa de comisión, y que ésta es la misma para todos los libros. Sin embargo, la gerencia puede cambiar la tasa de comisión cada vez que ocurre este acontecimiento.

4. El modelo también supone que debemos mostrar los *detalles* del pedido a los vendedores, porque (siendo vendedores paranoicos típicos) no creen en la computadora.

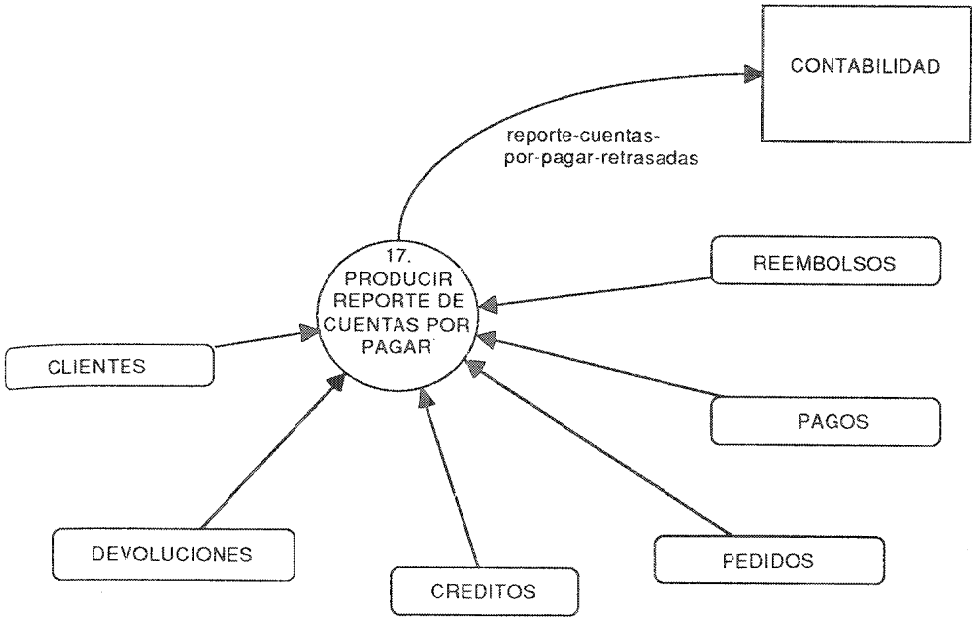
Acontecimiento 16: La gerencia fija un nuevo límite de crédito para un cliente.



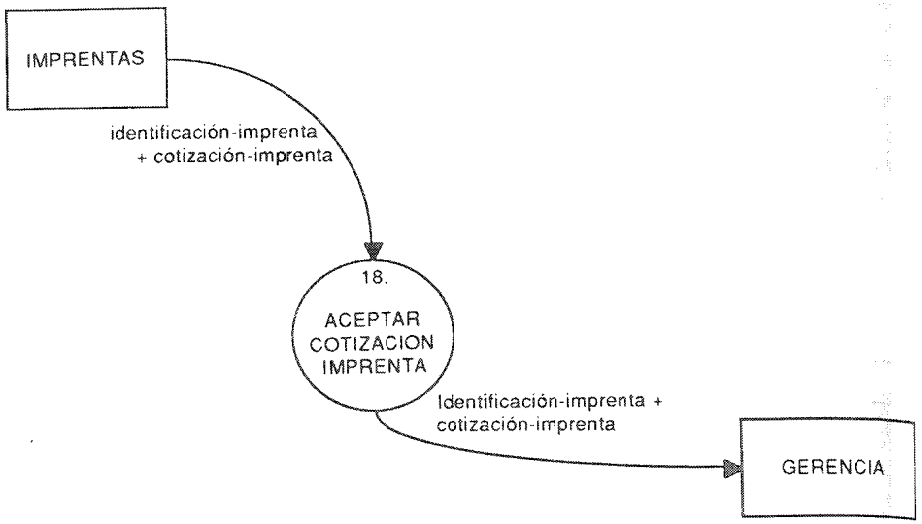
Notas:

1. La gerencia puede decidir cambiar el límite de crédito de un cliente por una diversidad de motivos, de los cuales el más común es la tardanza en los pagos de facturas previas. Sin embargo, también podría hacerse si el cliente ha ido a la quiebra o si la gerencia siente que han cambiado las condiciones generales de negocios.

Acontecimiento 17. El departamento de contabilidad requiere un reporte (mensual) de cuentas por pagar retrasadas.



Acontecimiento 18: La imprenta ofrece una cotización para un pedido de impresión (reimpresión).

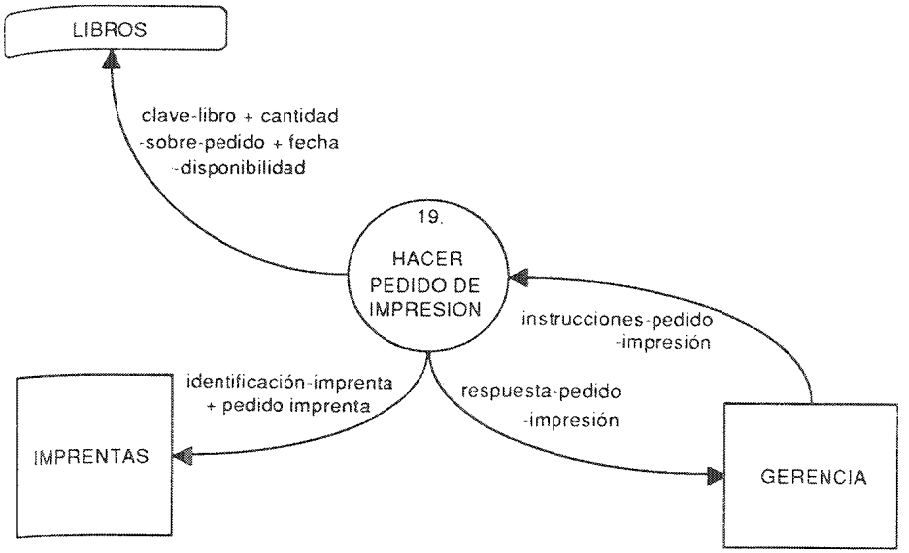


Notas:

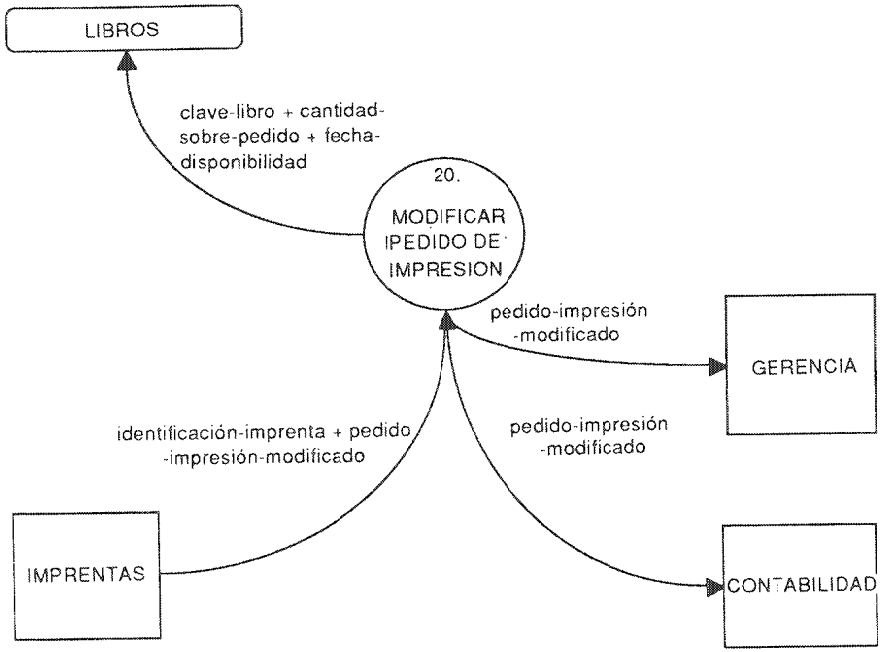
1. Observe que el sistema no procesa la cotización de la imprenta; simplemente la pasa a la gerencia.

2. Como el sistema no lo procesa, no queda claro el por qué sucede este acontecimiento. (Por otro lado, se puede argumentar que el sistema sí tiene la responsabilidad de servir como conducto, o interfase, entre imprentas externas y la gerencia de YOURDON Press.) De cualquier forma, proporciona capacidad futura de hacer pedidos automatizados, basados en los criterios presentes.

Acontecimiento 19: La gerencia autoriza un pedido de impresión.



Acontecimiento 20: La imprenta informa sobre la cantidad exacta de impresos y la fecha de entrega.

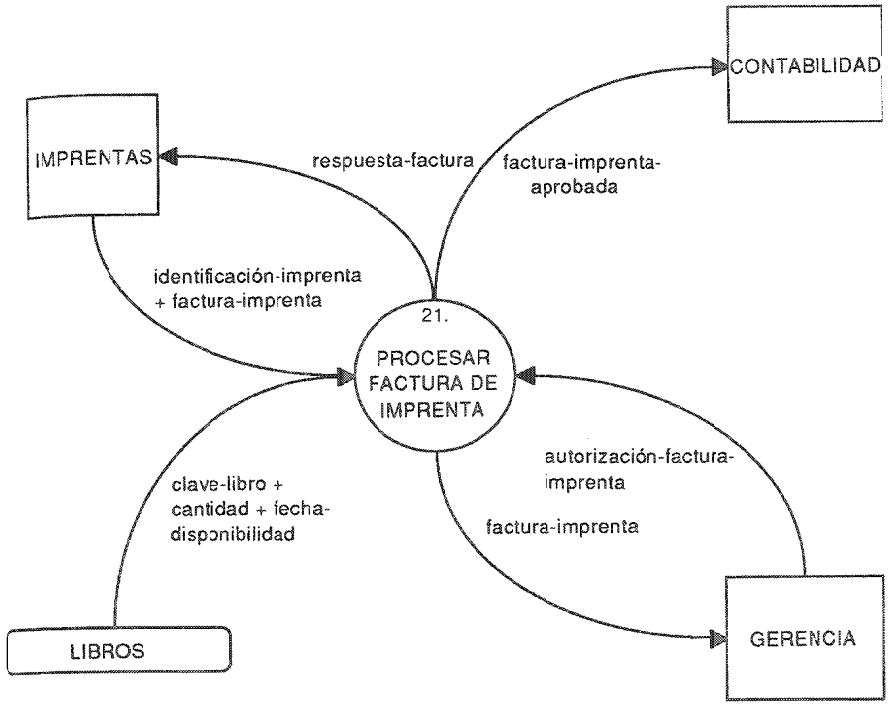


Notas:

1. El departamento de contabilidad necesita la información sobre el pedido modificado del libro para poder estar al corriente en costos unitarios. Esto, junto con el acontecimiento 14, le permite estar al corriente del inventario en forma primero-en-entrar-primero-en-salir (*FIFO,* por sus siglas en inglés).

2. Observe que esto supone que la imprenta no hará cambios substanciales a su cotización original. En la práctica, la imprenta típicamente imprime en exceso o en déficit de la cantidad original en un 1 o un 2 por ciento (por ejemplo, un pedido de impresión de 2000 copias de un libro puede resultar en realidad en la impresión de 2037 libros). Típicamente, la imprenta también espera hasta este momento para cotizar cargos de envío y otros.

Acontecimiento 21: La imprenta envía la factura del trabajo de impresión



Notas:

1. Esto supone que la gerencia responderá a la factura de la imprenta inmediatamente. Observe que YPIS no produce un cheque, sino que simplemente informa a la gerencia de la necesidad de ello.

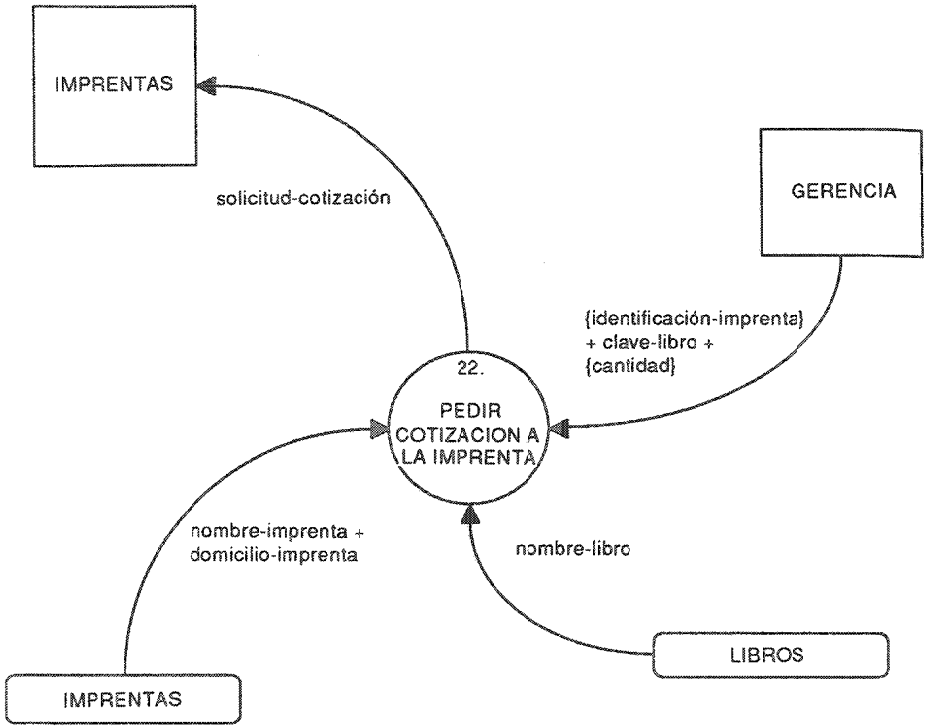
2. Este modelo supone que habrá una factura separada para cada pedido de impresión. También supone que sólo habrá un pedido de impresión a la vez.

3. Observe que la facturación no se hace en forma sincronizada con el envío de libros por la imprenta. Como acontecimiento aparte, la bodega informa al sistema que se recibieron los libros de la imprenta.

4. Esto supone también que el monto de la factura es el mismo que el que se muestra en la estimación modificada (acontecimiento 20).

5. Observe que algunas imprentas insisten en el pago parcial de facturas; este modelo no toma en cuenta eso.

Acontecimiento 22: La gerencia pide cotización de un pedido de impresión.

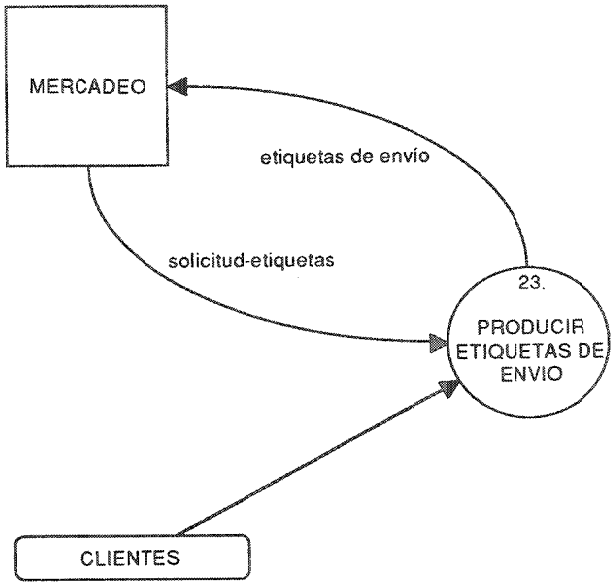


Notas:

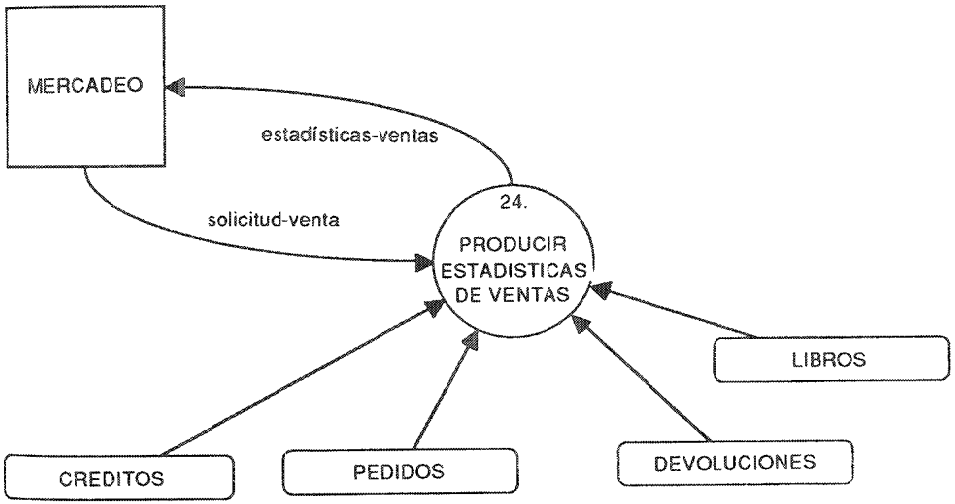
1. Observe que usualmente se solicitan cotizaciones a varias imprentas para poder comparar. Estas se reciben como acontecimientos no sincronizados, y cada una se manda a la gerencia (acontecimiento 18).

2. Observe que las imprentas no responden con una cotización de inmediato; sin embargo, suponemos que algún día lo harán.

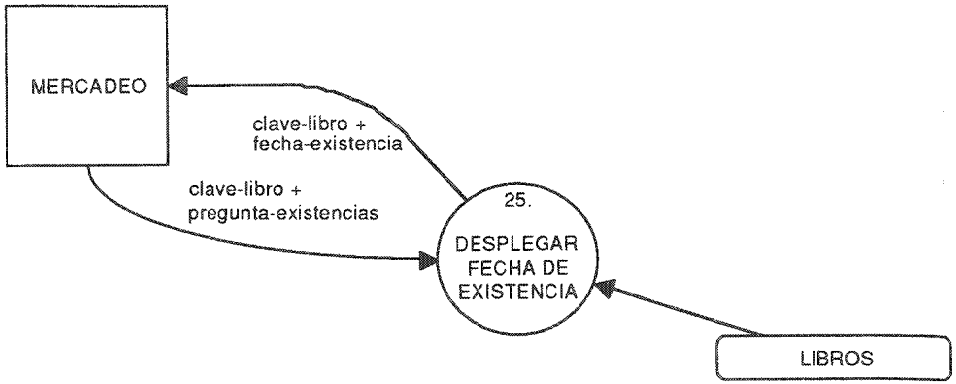
Acontecimiento 23: El departamento de mercadeo pide etiquetas de envío de la base de datos de clientes.



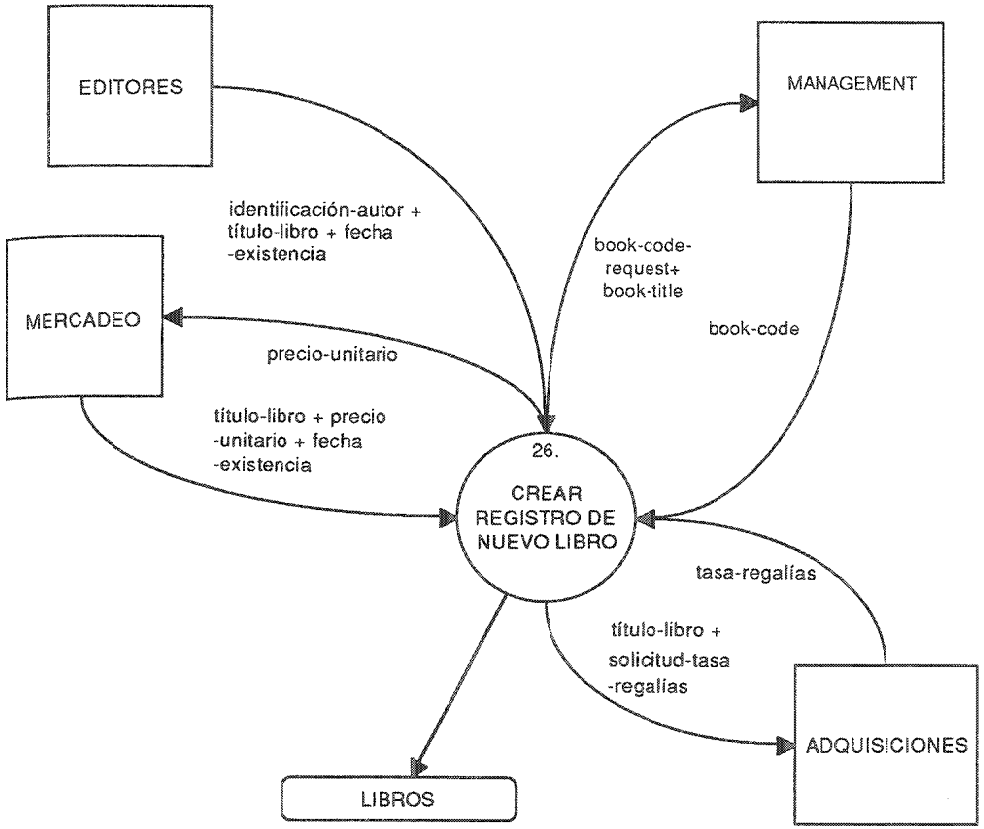
Event 24: Marketing needs statistics on book sales



Acontecimiento 25: El departamento de mercadeo necesita fecha de existencia de nuevos títulos.



Acontecimiento 26: Los editores anuncian un nuevo título (fecha en la que estará listo para impresión).

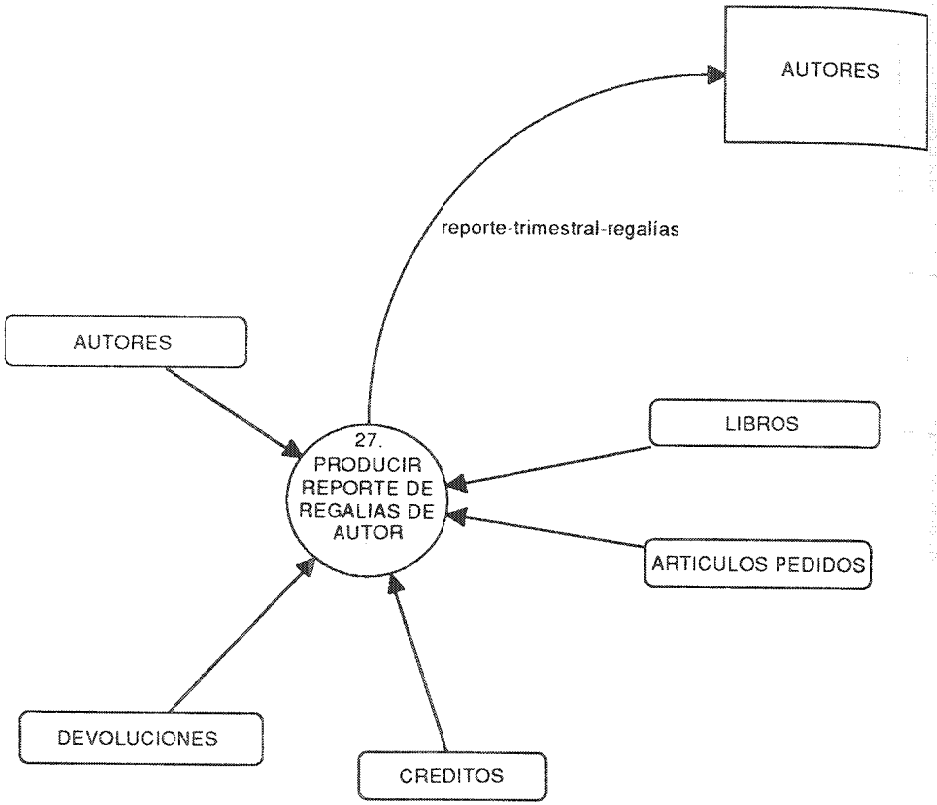


Notas:

1. Al hacer este modelo, vi la necesidad de eliminar algún día libros del sistema. Esto rara vez sucede, pero el departamento de mercadeo llega a decidir cuando "matar" un libro declarándolo agotado (acontecimiento 36).

2. Aunque este acontecimiento realmente crea un nuevo registro de **libro** (el libro no se considera real y parte del sistema a menos que los editores hayan terminado de editarlo y estén por enviarlo a impresión), también debemos crear un registro de autor. Esto se hace en el acontecimiento 34.

Acontecimiento 27: Los autores necesitan un reporte trimestral de regalías.

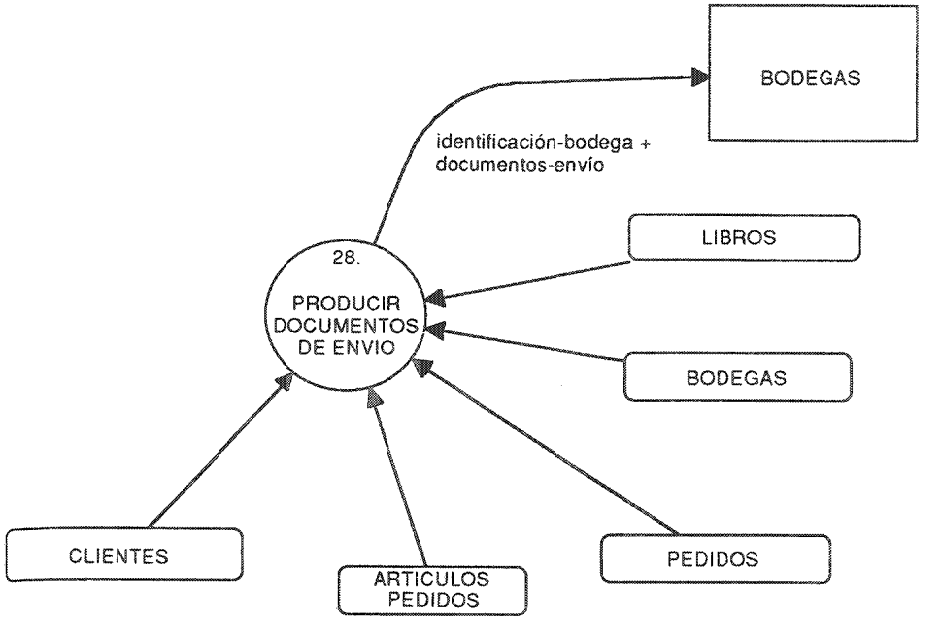


Notas:

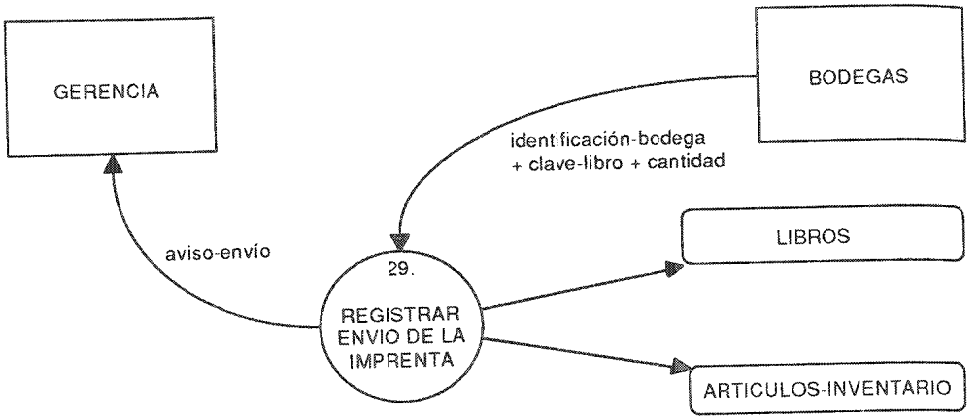
1. Esto es similar al acontecimiento 13, excepto que el reporte se manda a los autores y no a contabilidad.

2. Observe que el departamento de contabilidad requiere ver información detallada, incluyendo una identificación de los clientes a quienes se vendió un libro dado. A los autores no se les da esta información.

Acontecimiento 28: La bodega necesita datos de envío y etiquetas.



Acontecimiento 29: La bodega recibe libros de la imprenta.

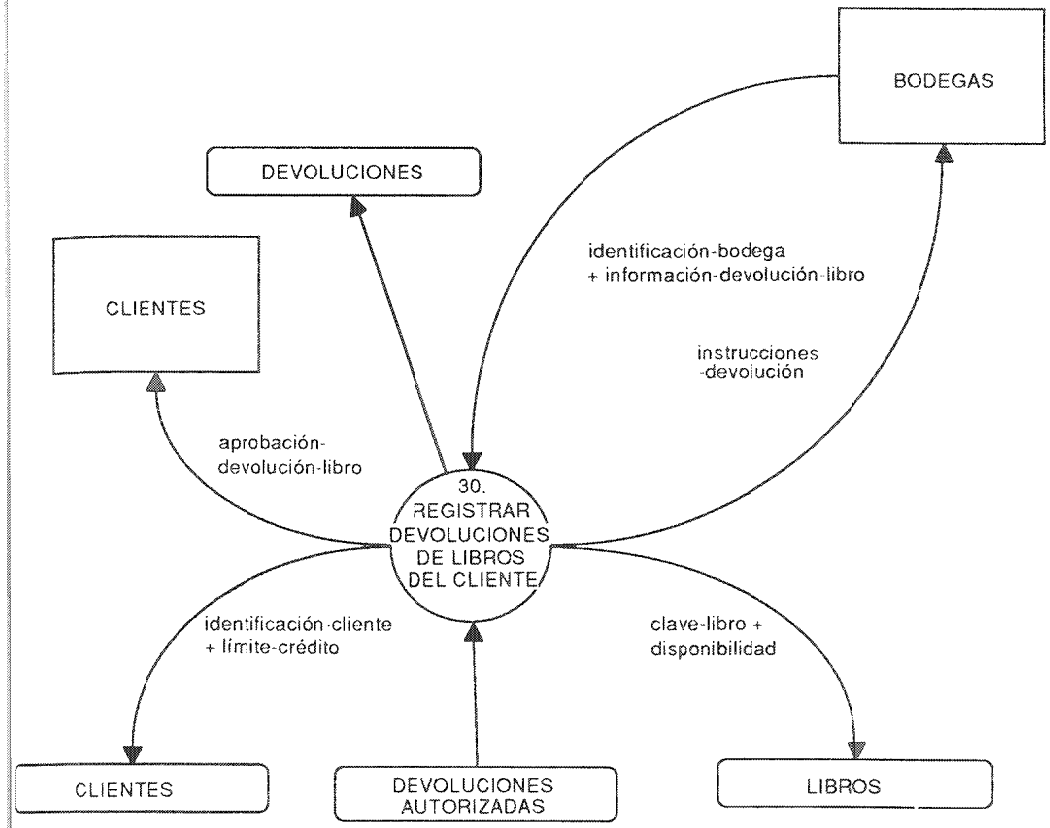


Notas:

1. Esto no toma en cuenta la posibilidad de envíos parciales de la imprenta. Esto sí sucede ocasionalmente: si existe una demanda tremenda de un libro nuevo (o tal vez de una reimpresión de uno existente), la imprenta puede apurar el envío de los primeros centenares de copias (tal vez por vía aerea) y mandar el resto más tarde.

2. Esto supone también que la cantidad recibida por la bodega es la misma que la especificada en el acontecimiento 20.

Acontecimiento 30: La bodega recibe libros de un cliente.

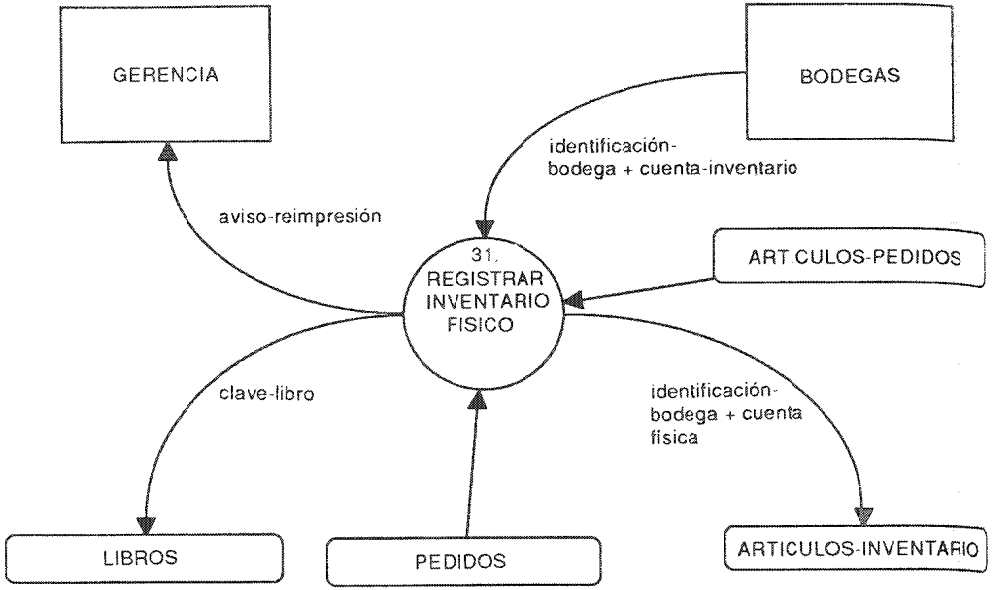


Notas:

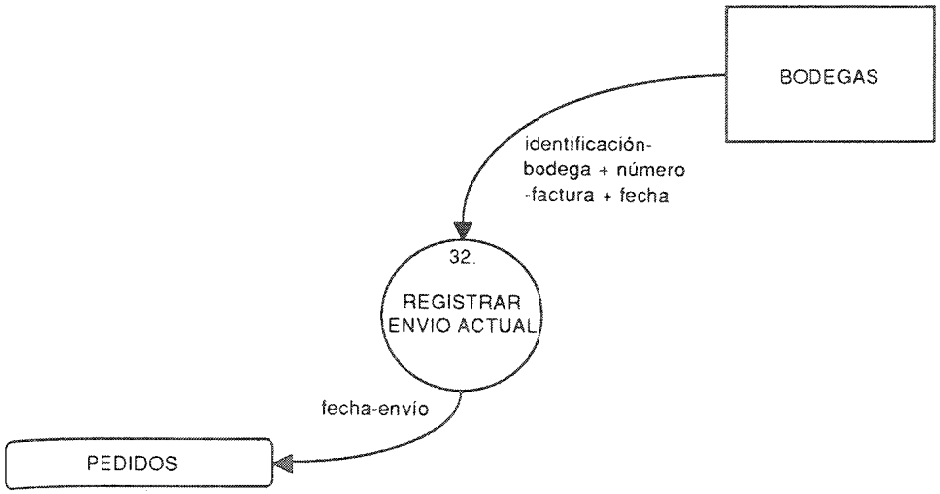
1. El sistema puede dar instrucciones a la bodega de rechazar las devoluciones si no han sido autorizadas. Esto significa que la bodega avisará a la oficina de correos (o a la agencia de transporte que trajo los libros) que el o los paquetes deben ser devueltos al remitente.

2. Observe que en ocasiones es imposible distinguir quién devolvió los libros; es decir, la información que la bodega encuentra en el paquete puede no corresponder a algún cliente conocido.

Acontecimiento 31: La bodega realiza un inventario físico (mensual).



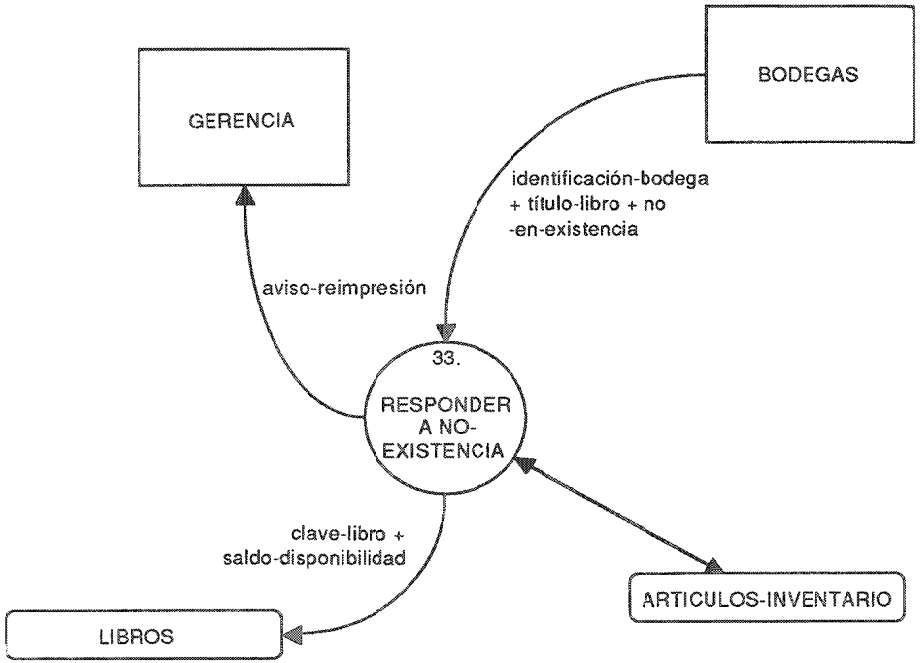
Acontecimiento 32: La bodega envía el pedido al cliente.



Nota:

1. Esto supone que no hay envíos parciales de un pedido para un cliente.

Acontecimiento 33: La bodega anuncia que no hay en existencia copias de un libro dado.

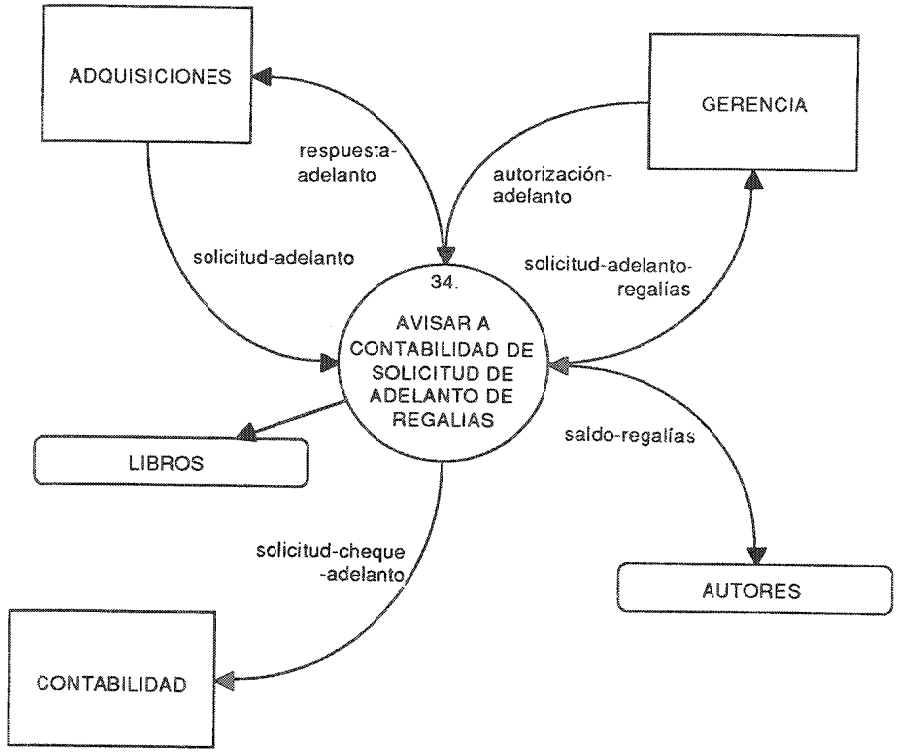


Notas:

1. Observe que la situación de no tener un libro en existencia puede ocurrir porque no haya llegado una reimpresión previamente pedida; o porque hubo un pedido inesperadamente grande; o porque hubo robos en la bodega, etc.

2. Como resultado de la situación de no tener en existencia un libro dado, pudiera ser que no se surtan por el momento pedidos subsecuentes que hayan sido procesados, pero ese es problema de la bodega en cuestión.

Acontecimiento 34: El departamento de adquisiciones anuncia un nuevo proyecto de libro.

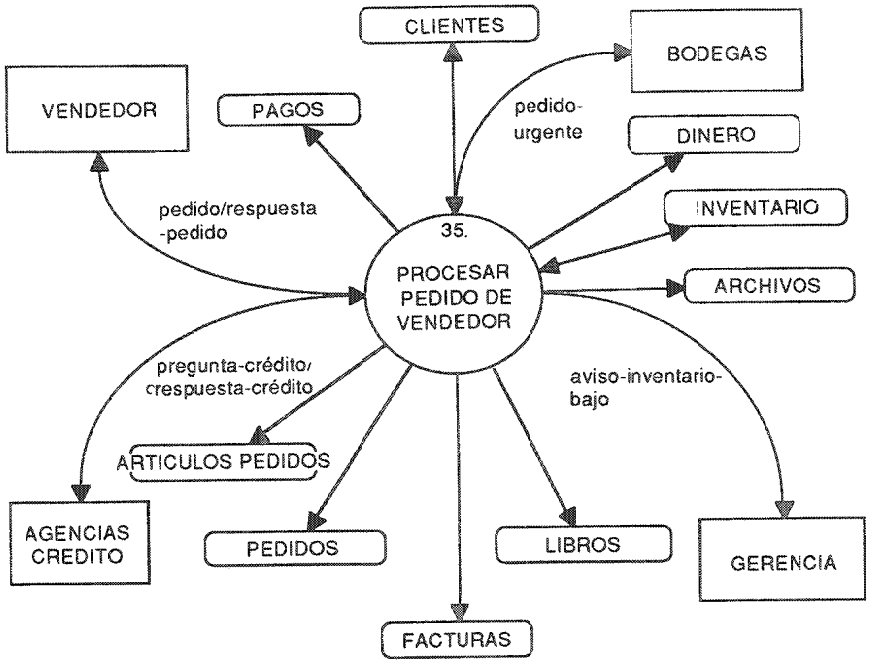


Notas:

1. Esto muestra que el acontecimiento 13 debe leer el almacén **LIBROS** para ver si existe un adelanto excedente.

2. Este acontecimiento también crea un nuevo registro de **autor** si se trata de un autor nuevo.

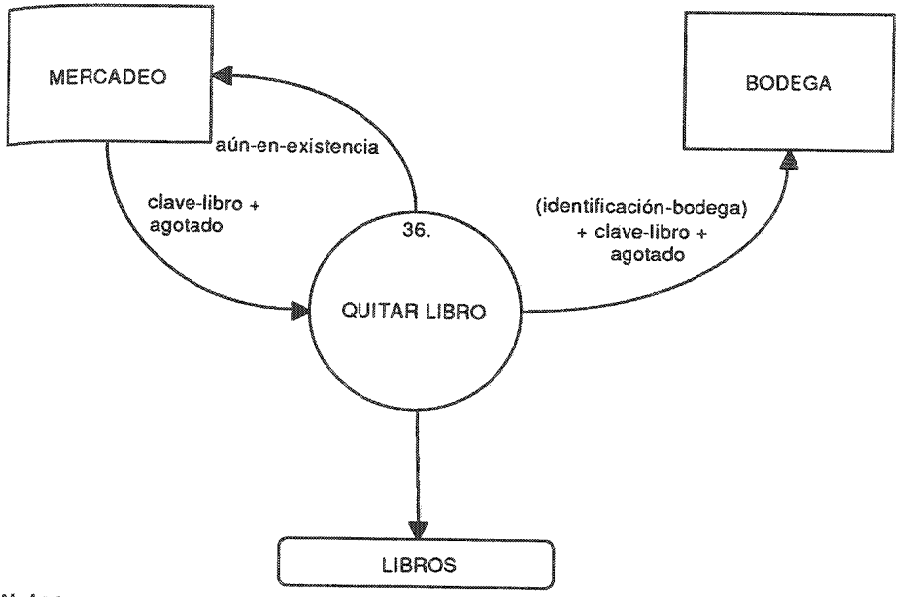
Acontecimiento 35: El vendedor hace un pedido de parte del cliente.



Notas:

1. Observe que éste es igual al acontecimiento 1, excepto que aquí el pedido lo hace un vendedor en lugar de un cliente.

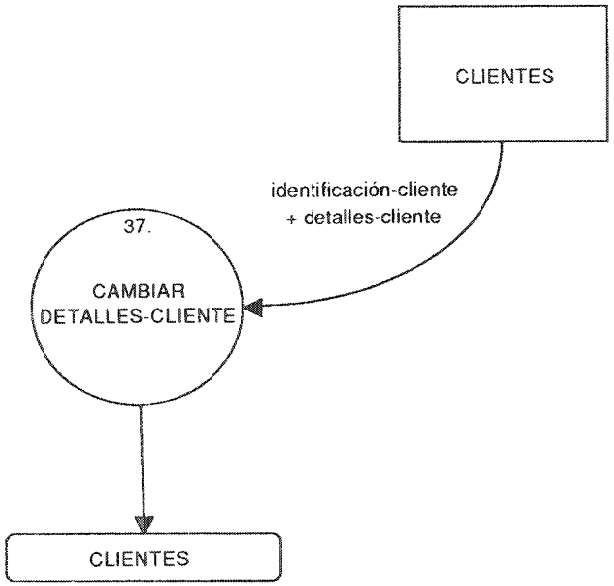
Acontecimiento 36: El departamento de mercadeo declara agotado un libro.



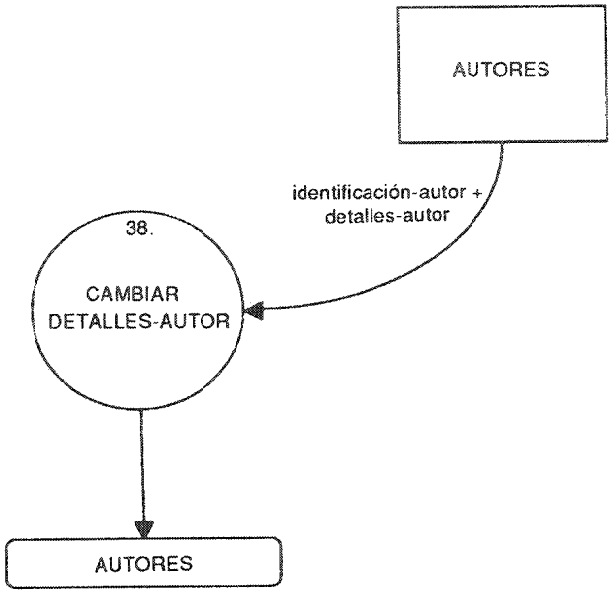
Notas:

1. Esto se puede lograr de manera externa simplemente deteniendo la propaganda de un libro determinado. Tarde o temprano los pedidos bajarán a cero.
2. Cuando se logra como se muestra aquí, excluye pedidos futuros. Se supone que las bodegas se desharán de las existencias restantes para tener más espacio libre en su inventario.
3. En situaciones reales, a menudo se hacen disposiciones para el "resto" de las existencias del libro en este punto. Pudiera permitirse al autor o a alguna tienda de descuento comprar las existencias restantes por, digamos, $0.10 dólares estadounidenses por copia.
4. Observe que el registro de libro no puede eliminarse de LIBROS por lo menos hasta que se hayan producido los reportes mensuales de contabilidad y la declaración trimestral de regalías. Además, puede haber una lista de pedidos que la bodega todavía no haya enviado.
5. Observe que es necesario informar a todas las bodegas cuando ocurre este acontecimiento.
6. Suponemos en este punto que no existen pedidos pendientes con la imprenta: si las ventas han sido tan lentas como para considerar sacar el libro de inventario resulta virtualmente imposible imaginar que se haría un pedido de reimpresión.

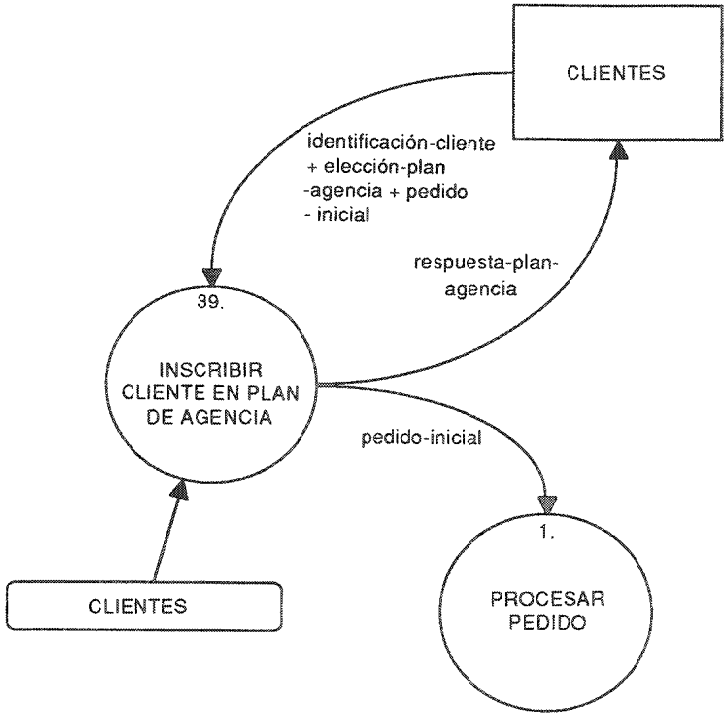
Acontecimiento 37: El cliente anuncia un cambio de domicilio.



Acontecimiento 38: El autor anuncia un cambio de domicilio.



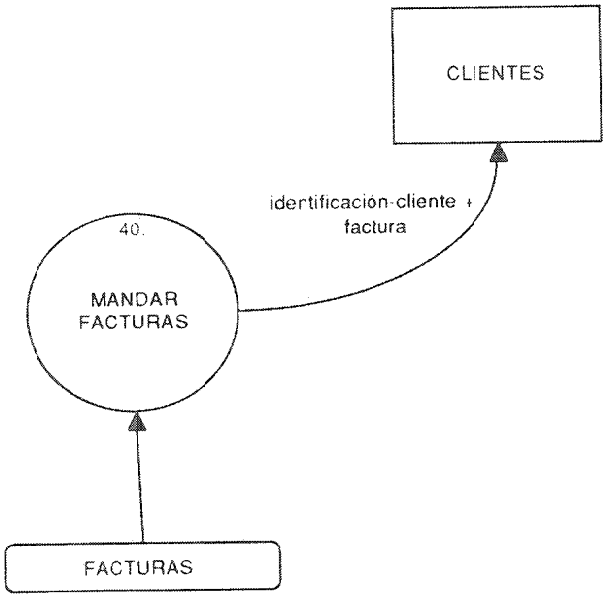
Acontecimiento 39: El cliente elige escoger el plan de la agencia.



Notas:

1. El plan de agencia se conoce bajo una variedad de nombres más en la industria publicitaria (por ejemplo, "plan de envío estándar", "plan de pedido vigente", etc.). Lo usan casi exclusivamente las librerías. Hacen un pedido inicial por un cierto número de libros y acuerdan aceptar un cierto número de copias de cada libro nuevo que publique YOURDON Press.

Acontecimiento 40: Se requiere mandar facturas a un cliente.



F.4.2 El modelo final del comportamiento: diagramas de flujo de datos

El modelo inicial del comportamiento mostrado en las últimas páginas se transformó en un conjunto de DFD por niveles. La nivelación ascendente produjo el diagrama de figura 1 que se muestra a continuación; es tan complejo que no mostré los detalles de todas las entradas y salidas de cada burbuja. Las figuras subsecuentes muestran los acontecimientos que se agruparon. En un caso, un solo acontecimiento (el 26) no se niveló de manera ascendente, y aparece como el proceso 5 de la figura 1. Y en un caso (el acontecimiento 1) se ocupó una nivelación descendente adicional debido a la complejidad del proceso.

Figura 0: El DFD de nivel superior

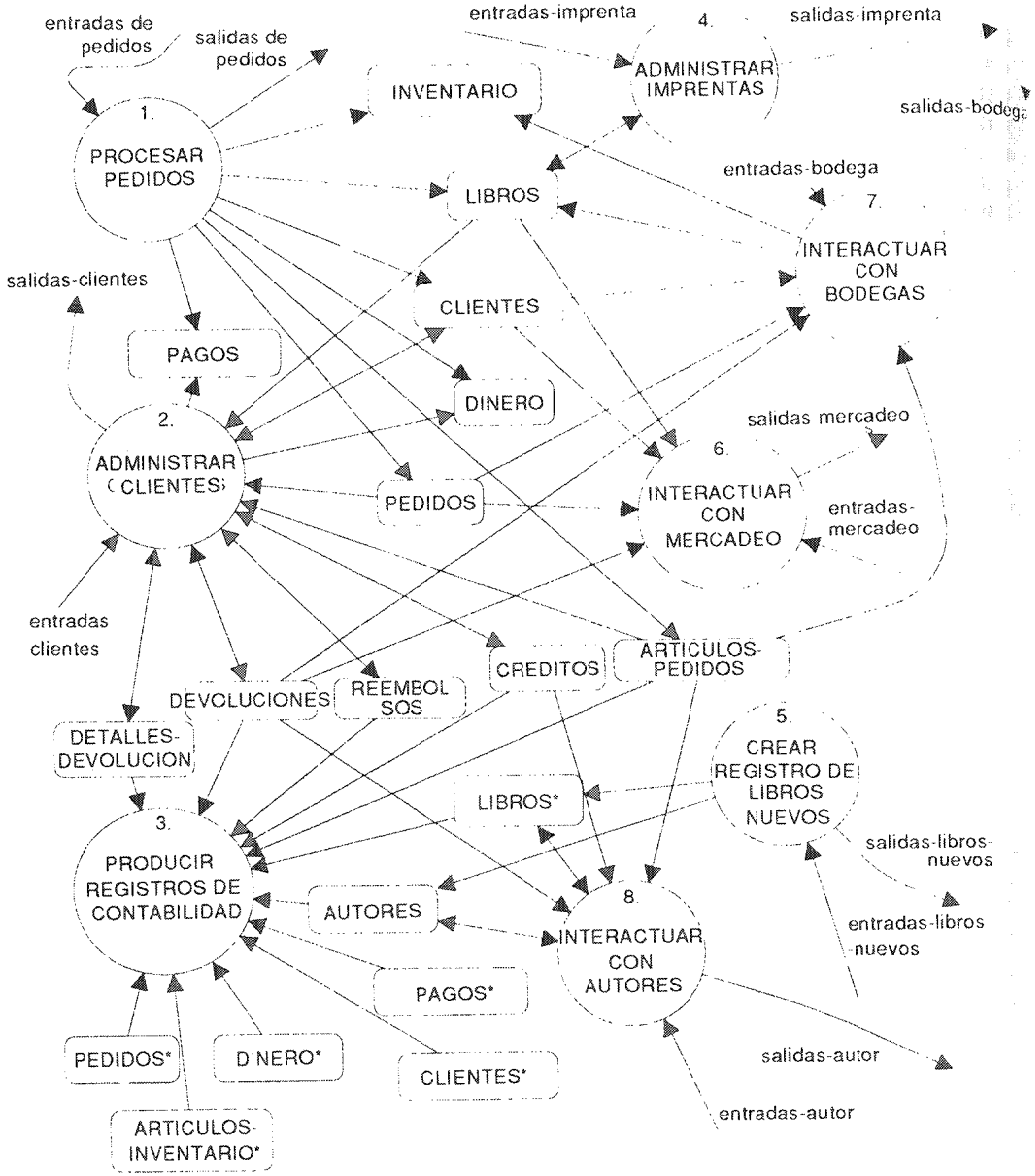


Figure 1: Process orders

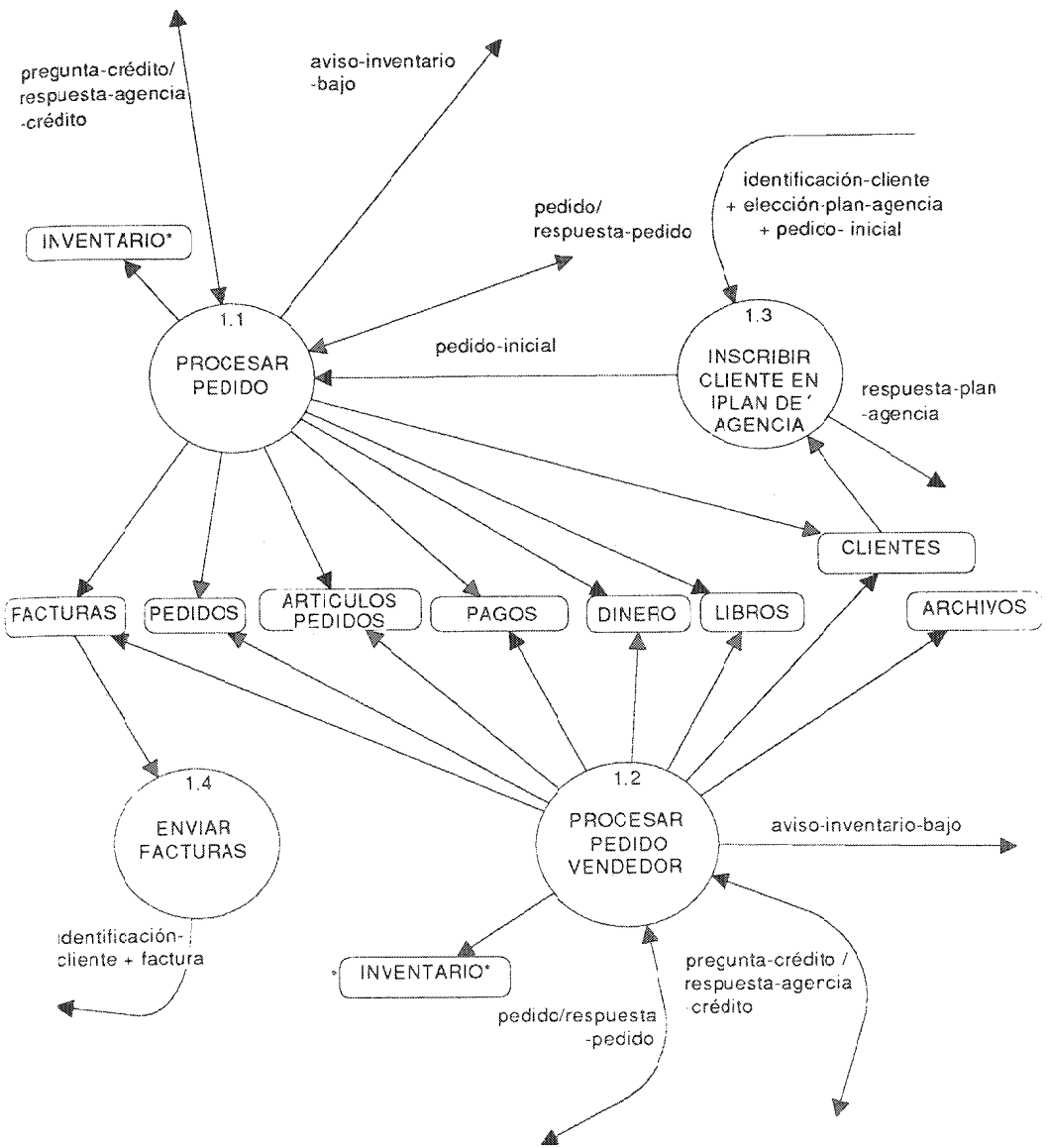


Figura 1.1 Procesar pedidos

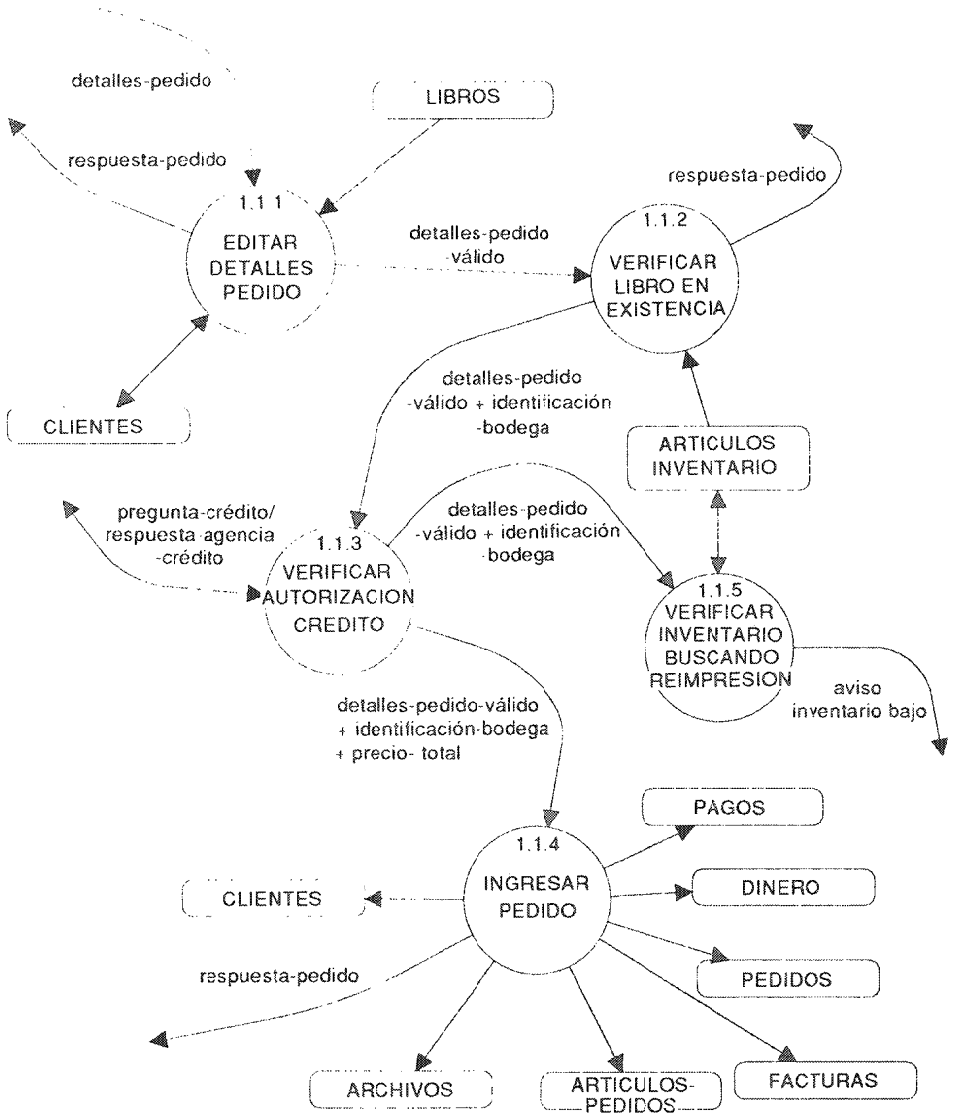


Figura 2: Administrar clientes

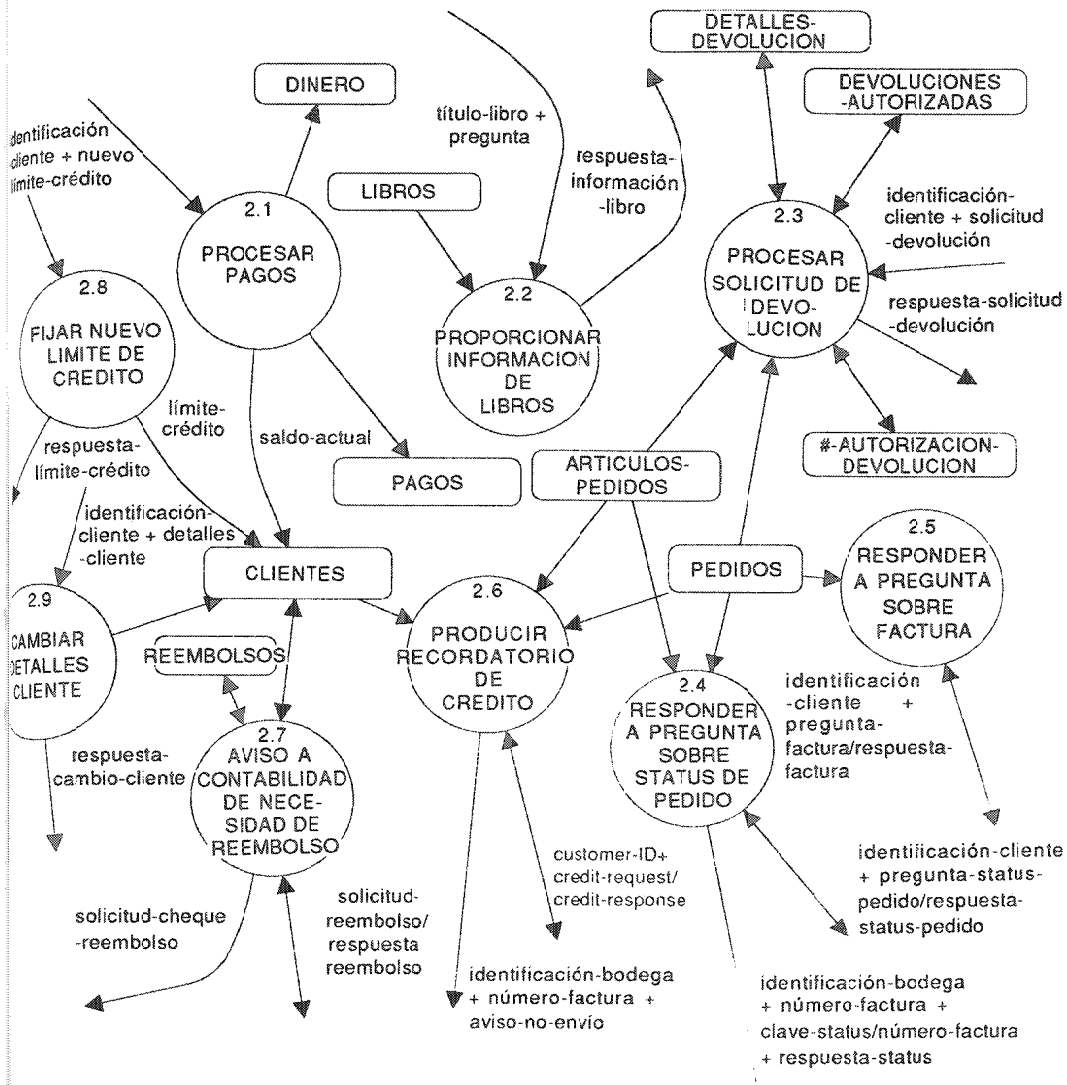


Figura 3: Producir reportes de contabilidad

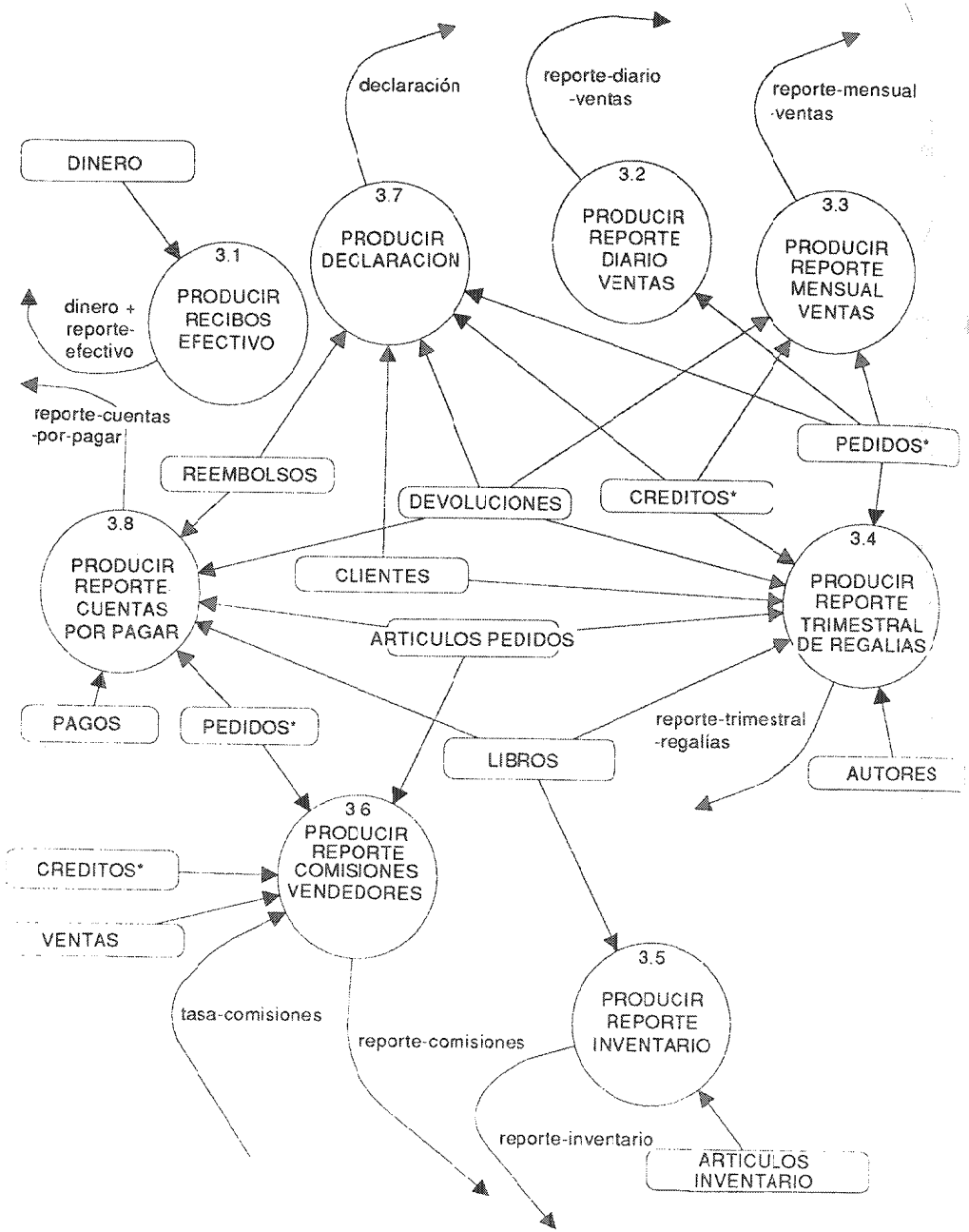


Figure 4: Manage printers

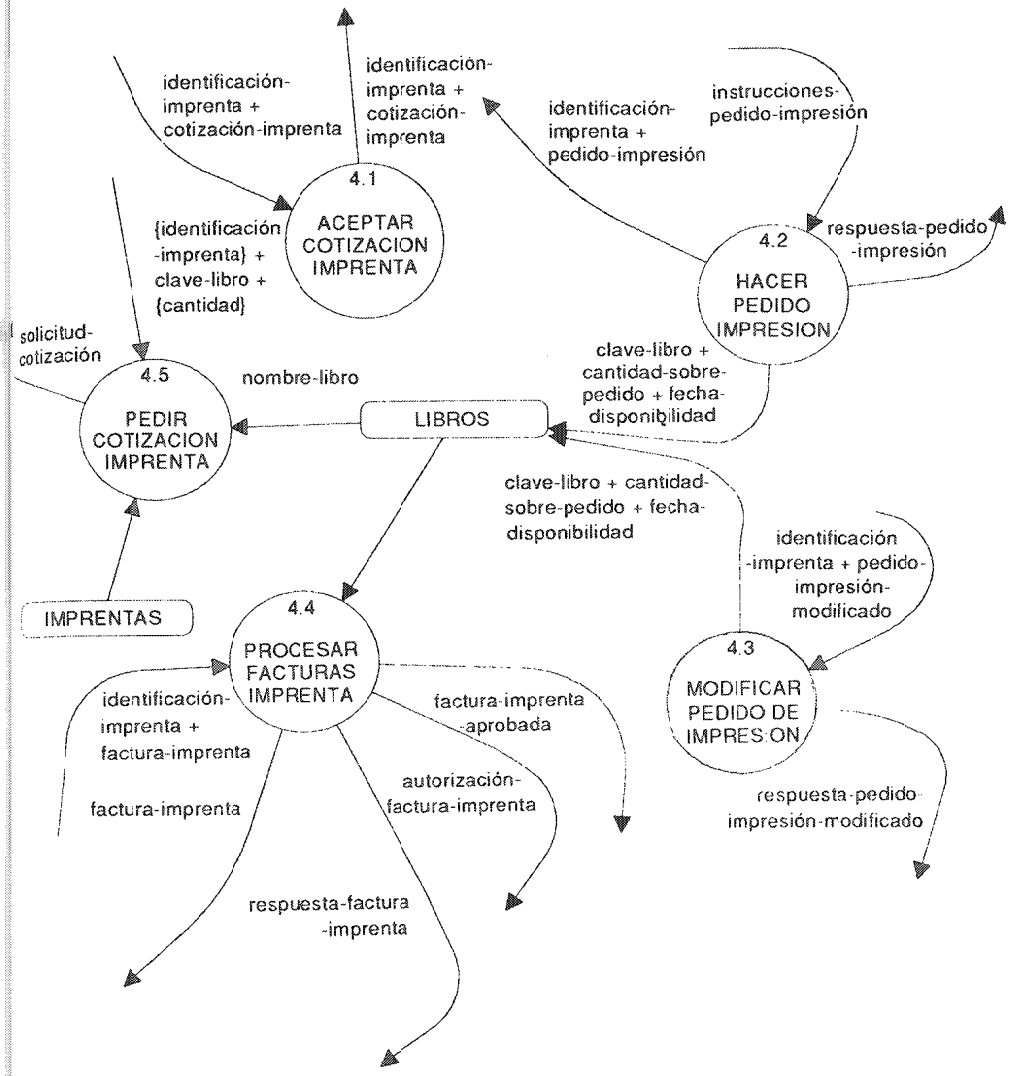


Figura 6: Interactuar con mercadeo

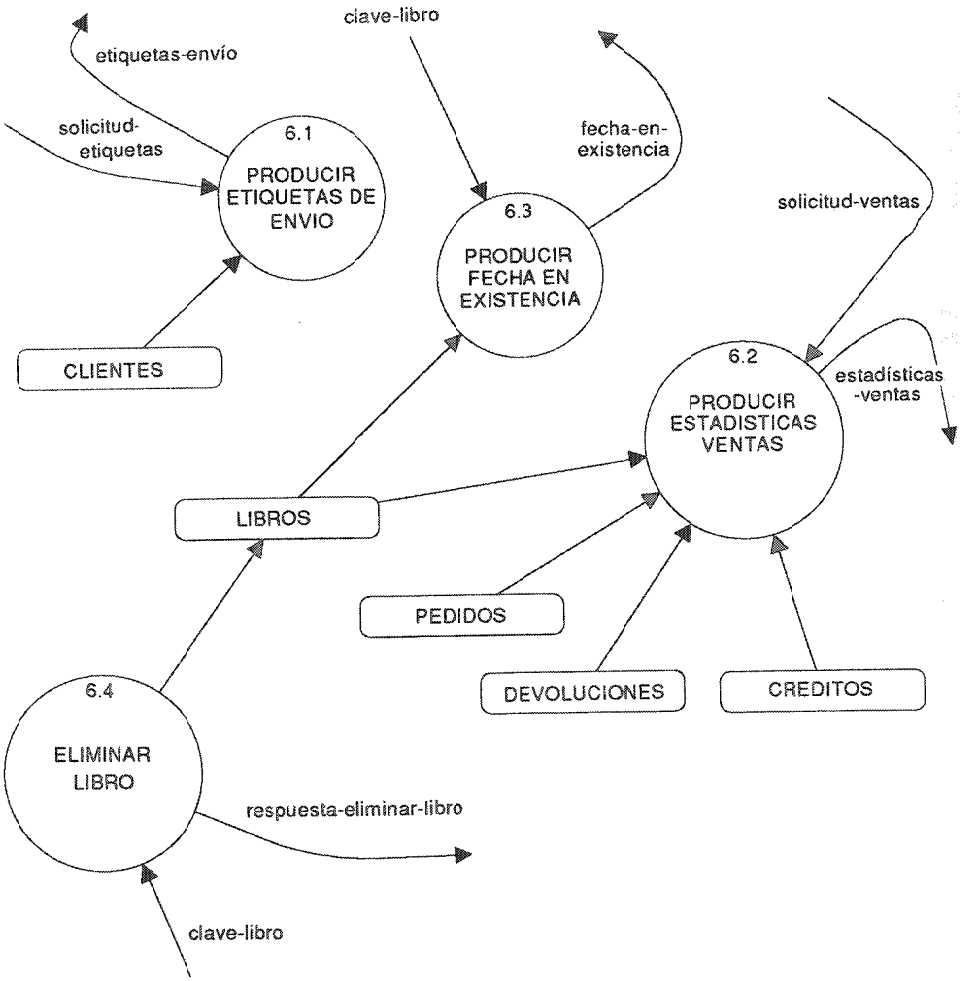


Figura 7: Interactuar con bodegas

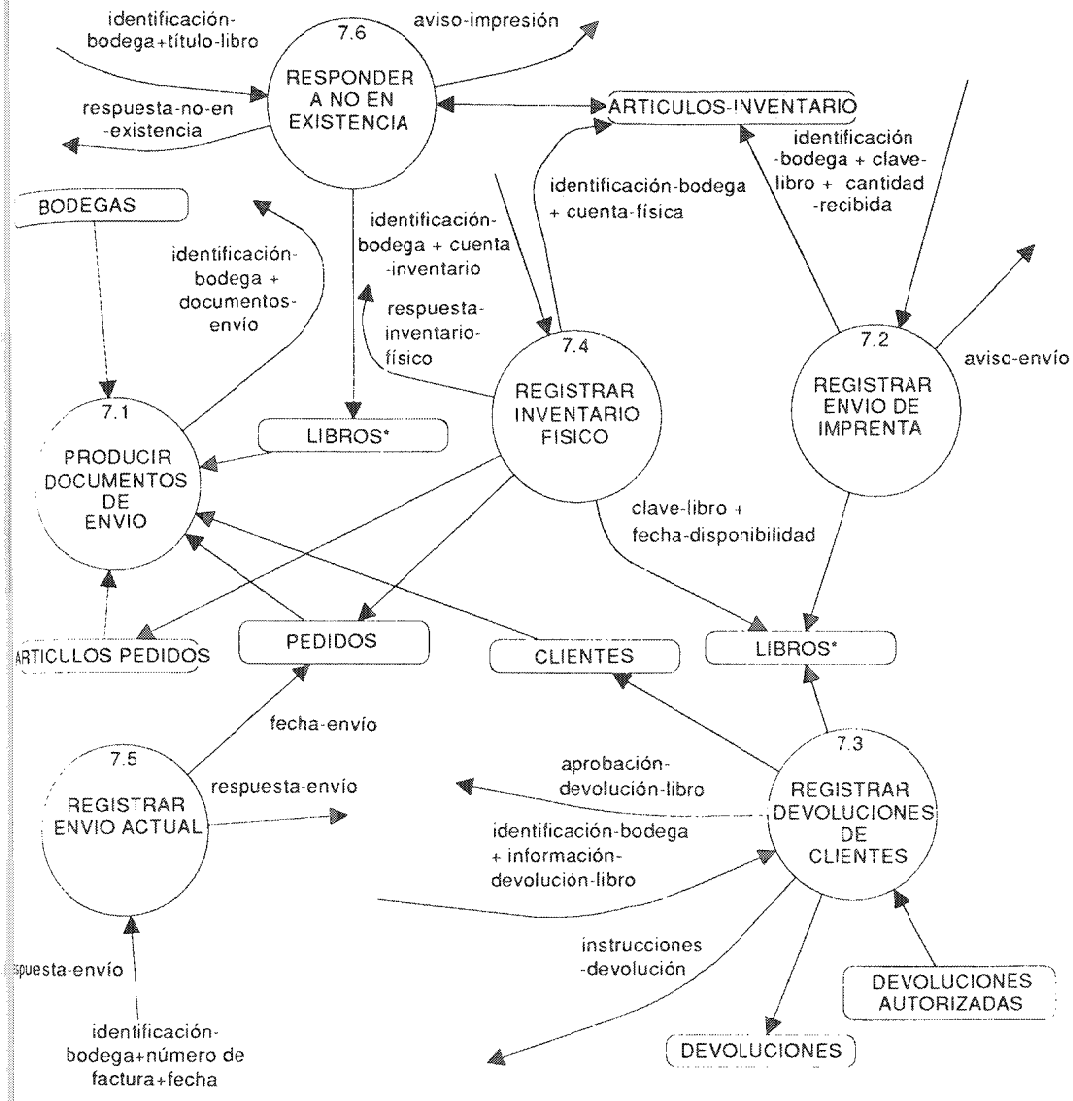
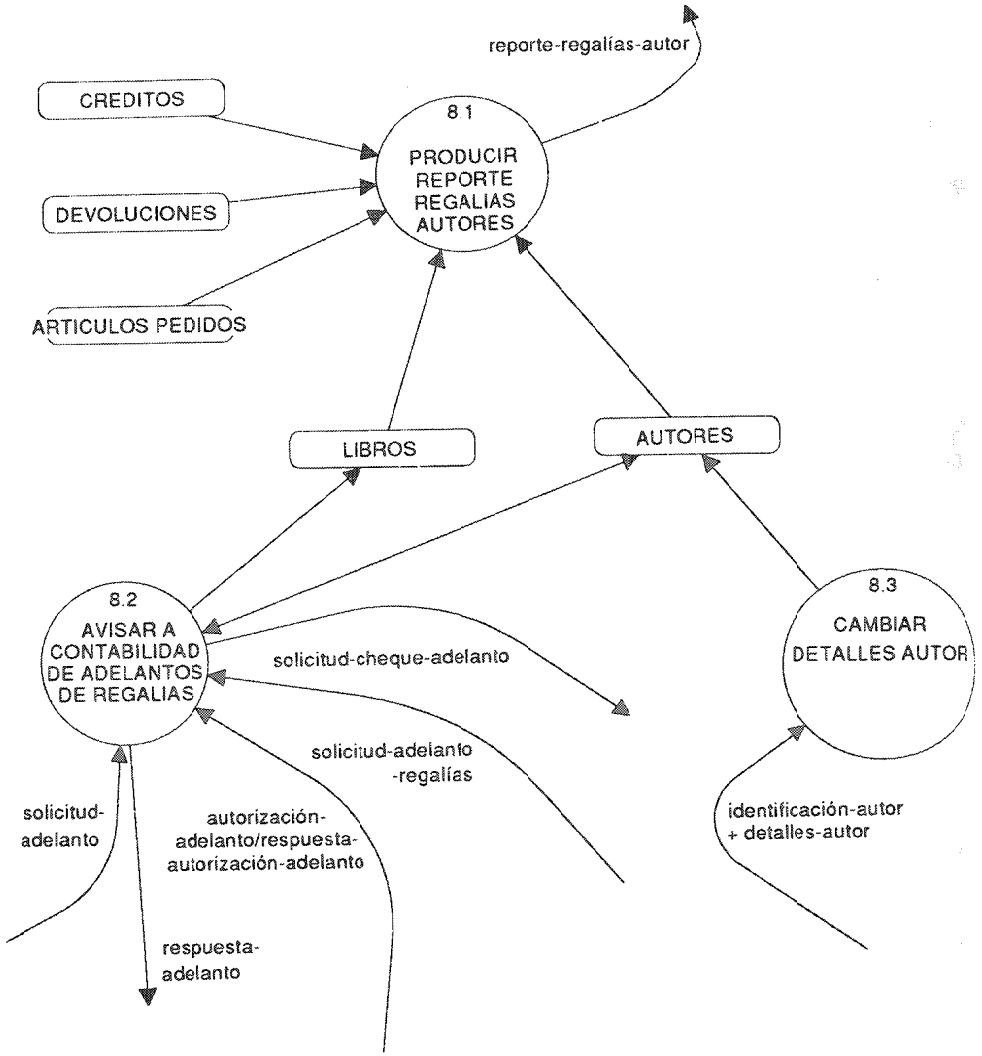


Figura 8: Interactuar con autores



F.4.3 El diccionario de datos

El diccionario de datos está organizado de la forma descrita en el capítulo 10. Los términos que se autodefinen (es decir los datos cuyos significados son lo suficientemente conocidos como para que no se requiera una definición explícita) contienen ** como definición. Vea, por ejemplo, la definición de "país" y de "estado".

adelanto = *cantidad de dinero solicitada como adelanto de las regalías por derechos de autor*
unidades: dólares

a-fecha = *fecha en la que se hizo el inventario físico*
**

apellido = *apellido de una persona*
**

aprobación-devolución-libro = *respuesta a un cliente cuando la bodega notifica al sistema que se recibieron los libros devueltos*
["Esta devolución no está autorizada" | "Esta devolución sí procede"]

artículo-devolución = *información sobre una o más copias de un mismo título que el cliente desea devolver*
clave-libro + cantidad-a-devolver

ARTICULOS-INVENTARIO = {artículo-inventario}

artículo-inventario = *grupo de libros, del mismo título, localizados en una sola bodega*
@clave-libro + @identificación-bodega + cantidad-inventario

ARTICULOS-PEDIDOS = {artículo-pedido}

artículo-pedido = @número-factura + @clave-libro + cantidad-pedida + precio-unitario + descuento

autor = *información almacenada acerca de cada autor*
@identificación-autor + detalles- autor + balance-regalías

AUTORES = {autor}

#-AUTORIZACION-DEVOLUCION = *almacén que se usa para seguir la pista del siguiente número autorizado de devolución*
#-autorización-devolución

#-autorización-devolución = *número secuencial que se usa para identificar un conjunto específico de libros devueltos que se hayan autorizado*
{dígito-numérico}

autorización-factura-imprenta = *respuesta de la administración luego de revisar una factura de imprenta*
["SI" | "NO"]

aviso-envío = *aviso de la bodega cuando se recibe un pedido de impresión de la imprenta*
["No existe tal libro" | "Se recibió de la imprenta" + clave-libro + cantidad-recibida]

aviso-inventario-bajo = *mensaje enviado a la administración cuando el sistema descubre que el inventario total de un libro determinado ha descendido por debajo de un cierto nivel prescrito*
clave-libro + total-en-existencia + "hora de reimprimir"

cantidad-a-devolver = *número de copias de un solo libro que un cliente desea devolver a cambio de crédito*
**

cantidad-devuelta = *cantidad de copias de un libro que un cliente devolvió a una bodega*
**

cantidad-dinero = *cantidad de dinero recaudada en un solo pago*
unidades: dólares

cantidad-inventario = *cuenta del número de libros del mismo título que se tienen en una sola bodega*
**

cantidad-libros-a-crédito = *número de copias del libro para el cual se está solicitando crédito*
**

cantidad-modificada = *modificación del número de libros que la imprenta imprimirá como parte de un pedido*
**

cantidad-pedida = *número de copias de un libro que se pidieron*
**

cantidad-recibida = *número de copias de un libro que se recibieron de la imprenta como resultado de un pedido de impresión*
**

cantidad-seleccionada = *número de copias de un solo libro que se debe escoger para satisfacer los pedidos de un día en una bodega*
**

cantidad-sobre-pedido = *número de copias de un libro que actualmente conforman un pedido a una imprenta*
**

carácter-alfabético = *una letra del alfabeto*
**

carácter alfanumérico = *un número, una letra o un signo de puntuación*
[carácter-alfabético | dígito- numérico | signo-puntuación]

carços-envío = *cantidad cobrada por el envío de un libro a un cliente. Puede ser una cantidad estándar, por ejemplo, US $1.50, o puede estar basado en los cargos reales de transporte*
unidades: dólares; escala: 0 - 100

clave-libro = *clave numérica que identifica a cada libro*
1{dígito-numérico}

cliente = *un cliente de YOURDON Press*
@identificación-cliente +(nombre-compañía) + nombre-cliente + domicilio-cliente + saldo-actual +límite-crédito + nivel-plan-agencia

CLIENTES = {cliente}

código-año = *los últimos dos dígitos del año actual, p. ej., "92" si el año actual es 1992*
2{dígito numérico}2

código-postal = *código postal de EUA, de Canadá, o de Gran Bretaña*
**

cotización-imprenta = *cotización dada por una imprenta, oferta de imprimir cierto número de libros a un cierto precio*
clave-libro + precio-imprenta

crédito = *crédito individual que se le otorga a un cliente debido a algún problema con algún pedido* @número-factura + identificación-cliente + fecha-crédito + clave-libro + cantidad-libros-a-crédito + monto-de-crédito

CREDITOS = {crédito}

créditos-venta = *créditos asociados con un mismo libro durante un período específico de tiempo* *unidades: dólares*

cuenta-física = *cuenta del número de copias de un mismo título que se encuentra durante el inventario físico de una bodega* **

cuenta-inventario = *información acerca de una cuenta física de inventario que se haya hecho en una bodega* {detalle-inventario} + a-fecha

descuento = *porcentaje de descuento que se otorga en un pedido de volumen, expresado como fracción del precio a pagar, es decir, un 10 por ciento de descuento se expresaría como 0.90* *escala: 0 - 1.00*

detalle-inventario = *cuenta de inventario físico para un solo título* clave-libro + cuenta-física

detalles-autor = título + nombre + apellido + domicilio + ciudad + código postal + (país) + número telefónico

detalles-cliente = *información proporcionada por un cliente para cambiar información ya existente en su registro por ejemplo, nuevo domicilio de cliente* (nombre-cliente) + (nombre-compañía) + (domicilio-cliente)

detalles-devolución = ** {artículo-devolución}

detalles-pago = *información detallada respecto a un artículo o una factura que se está pagando* {número-factura} + monto-total

detalles-pedido = *datos que se manejan para la construcción de un pedido válido y su almacenamiento en PEDIDOS* identificación-cliente-posible + {artículo-pedido} + tasa-impuestos-venta + cargos-envío + tipo-pago + (pago-de-pedido) + (número-tarjeta-crédito) + lista-espera-OK + tipo-pedido

detalles-pedido-válido = *datos con los cuales se construirá un pedido válido para ser ingresado al almacén de PEDIDOS* identificación-cliente + {artículo- pedido} + tasa-impuestos-venta + cargos-envío + tipo-pago + (pago-de- pedido) + (número-tarjeta-crédito) + lista-espera-OK + tipo-pedido

devolución = *información acerca de un grupo de libros que se devolvieron y que YOURDON Press ha aceptado* fecha-devolución + clave-libro + cantidad-devuelta + valor-devolución

DEVOLUCIONES = {devolución}
devolución-autorizada

= *información sobre algún grupo de libros que YOURDON Press haya autorizado que un cliente devuelva a cambio de crédito* @#-autorización-devolución + detalles-devolución

DEVOLUCIONES-AUTORIZADAS = {devolución-autorizada}

devoluciones-ventas = *devoluciones por un libro determinado durante un período específico de tiempo* *unidades: dólares*

dígito-numérico = *un simple y vulgar dígito* **

DINERO = {dinero}

dinero = *información referente a cheques, efectivo u otros* @fecha-del-dinero + @identificación- cliente + {número-factura} + cantidad-dinero

documentos-envío = *lista de selección y etiquetas de envío que se mandan a la bodega para que puedan seleccionar suficientes copias de cada libro y enviar los pedidos del día, además de una copia de cada pedido para que la bodega pueda saber cuántos libros empacar para cada cliente*
{identificación-bodega + lista-de- selecciones + {etiqueta-envío} + {pedido}}

domicilio-cliente = *domicilio de cobro: a dónde mandar la factura del cliente*
domicilio + ciudad + estado + código-postal + (país)

domicilio-imprenta = *domicilio en donde se puede localizar al dueño de la imprenta*
domicilio + ciudad + estado + código-postal

elección-plan-agencia = *elección del cliente sobre cual nivel de plan de agencia adoptar*
escala: 0-4

estadísticas-venta = *reporte al departamento de mercadeo sobre las ventas netas de libros para un periodo de tiempo dado*
{clave-libro + ingresos-venta + devoluciones-ventas + crédito venta}

estado = *estado o provincia en un domicilio*
**

etiqueta-envío = *etiquetas de envío para los pedidos del día a la bodega*
nombre-cliente + domicilio-cliente + número-factura

etiquetas-correo = *etiquetas de envío producidas por el departamento de mercadeo*
{cliente}

factura = *información contenida en una factura de YOURDON Press*
@número-factura + nombre-cliente + domicilio-cliente + pedido

FACTURAS = {factura}

factura-imprenta = *factura recibida de una imprenta por concepto de un pedido de impresión*
clave-libro + monto-de-factura

factura-imprenta-aprobada = *factura de imprenta que haya sido aprobada por la administración*
clave-libro + monto-de-factura

fecha-crédito = *fecha en la cual se otorgó un crédito*
**

fecha-devolución = *fecha en la que se devolvió un lote de libros*
**

fecha-disponibilidad = *fecha en la que se espera que el pedido de impresión de algún libro llegue a una bodega*
**

fecha-envío = *fecha en la que la bodega envía un pedido*
**

fecha-modificada = *nueva fecha fijada por la imprenta para la entrega de un lote de libros que está imprimiendo*
**

fecha-pago = *fecha en la que se realiza un pago*
**

fecha-reembolso = *fecha en la que se aprobó el reembolso*
**

fecha-saldo-actual = *fecha en la cual se hizo el cómputo del balance actual del cliente (usualmente durante la producción de su estado de cuenta)*
**

fecha-venta = *fecha después de la cual deben incluirse en el reporte estadístico de ventas todos los pedidos, créditos y devoluciones*
**

identificación-autor = *identificación de cada autor de YOURDON Press No se utilizan los números de Seguro Social pues no todos los autores son ciudadanos de los EUA*
apellido + nombre

identificación-bodega = *identificación de las diversas bodegas donde se almacenan libros de YOURDON Press* ("NYC" | "LON" | "DC" | "SFO" | "YONKERS" | "OTTAWA")

identificación-cliente = *identificación de un cliente de YOURDON Press; los clientes desconocidos o no identificados se conocen como "efectivo no aplicado", sobre todo en lo referente a pagos* [{dígito-numérico} | "efectivo no aplicado"]

identificación-cliente-posible = *información sobre un cliente cuando por primera vez hace un pedido* {identificación-cliente + (nombre- cliente) | "nuevo" + nombre-cliente + (nombre-compañía) + domicilio-cliente}

identificación-imprenta = *clave única que identifica a la imprenta* {dígito-numérico}

identificación-vendedor = *identificación de un vendedor de YOURDON inc.* **

imprenta = *información acerca de cada una de las imprentas con las que YOURDON Press tiene tratos* @identificación-imprenta + nombre-imprenta + domicilio-imprenta

IMPRENTAS = {imprenta}

impuestos-venta = *impuestos local y estatal asociados con un pedido* *unidades: dólares*

indicador-agotado = *indicador binario sobre si un libro ya no se tiene en existencia para que se rechacen pedidos subsecuentes (si los hay)* [SI | NO]

información-devolución-libro = *información acerca de un grupo de libros que algún cliente haya devuelto a la bodega* (identificación-cliente) + (nombre-cliente) + {clave-libro + cantidad-devuelta} + #-autorización- devolución

ingresos-venta = *ingresos netos de las ventas de un mismo libro durante un periodo de tiempo específico* *unidades: dólares*

instrucciones-devolución = *instrucciones a la bodega sobre qué hacer con un lote de libros que el cliente haya devuelto* ("No es posible identificar al cliente; acepte de todos modos los libros" | "Esta devolución no está autorizada; favor de regresarla" | "No existe tal libro" | "Se autoriza la devolución")

instrucciones-pedido-de-impresión = *instruciones de la administración para reimprimir un libro* identificación-imprenta + clave-libro + tiraje + fecha-disponibilidad

inventario-total-libros = * número total de libros de un título dado, en todas las bodegas de YOURDON Press* **

libro = *información almacenada acerca de un libro de YOURDON Press*

@clave-libro + título + identificación-autor + total-en-existencia + cantidad-sobre-pedido + tasa-regalías + indicador-agotado + nivel-para-reorden

LIBROS = {libro}

límite-crédito = *monto del crédito que se le dará a un cliente para pedidos que no se pagan por adelantado* *unidades: dólares; escala: 1 -10,000*

lista-de-selecciones = *indicación del número de copias de cada libro que se debe escoger en una bodega para satisfacer los pedidos de un día* {título-libro + cantidad-seleccionada}

lista-espera-OK = *indicación de si un cliente hará el pedido aunque actualmente no haya suficientes libros en existencia* ["Sí" | "No"]

monto-de-comisión = *monto de la comisión que se le paga a un vendedor por cada pedido individual; se calcula en el proceso 3.6*
**

monto-de-crédito = *cantidad de dinero por la cual se da crédito*
unidades: dólares

monto-de-crédito-solicitado = *cantidad de dinero solicitada para crédito*
unidades: dólares

monto-de-factura = *cantidad de dinero cobrado por una imprenta en la factura por concepto de un pedido de impresión*
unidades: dólares

monto-del-reembolso = *cantidad de dinero que se le debe reembolsar a un cliente*
unidades: dólares

nivel-plan-agencia = *código para indicar el nivel de "pedidos vigentes" escogido por el cliente para los próximos libros de YOURDON Press*
escala: 0-4

nombre = *el nombre de una persona*
**

nombre-cliente = título + nombre + apellido

nombre-compañía = *nombre de una compañía u organización*
**

nombre-imprenta = *nombre de la compañia impresora*
**

nombre-vendedor = *nombre de un vendedor de YOURDON inc.*
**

número-factura = *número exclusivo asignado a cada factura; un número de factura típico es B88-5067*
"B" + código-año + {dígito-numérico}

número-tarjeta-crédito = *número de tarjeta de crédito dado por el cliente si desea cargar la cuenta de un pedido de libros a su tarjeta*
**

número-telefónico = *un número telefónico*
**

pago = *pago realizado por concepto de un pedido o para pagar una factura*
(identificación-cliente) + fecha-pago + detalles-pago

PAGOS = {pago}

pago-de-pedido = *pago que el cliente adjunta a su pedido*
unidades: dólares

país = *nombre de un país, por ejemplo, "Canadá"*
**

pedido = *un pedido de algún libro de YOURDON Press*
@número-factura + identificación-cliente + fecha-pedido + {artículo-pedido} + cargos-envío + impuestos-venta + fecha-envío + (identificación-vendedor) + total- del-pedido + identificación-bodega

PEDIDOS = {pedido}

pedido-impresión = *pedido de impresión que se le da a una imprenta*
clave-libro + tiraje

pedido-impresión-modificado = *modificación del pedido de impresión de una imprenta. Usualmente involucra cambios menores en la cantidad a imprimir*
clave-libro + cantidad-modificada + fecha-modificada

precio-imprenta = *precio de una cotización de la imprenta*
**

precio-unitario = *precio que se cobra por una copia de un libro de YOURDON Press en un pedido individual; observe que puede no ser el mismo que el precio unitario "estándar" o el de "venta" que se anuncia*
unidades: dólares

pregunta-crédito = *solicitud de autorización a una agencia de tarjetas de crédito para la compra de un pedido de libros*
"Solicitud de autorización" +
número-tarjeta-crédito + total-del-pedido

razón-crédito = *motivo que tiene el cliente para solicitar crédito*
["Pago excesivo" | "retraso excesivo" | "envío insuficiente" | "libros dañados"]

reembolso = *información acerca de un reembolso*
@fecha-reembolso + @identificación-cliente + monto-del-reembolso

REEMBOLSOS = {reembolso}

regalías-artículo = *utilidades por derechos de autor obtenidas por la venta de una o más copias del mismo libro en un solo pedido*
unidades: dólares

reporte-comisiones = *reporte de comisiones por ventas*
{{identificación-vendedor +
número-factura + monto-de-comisión}
+ total-comisión}

reporte-cuentas-por-cobrar = *reporte para el departamento de contabilidad donde se muestra el balance actual de cada cliente*
{identificación-cliente + saldo-actual}

reporte-inventario = *reporte producido para el departamento de contabilidad*
{{identificación-bodega + clave-libro + cantidad-inventario} +
inventario-total-libros}

reporte-regalías-autor = *reporte para los autores que muestra regalías por concepto de derechos de autor obtenidas o perdidas sobre ventas, créditos y devoluciones de cada libro durante un periodo de tres meses*
{total-copias + total-ventas-libro + total-regalías-libro}

reporte-trimestral-regalías = reporte para el departamento de contabilidad que muestra las utilidades o pérdidas por concepto de derechos de autor obtenidas por ventas, créditos y devoluciones de cada libro durante un periodo de tres meses*
{{identificación-cliente + nombre-cliente + número-factura +
ventas-artículo + regalías-artículo} + total-copias +
total-ventas-libro + total-regalías-libro}

reporte-ventas-diario = *reporte que se manda diariamente al departamento de contabilidad*
{número-factura + nombre-cliente +
nombre-compañía + total-del-pedido}
+ total-ventas-diario

reporte-ventas-mensual = *reporte del total de ventas, devoluciones y créditos otorgados en un solo mes, enviados al departamento de contabilidad*
total-ventas + total-devoluciones +
total-créditos

respuesta-adelanto = *respuesta al editor de adquisiciones cuando solicita un adelanto de regalías para un autor*
["No existe tal autor" | "No existe tal libro" |
"Adelanto aprobado" | "Adelanto negado"]

respuesta-agencia-crédito = *respuesta de alguna agencia de tarjetas de crédito a una solicitud de autorización de cargo*
["Sí" | "No"]

respuesta-autorización-adelanto = *respuesta de la administración a la solicitud de adelanto por concepto de regalías por derechos de autor*
["Sí" | "No"]

respuesta-cambio-cliente = *respuesta a un cliente cuando notifica al sistema de un cambio de domicilio, etc.*
["No existe tal cliente" | "Cambio aceptado"]

respuesta-crédito = *respuesta a un cliente que desea un recordatorio de crédito*
["No existe tal pedido para este cliente" | "Ya se otorgó crédito" | "No se hizo pago para tal número de factura" | "El crédito aparecerá en el siguiente estado de cuenta" | "La factura no se pagó en exceso" | "Se otorga crédito por el monto total del pedido" | "Crédito por envío insuficiente:" + monto-de-crédito |
"Crédito por libros dañados:" + monto-de-crédito]

respuesta-de-existencias = *respuesta al departamento de mercadeo cuando piden la fecha en la que se tendrá en existencia un envío de libros de la imprenta*
["No existe tal libro" | "No hay envío pendiente" | Fecha- disponibilidad]

respuesta-envío = *mensaje de error a la bodega en respuesta a su aviso de que enviaron el pedido de un cliente*
"No se encuentra el pedido"

respuesta-factura = *respuesta a la pregunta de un cliente referente a una factura*
["no existe pedido con ese número de factura" | fecha-de-pedido + {artículos-pedidos} + cargos-envío + impuestos-venta]

respuesta-factura-imprenta = *respuesta del sistema a la imprenta cuando recibe una factura de ésta*
["No quedan pedidos de este libro"]

respuesta-inventario-físico = *mensaje del sistema a una bodega en respuesta a un inventario físico que se haya hecho*
["No existe tal bodega" | "Clave de libro ilegal" + clave-libro]

respuesta-libro-agotado = *respuesta al departamento de mercadeo cuando éste indica que debe considerarse que un libro está fuera de catálogo*
["No existe tal libro" | "El libro está descatalogado"]

respuesta-límite-crédito = *respuesta a las instrucciones de la administración de cambiar el límite de crédito de un cliente*
["No existe tal cliente" | "límite de crédito ilegal" | "Nuevo límite de crédito aprobado"]

respuesta-información-libro = *información acerca del título, precio, etc., de un libro*
libro

respuesta-no-en-existencia = *mensaje a la bodega en respuesta a su indicación de que un título determinado no se tiene en existencia en dicha bodega*
["No existe tal libro" | Error: no se encuentra el artículo de inventario" | "Mensaje de no existencia aceptado"]

respuesta-pedido = *respuesta al cliente cuando hace un pedido*
["El precio de la mercancía excede a la cantidad pagada" | "Solicitud de crédito negada" | "El pedido excede a su límite de crédito" | "Insuficientes libros en existencia para satisfacer su pedido" | "No existe tal cliente" | "No existe tal libro" | "Porcentaje ilegal de impuesto sobre la venta" | "Cargos de envío ilegales" | "Pedido aceptado"]

respuesta-pedido-impresión = *respuesta del sistema sobre el pedido de reimpresión de la administración*
["No existe tal libro" | "Tiraje ilegal" | "Pedido de impresión aceptado"]

respuesta-pedido-impresión-modificado = *respuesta del sistema cuando se recibe un pedido modificado de la imprenta*
["No existe tal libro" | "El pedido de impresión modificado está correcto"]

respuesta-reembolso = *respuesta a un cliente que ha solicitado un reembolso*
["No existe tal cliente" | "No hay reembolso" + "saldo actual es de" + saldo-actual | "Reembolso aprobado"]

respuesta-solicitud-devolución
= *respuesta a un cliente que desea devolver un libro*
("No se encuentra este pedido" | "Los libros se enviaron hace más de un año" | "No se pueden devolver tantos libros" | "Se aprueba la devolución" + "Favor de identificar la devolución por medio de" + #-autorización-devolución)

respuesta-status-pedido = *respuesta a un cliente que pregunta sobre status de un pedido que ha hecho*
["No existe pedido para tal número de factura" | fecha-pedido + {artículos-pedidos} + fecha-envío]

saldo-actual = *cantidad de dinero que actualmente debe un cliente*
A LA FECHA DEL SALDO ACTUAL
unidades: dólares; escala: 1 -10,000

signo-puntuación = *una coma, un punto, un signo de admiración, etc.* **

solicitud-adelanto-regalías = *solicitud del editor de adquisiciones para que se otorgue un adelanto de utilidades a un autor, con relación a un libro*
identificación-autor + clave-libro + adelanto

solicitud-autorización-adelanto = *mensaje para la administración donde se solicita la aprobación de un adelanto de utilidades por concepto de derechos de autor para el proyecto de un libro*
identificación-autor + clave-libro + adelanto

solicitud-cheque-adelanto = *mensaje para el departamento de contabilidad en el que se solicita el pago del adelanto autorizado de utilidades por concepto de derechos de autor*
identificación-autor + clave-libro + adelanto

solicitud-cheque-reembolso = *mensaje al departamento de contabilidad solicitando que se expida un cheque de reembolso para un cliente*
"Favor de pagar" + identificación-cliente + monto-del-reembolso

solicitud-clave-libro = *mensaje a la administración solicitando que se asigne clave a un libro nuevo*
"Favor de asignar clave para el siguiente libro:"

solicitud-crédito = *solicitud del cliente para que se le otorgue crédito sobre un pedido*
identificación-cliente + número-factura + clave-libro + cantidad-libros-a-crédito + razón-crédito + monto-de-crédito-solicitado

solicitud-devolución = *información sobre uno o más libros que el cliente desea devolver a cambio de crédito*
número-factura + detalles-devolución

solicitud-etiquetas = *solicitud del departamento de mercadeo para producir etiquetas de envío"
"Favor de producir etiquetas de envío"

solicitud-precio-unitario = *solicitud del precio unitario de un libro nuevo al departamento de mercadeo*
"Favor de indicar el precio unitario del siguiente libro:"

solicitud-reembolso = *solicitud de un cliente para que le sea expedido un cheque de reembolso por el monto de su saldo actual de crédito*
identificación-cliente + "solicitud de reembolso"

solicitud-tasa-regalía = *mensaje al departamento de adquisiciones en donde se solicita conocer la tasa de regalías de un libro nuevo*
"Favor de indicar la tasa de regalías para el siguiente libro:"

solicitud-venta = *solicitud del departamento de mercadeo para producir un reporte de ventas para todos los pedidos, créditos y devoluciones que ocurrieron después de la fecha especificada*
fecha-venta

tasa-comisión = *porcentaje de comisión que se le paga a un vendedor por la venta de libros. Se expresa como fracción*
escala: 0-0.25

tasa-impuestos-venta = *porcentaje de los impuestos sobre la venta, expresados como una fracción decimal, p. ej., un impuesto del 7% sería 0.07*
escala: 0.00 - 1.00

tasa-regalías = *tasa de las regalías que se le pagan a un autor por un libro, p. ej., 10 significa el 10%*
escala: 5-25

tipo-pago = *forma en que el cliente pagará su pedido de libros*
["Efectivo" | "Cheque" | "Tarjeta de crédito" | "Mandar la cuenta"]

tipo-pedido = *indicación de si un pedido se hizo por correo, por teléfono, o personalmente*
["Teléfono" | "Correo" | "En persona"]

tiraje = *cantidad de copias de un libro a imprimir*
**

título = *prefijo que se coloca antes del nombre de una persona*
["Sr." | "Sra." | "Srta." | "Dr." | "Prof."]

título-libro = *título completo de un libro de YOURDON Press*
1{carácter alfanumérico}

total-comisión = *total de comisiones que se le pagan a un vendedor durante el período de un mes, basado en todos los libros que vendió. Se calcula en el proceso 3.6*
**

total-copias = *número total de copias vendidas de un mismo libro, tomando en cuenta devoluciones y créditos, durante un período de tres meses*
**

total-créditos = *cantidad total de créditos que se da a todos los clientes durante un mes*
unidades: dólares

total-del-pedido = *cantidad total de dinero facturado por un pedido*
unidades: dólares

total-devoluciones = *cantidad total de dinero por libros que los clientes hayan devuelto durante un mes*
unidades: dólares

total-en-existencia = *número de copias de un libro de YOURDON Press que se tiene en existencia en todas las bodegas*
escala: 1 - 10,000

total-regalías-libro = *total de regalías obtenidas (o perdidas) por la venta, devolución o crédito de un libro durante un período de tres meses*
unidades: dólares

total-ventas = *cantidad total de ingresos por concepto de todos los pedidos en un mes*
unidades: dólares

total-ventas-diario = *monto total de ventas nuevas registradas diariamente*
unidades: dólares

total-ventas-libro = *total de ingresos obtenidos por la venta de un solo libro durante un período de tres meses, tomando en cuenta créditos y devoluciones*
unidades: dólares

valor-devolución = *valor de un lote de libros que se haya devuelto*
unidades: dólares

vendedor = @identificación-vendedor + nombre- vendedor

VENDEDORES = {vendedor}

ventas-artículo = *ingreso bruto por la venta de una o más copias del mismo libro en un solo pedido*
unidades: dólares

F.4.4 El diagrama de entidad-relación

A	DINERO
B	consiste en
C	PAGO
D	se hizo para
E	ARTICULO-PEDIDO
F	consiste en
G	PEDIDO
H	se basa en
I	DEVOLUCION
J	se basa en
K	CREDITO
L	se hizo para
M	REEMBOLSO
N	consiste en
O	se almacena en
P	ARCHIVO
Q	DETALLE-DEVOLUCION
R	CLIENTE
S	AUTOR
T	escribe
U	BODEGA
V	INVENTARIO
W	consiste en
X	ARTICULO-INVENTARIO
Y	LIBRO
Z	imprime
a	IMPRENTA

F.4.5 Las Especificaciones del Proceso

PROCESO 1.1.1: EDITAR DETALLES DEL PEDIDO
COMIENZA
SI **identificación-cliente-posible** = "nuevo"
    identificación-cliente = siguiente identificación-cliente disponible
    límite-crédito = límite de crédito estándar
    saldo-actual = 0
    nivel-plan-agencia = 0
    cliente = **identificación-cliente** + **nombre-cliente** + (**nombre-compañía**) + **domicilio-cliente** + **saldo-actual** + **límite-crédito** + **nivel-plan-agencia**
    AÑADIR registro de **cliente** nuevo a **CLIENTES**
OTRO
  ENCONTRAR **cliente** en **CLIENTES** con **identificación-cliente** = **identificación-cliente** en **detalles-pedido**
    SI no se encuentra registro
        respuesta-pedido ="No existe tal cliente"
        DESPLEGAR **respuesta-pedido**
        SALIR
        nota: esto significa que no se seguirá procesando el pedido
    FIN_SI
MIENTRAS haya más **artículo-pedido** en **detalles-pedido**
    ENCONTRAR libro en LIBROS con **clave-libro** = **clave-libro** en **artículo-pedido**
    SI no se encuentra registro
        respuesta-pedido = "No existe tal libro"
        DESPLEGAR **respuesta-pedido**
        SALIR
        nota: esto significa que no se seguirá procesando el pedido
    FIN_SI
FIN_MIENTRAS
SI **tasa-impuestos-ventas** está fuera de rango
    respuesta-pedido = "tasa de impuestos inválida"
    DESPLEGAR **respuesta-pedido**
    SALIR
    nota: esto significa que no se seguirá procesando el pedido
FIN_SI
SI **cargos-envío** están fuera de rango
    respuesta-pedido "cargos de envío inválidos"
    DESPLEGAR **respuesta-pedido**
    SALIR
    nota: esto significa que no se seguirá procesando el pedido

FIN_SI
detalles-pedido-válido = **identificación-cliente** + {**artículo- pedido**} + **tasa-impuestos-ventas** + **cargos-envío** + **tipo-pago** + (**pago- pedido**) + (**número-tarjeta-crédito**) + **pedido-atrasado-OK** + tipo-pedido
DESPLEGAR (al proceso 1.1.2) **detalles-pedido-válido**
TERMINA

PROCESO 1.1.2: VERIFICAR LIBRO EN EXISTENCIA
COMIENZA
SI **pedido-atrasado-OK** ="SÍ"
    DESPLEGAR (a burbuja 1.1.3) **detalles-pedido-válido** + "YONKERS"
OTRO
    SI tipo-pedido ="en-persona"
        MIENTRAS haya más **artículo-pedido** en **detalles-pedido-válido**
            ENCONTRAR **artículo-inventario** en ARTICULOS-INVENTARIO con **clave-libro** = **clave-libro** en **detalles-pedido-válido** y **identificación-bodega** = el lugar donde se hizo el pedido en persona
            LEER registro **artículo-inventario**
            SI **cantidad-inventario** < **cantidad-pedida**
                respuesta-pedido = "No hay suficientes libros para satisfacer su pedido"
                SALIR
                nota: esto significa que no se seguirá procesando el pedido
            FIN_SI
        FIN_MIENTRAS
        DESPLEGAR (a burbuja 1.1.4) **detalles-pedido-válido** + **identificación-bodega** (de la bodega en el lugar donde se recibió el pedido)
    OTRO
        suficientes-libros = "SI"
        REPITE HASTA que ya no haya **bodegas** en **BODEGAS** o suficientes-libros ="SI"
            observe que esto significa que se examinará por lo menos una bodega
            suficientes-libros = "SI"
            MIENTRAS haya más **artículo-pedido** en **detalles-pedido-válido**
            ENCONTRAR **artículo-inventario** en ARTICULOS-INVENTARIO con **clave-libro** = **clave-libro** en **detalles-pedido-válido** y **identificación-bodega** = **identificación-bodega** del registro actual de bodega

            LEER registro de **artículo-inventario**
            SI **cantidad-inventario** < **cantidad-pedida**
                suficientes-libros = "NO"
            FIN_SI
            FIN_MIENTRAS
        FIN_REPITE
        SI **suficientes-libros** ="NO"
            respuesta-pedido = "No hay suficientes libros para satisfacer su pedido"
            DESPLEGAR **respuesta-pedido**
        OTRO
            DESPLEGAR (a burbuja 1.1.4) **detalles-pedido-válido** + **identificación-bodega** del registro de bodega actual
        FIN_SI
    FIN_SI
TERMINA

PROCESO 1.1.3: VERIFICAR AUTORIZACION DE CREDITO
COMIENZA
total-precio = 0
REPITE HASTA que ya no haya **artículo-pedido** en detalles-**pedido-válido**
SUMAR (**cantidad-pedida** * **precio-unitario** * **descuento**) a total-precio
FIN_REPITE
MULTIPLICAR total-precio por (1 + **tasa-impuestos-ventas**)
SUMAR **cargos-envío** a total-precio
crédito-OK = "SI"
CASO **tipo-pago** de
    CASO **tipo-pago** = "Efectivo" o tipo-pago = "CHEQUE"
        SI total-precio > **pago-pedido**
            respuesta-pedido = "Precio de compra excede a la cantidad pagada"
            DESPLEGAR **respuesta-pedido**
            crédito-OK= "No"
            nota: esto significa que no se seguirá procesando el pedido en este momento
        FIN_SI
    CASO **tipo-pago** = "Tarjeta-crédito"
        pregunta-crédito = "Solicitud de autorización" + total-precio
        DESPLEGAR (a agencia de tarjetas de crédito) pregunta-crédito
        ACEPTAR (de agencia de crédito) **respuesta-agencia-crédito**
        SI **respuesta-agencia-crédito** = "No"
            respuesta-pedido = "Solicitud de crédito negada"

```
                DESPLEGAR respuesta-pedido
                crédito-OK = "No"
                *nota: esto significa que no se seguirá
                procesando el pedido en este momento*
            FIN_SI
        CASO tipo-pago = "Enviar la cuenta"
            ENCONTRAR cliente en CLIENTES con identificación-
                    cliente = identificación-cliente en detalles-pedido-válido
            LEER registro de cliente
            SI saldo-actual + total-precio > límite-crédito
                respuesta-pedido = "el pedido excede su límite de crédito"
                DESPLEGAR respuesta-pedido
                crédito-OK = "No"
                *Nota: esto significa que no se seguirá
                procesando en este momento*
            FIN_SI
FIN_CASO
SI crédito-OK="sí"
    DESPLEGAR (a burbuja 1.1.4) detalles-pedido-válido +
            total-precio + identificación-bodega
    DESPLEGAR (a burbuja 1.1.5) detalles-pedido-válido +
            identificación-bodega
FIN_SI
TERMINA
```

PROCESO 1.1.4: INGRESAR PEDIDO

```
COMIENZA
MIENTRAS haya más artículo-pedido en detalles-pedido-válido
    CREAR registro de artículo-pedido a partir del siguiente artículo-pedido en
        detalles-pedido-válido
    AÑADIR registro de artículo-pedido a ARTICULOS-PEDIDOS
FIN_MIENTRAS
CREAR registro de pedido a partir de detalles-pedido-válido y identificación-
    bodega
AÑADIR registro de pedido a PEDIDOS
CREAR registro de factura a partir de detalles-pedido-válido
AÑADIR registro de factura a FACTURAS
SI tipo-pago = "Efectivo" o "Cheque" o "Tarjeta de crédito"
    CREAR registro de dinero a partir de detalles-pedido-válido
    AÑADIR registro de dinero a DINERO
    CREAR registro de pago a partir de detalles-pedido-válido
    AÑADIR registro de pago a PAGOS
```



```
FIN_SI
AÑADIR detalles-pedido-válido a ARCHIVOS
respuesta-pedido = "Pedido aceptado"
DESPLEGAR respuesta-pedido
TERMINA
```

PROCESO 1.1.5: VERIFICAR INVENTARIO PARA REIMPRIMIR

```
COMIENZA
MIENTRAS haya más artículo-pedido en detalles-pedido-válido
    ENCONTRAR artículo-inventario en ARTICULOS-INVENTARIO con
        clave-libro = clave-libro en artículo-pedido y
        identificación-bodega dada como entrada en este proceso
    LEER registro de artículo-inventario
    RESTAR cantidad-pedida de cantidad-inventario
    *nota: esto pudiera resultar en un inventario negativo;
    simplemente significa que la bodega no podrá surtir el
    pedido hasta que llegue una reimpresión"
    ESCRIBIR registro de artículo-inventario
    ENCONTRAR libro en LIBROS con clave-libro = clave-libro en
        artículo-pedido
    LEER registro de libro
    RESTAR cantidad-pedida de total-en-existencia
    ESCRIBIR registro de libro
    SI total-en-existencia < umbral-repetir-pedido
        aviso-inventario-bajo = clave-libro + TOTAL-EN-
        EXISTENCIA + "tiempo de reimprimir"
        DESPLEGAR aviso-inventario-bajo
    FIN_SI
FIN_MIENTRAS
TERMINA
```

PROCESO 1.2: PROCESAR PEDIDO DE VENDEDOR

Por ahora, la política para procesar un pedido de un vendedor es la misma que para procesar un pedido normal de un cliente. Vea la Figura 1.1 para más detalles

PROCESO 1.3: INSCRIBIR CLIENTE EN PLAN DE AGENCIA

Precondición-1.
Existe un **cliente** en **CLIENTES** que al que corresponde
    identificación-cliente con **nivel-plan-agencia** = 0
    y con **elección-plan-agencia** > 0
    y **elección-plan-agencia** menor o igual que máximo nivel de agencia.

postcondición-1.
nivel-plan-agencia en **cliente** se hace igual a **elección-plan-agencia**

```
PROCESO 1.4:        ENVIAR FACTURAS
COMIENZA
MIENTRAS haya más factura en FACTURAS
      LEER siguiente factura
DESPLEGAR factura
FIN_MIENTRAS
TERMINA

PROCESO 2.1:        PROCESAR PAGOS
COMIENZA
SI identificación-cliente está presente
      ENCONTRAR registro en CLIENTES con identificación-
                  cliente correspondiente
      SI no se puede encontrar registro
            ESCRIBIR detalles-pago a PAGOS con identificación-cliente =
                  "Efectivo no aplicado"
      OTRO
            RESTAR monto-total a saldo-actual
            ESCRIBIR registro a CLIENTES
            ESCRIBIR detalles-pago a PAGOS
OTRO
      ESCRIBIR detalles-pago a PAGOS con identificación-
                  cliente = "Efectivo no aplicado"
      ESCRIBIR fecha actual + identificación-cliente +
                  detalles-pago a DINERO
FIN_SI
TERMINA

PROCESO 2.2:  PROPORCIONAR INFORMACION DE LIBROS
COMIENZA
ENCONTRAR registro de libro en LIBROS con título-libro
                  correspondiente
respuesta-información-libro = contenido de todo el registro libro
DESPLEGAR respuesta-información-libro
TERMINA

PROCESO 2.3:  PROCESAR SOLICITUD DE DEVOLUCION
COMIENZA
ENCONTRAR pedido en PEDIDOS que corresponda con número-
                  factura en solicitud-devolución
SI no se encuentra el registro
      respuesta-solicitud-devolución = "No se encuentra este pedido"
      DESPLEGAR respuesta-solicitud-devolución
OTRO
      LEER registro de pedido
```



```
      SI fecha de envío es hace más de un año
            respuesta-devolución-libro = "los libros se
                  enviaron hace más de un año "
            DESPLEGAR respuesta-solicitud-devolución
      OTRO
            todo-OK = "sí"
            REPITE HASTA que no haya más artículo-devuelto en
            detalles-devolución
                  ENCONTRAR artículo-pedido en ARTICULOS-PEDIDOS
                        que corresponda con número-factura
                        en solicitud-devolución y clave-
                        libro en artículo-devuelto
                  SI no se encuentra registro
                        DESPLEGAR "Este libro no fue parte del pedido"
                        todo-OK = "no"
      OTRO
            LEER registro de artículo-pedido
            SI cantidad-a-devolver en artículo-
            devuelto es más que la mitad de la
            cantidad-pedida en artículo-pedido
            respuesta-solicitud-devolución = "No se pueden devolver
                        tantos libros"
                        DESPLEGAR respuesta-solicitud-devolución
                        todo-OK = "no"
                  FIN_SI
            FIN_SI
      FIN_REPITE
      SI todo-OK = "sí"
            LEER #-autorización-devolución de #-
                  AUTORIZACION-DEVOLUCION
            respuesta-solicitud-devolución = "Devolución
            correcta" + "Favor de identificar devolución
            actual mediante" + #-autorización-devolución
            DESPLEGAR respuesta-solicitud-devolución
            ESCRIBIR detalles-devolución, #-autorización-
                        devolución a DEVOLUCIONES-AUTORIZADAS
            SUMAR 1 a #-autorización-devolución
            ESCRIBIR #-autorización-devolución a #-
                        AUTORIZACION-DEVOLUCION
            FIN_SI
      FIN_SI
FIN_SI
TERMINA
```

PROCESO 2.4: RESPONDER A PREGUNTA SOBRE STATUS DE PEDIDO

```
COMIENZA
ENCONTRAR pedido en PEDIDOS con número-factura correspondiente
SI no se encuentra registro
    respuesta-status-pedido = "No existe pedido con tal número de factura"
    DESPLEGAR respuesta-status-pedido
OTRO
    LEER registro de pedido en PEDIDOS con número-factura correspondiente
    respuesta-status-pedido = fecha-pedido + {artículos-
                              pedidos} + fecha-envío
    DESPLEGAR respuesta-status-pedido
FIN_SI
TERMINA
```

PROCESO 2.5: RESPONDER A PREGUNTA SOBRE FACTURA

```
COMIENZA
ENCONTRAR pedido en PEDIDOS con número-factura correspondiente
SI no se encuentra registro
    respuesta factura = "no existe pedido con tal número de factura"
    DESPLEGAR respuesta-factura
OTRO
    LEER registro de pedido
    respuesta-factura = fecha-pedido + {artículo-pedido} +
                        cargos-envío + impuestos-venta
    DESPLEGAR respuesta-factura
FIN_SI
TERMINA
```

PROCESO 2.6: PRODUCIR RECORDATORIO DE CREDITO

```
COMIENZA
ENCONTRAR pedido en PEDIDOS con identificación-cliente
        correspondiente y número-factura que corresponda
        con número-factura en solicitud-crédito
    SI no se encuentra registro
        respuesta-crédito = "No existe tal pedido para este cliente"
        DESPLEGAR respuesta-crédito
    OTRO
        ENCONTRAR crédito en CREDITOS con número-factura
                que corresponda con número-factura en
                solicitud-crédito
        SI se encuentra registro
            respuesta-crédito = "El crédito ya se otorgó"
            DESPLEGAR respuesta-crédito
        OTRO
            CASO razón-crédito de
```



```
            CASO razón-crédito = "EXCESO DE PAGO"
                ENCONTRAR pago en PAGOS con número-
                    factura que corresponda con
                    número-factura en solicitud-crédito
                SI no se encuentra registro
                    respuesta-crédito = "No se hizo ningún pago
                    para tal número de factura"
                    DESPLEGAR respuesta-crédito
                OTRO
                    LEER pago
                    ENCONTRAR pedido en PEDIDOS con
                        número-factura = número-
                        factura en solicitud-crédito
                    SI monto-total > total-del-pedido respuesta-crédito
                        = "El crédito se mostrará en el siguiente saldo"
                    OTRO
                        respuesta-crédito = "No hubo
                        exceso en el pago de la factura"
                    FIN_SI
                    DESPLEGAR respuesta-crédito
                FIN_SI
            CASO razón-crédito = "Retraso excesivo"
                crédito = numero-factura +
                identificación-cliente + fecha-actual +
                total-del-pedido
                AÑADIR crédito a CREDITOS
                respuesta-crédito = "Crédito por cantidad total del pedido"
                DESPLEGAR respuesta-crédito
            CASO razón-crédito = "Envío incompleto"
                SI total-pedidos > monto-de-crédito-solicitado
                    crédito = número-factura +
                    identificación-cliente + fecha-actual +
                    monto-de-crédito-solicitado
                    respuesta-crédito = "Crédito por
                    envío incompleto: " + monto-de-crédito-solicitado
                    DESPLEGAR respuesta-crédito
                OTRO
                    crédito= número-factura + identificación-cliente +
                    fecha actual + total-del-pedido
                    respuesta-crédito = "Crédito por
                    envío incompleto: " + total-del-pedido
                    DESPLEGAR respuesta-crédito
                FIN_SI
                AÑADIR crédito a CREDITOS
```

```
                CASO razón-crédito = "libros dañados"
                    SI total-pedidos > monto-de-crédito-solicitado
                        crédito = número-factura +
                        identificación-cliente + fecha actual +
                        monto-de-crédito-solicitado
                        respuesta-crédito = "Crédito por
                        libros dañados: " + monto-de-crédito-solicitado
                        DESPLEGAR respuesta-crédito
                    OTRO
                        crédito = número-factura +
                        + identificación-cliente + fecha-
                        actual + total-del-pedido
                        DESPLEGAR respuesta-crédito
                    FIN_SI
                    AÑADIR crédito a CREDITOS
                FIN_CASO
            FIN_SI
        FIN_SI
TERMINA
```

PROCESO 2.7: AVISAR A CONTABILIDAD DE LA NECESIDAD DE REEMBOLSO

```
COMIENZA
ENCONTRAR cliente en CLIENTES con identificación-cliente que corresponda
                con identificación-cliente en solicitud-reembolso
SI no se encuentra registro
    solicitud-reembolso = "no existe tal cliente"
    DESPLEGAR solicitud-reembolso
OTRO
    LEER registro de cliente
    SI saldo-actual es mayor o igual a cero solicitud-reembolso
        = "No hay reembolso" + "Saldo actual es" + saldo-actual
        DESPLEGAR solicitud-reembolso
    OTRO
        solicitud-reembolso = "reembolso aprobado"
        solicitud-cheque-reembolso = "Favor de pagar" +
                identificación-cliente + saldo-actual
        DESPLEGAR solicitud-reembolso
        DESPLEGAR solicitud-cheque-reembolso
        ESCRIBIR cero en saldo-actual en registro de cliente
        ESCRIBIR fecha actual + identificación-cliente +
                saldo-actual a REEMBOLSOS
    FIN_SI
FIN_SI
TERMINA
```


PROCESO 2.8: FIJAR NUEVO LIMITE DE CREDITO

```
COMIENZA
ENCONTRAR cliente en CLIENTES que corresponda con
                identificación-cliente
SI no se encuentra registro
    respuesta-límite-crédito = "No existe tal cliente"
OTRO
    LEER registro de cliente
    SI nuevo-límite-crédito < 0
        respuesta-límite-crédito = "límite de crédito inválido"
        DESPLEGAR respuesta-límite-crédito
    OTRO
        respuesta-límite-crédito = "Nuevo límite de crédito correcto"
        DESPLEGAR respuesta-límite-crédito
        REEMPLAZAR límite-crédito por nuevo-límite-crédito
        ESCRIBIR registro de cliente
    FIN_SI
FIN_SI
TERMINA
```

PROCESO 2.9: MODIFICAR DETALLES DE CLIENTES

```
COMIENZA
ENCONTRAR cliente en CLIENTES que corresponda con
                identificación-cliente
SI no se encuentra registro
    respuesta-modificación-cliente = "No existe tal cliente"
    DESPLEGAR respuesta-modificación-cliente
OTRO
    LEER registro de cliente
    REEMPLAZAR nombre-cliente, nombre-compañía, domicilio-
                cliente con nombre-cliente, nombre-compañía,
                domicilio-cliente en detalles-cliente
    respuesta-modificación-cliente = "modificación aceptada"
    DESPLEGAR respuesta-modificación-cliente
FIN_SI
TERMINA
```

PROCESO 3.1: PRODUCIR RECIBOS DE EFECTIVO

```
COMIENZA
efectivo-recolectado = 0
MIENTRAS haya más registros en DINERO
    LEER siguiente registro de dinero
    DESPLEGAR dinero
    efectivo-recolectado = efectivo-recolectado + cantidad-dinero
```

FIN_MIENTRAS
informe-efectivo = efectivo-recolectado
DESPLEGAR **informe-efectivo**
TERMINA

PROCESO 3.2: PRODUCIR INFORME DE VENTAS DIARIO
COMIENZA
total-diario = 0
MIENTRAS haya más **pedido** en **PEDIDOS** con **fecha-pedido** = fecha actual
    LEER siguiente **pedido** con **fecha-pedido** = fecha actual
    SUMAR **número-factura, nombre-cliente, nombre-compañía, pedido-total** como nuevo renglón en **informe-ventas-diario**
    SUMAR **total-pedidos** a total-diario
FIN_MIENTRAS
SUMAR total-diario como nuevo renglón en **informe-ventas-diario**
DESPLEGAR **informe-ventas-diario**

PROCESO 3.3: PRODUCIR INFORME DE VENTAS MENSUAL
total-ventas = 0
total-devoluciones = 0
total-créditos = 0
MIENTRAS haya más **pedido** en **PEDIDOS** con **fecha-pedido** de este mes
    SUMAR **total-pedidos** a **total-ventas**
FIN_MIENTRAS
MIENTRAS haya más **devolución** en **DEVOLUCIONES** con **fecha-devolución** de este mes
    SUMAR **valor-devolución** a **total-devoluciones**
FIN_MIENTRAS
MIENTRAS haya más **crédito** en **CREDITOS** con **fecha-crédito** de este mes
    SUMAR **monto-de-crédito** a **total-créditos**
FIN_MIENTRAS
informe-ventas-mensual = **total-ventas, total-devoluciones, total-créditos**
DESPLEGAR **informe-ventas-mensual**
TERMINA

PROCESO 3.4: PRODUCIR INFORME DE REGALIAS TRIMESTRAL
COMIENZA
MIENTRAS haya más **libro** en **LIBROS**
total-libros = 0
total-ventas = 0
total-regalías = 0
LEER siguiente registro de **libro**
MIENTRAS haya más **pedido** en **PEDIDOS** con **fecha-pedido** de este trimestre

    LEER siguiente registro de **pedido**
    MIENTRAS haya más **artículo-pedido** en **ARTICULOS-PEDIDOS** con **número-factura** que corresponda con **número-factura** en registro de **pedido** actual y **clave-libro** que corresponda con **clave-libro** en registro de **libro** actual
        LEER siguiente **artículo-pedido**
        SUMAR **cantidad-pedida** a total-libros
        esta-venta = **cantidad-pedida** * **precio-unitario** * **descuento**
        SUMAR esta-venta a total-ventas
        SUMAR (esta-venta * **total-regalías**) a total-regalías
        AÑADIR **identificación-cliente, nombre-cliente, número-factura**, esta-venta, (esta-venta * **total-regalías**) al siguiente renglón de **informe-regalías-trimestral**
    FIN_MIENTRAS
    FIN_MIENTRAS
    MIENTRAS haya más **crédito** en **CREDITOS** con clave-libro que corresponda con **clave-libro** en registro de **libro** actual y **fecha-crédito** de este trimestre
        LEER siguiente **crédito**
        RESTAR **cantidad-libros-a-crédito** de total-libros
        RESTAR **monto-de-crédito** de total-ventas
        RESTAR (**monto-de-crédito** * **tasa-regalías**) de total-regalías
        AÑADIR **identificación-cliente, nombre-cliente, número-factura, monto-de-crédito,** (**monto-de-crédito** * **tasa-regalías**) al siguiente renglón de **informe-regalías-trimestral**
    FIN_MIENTRAS
    MIENTRAS haya más **devolución** en **DEVOLUCIONES** con **clave-libro** que corresponda con **clave-libro** en registro de **libro** actual y **fecha-devolución** de este trimestre
        LEER siguiente **devolución**
        RESTAR **cantidad-devuelta** de total-libros
        RESTAR **valor-devolución** de total-ventas
        RESTAR (**valor-devolución** * **tasa-regalías**) de total- regalías
        AÑADIR **identificación-cliente, nombre-cliente, número-factura, valor-devolución,** (**valor-devolución** * **tasa-regalías**) al siguiente renglón de **informe-trimestral-regalías**
    FIN_MIENTRAS
    AÑADIR total-libros, total-ventas, total-regalías al siguiente renglón de **informe-regalías-trimestral**
FIN_MIENTRAS
DESPLEGAR **informe-regalías-trimestral**
TERMINA

PROCESO 3.5: PRODUCIR INFORME DE INVENTARIO
COMIENZA
REPITE HASTA que ya no haya **libro** en **LIBROS**
    LEER siguiente **libro** en **LIBROS**
    total-inventario = 0
    REPITE hasta que no haya más **artículo-inventario** en **ARTICULOS-INVENTARIO** con **clave-libro** que corresponda con **clave-libro** en **libro**
        SUMAR **cantidad-inventario** a total-inventario
        SUMAR **identificación-bodega, clave-libro, cantidad- inventario** al siguiente renglón de **reporte-inventario**
    FIN_REPITE
        SUMAR total-inventario al siguiente renglón de **reporte-inventario**
FIN_REPITE
TERMINA

PROCESO 3.6: PRODUCIR INFORME DE COMISION DE VENTAS
COMIENZA
MIENTRAS haya más **vendedor** en **VENDEDORES**
    LEER siguiente registro de **vendedor**
    comisión-vendedor = 0
    MIENTRAS haya más **pedido** en **PEDIDOS** con **identificación-vendedor** que corresponda con **identificación-vendedor** en vendedor y con **fecha-pedido** de este mes
        LEER siguiente registro de **pedido**
        comisión = **tasa-comisión** * **total-del-pedido**
        SUMAR comisión a comisión-vendedor
        AÑADIR **identificación-vendedor, número-factura,** comisión al siguiente renglón de **reporte-comisión**
    FIN_MIENTRAS
    AÑADIR comisión-vendedor al siguiente renglón de **reporte-comisión**
FIN_MIENTRAS
TERMINA

PROCESO 3.7: PRODUCIR DECLARACIONES
COMIENZA
REPITE HASTA que ya no haya **cliente** en **CLIENTES**
    LEER registro de **cliente**
    saldo-nuevo = **saldo-actual**
    MIENTRAS haya más **pedido** en **PEDIDOS** con **identificación-cliente** = **identificación-cliente** en registro de **cliente** actual y **fecha-pedido** posterior a **fecha-saldo-actual**
        LEER siguiente registro de **pedido**
        SUMAR **total-pedidos** a saldo-nuevo
        AÑADIR **pedido** al siguiente renglón de **declaración**

    FIN_MIENTRAS
    MIENTRAS haya más **pago** en **PAGOS** con **identificación-cliente** = **identificación-cliente** en registro de **cliente** actual y fecha-pago posterior a **fecha-saldo-actual**
        LEER siguiente registro de **pago**
        RESTAR **total-del-pedido** de saldo-nuevo
        AÑADIR **pago** al siguiente renglón de **declaración**
    FIN_MIENTRAS
    MIENTRAS haya más **reeembolso** en **REEMBOLSOS** con **identificación-cliente** = **identificación-cliente** en registro actual de cliente y **fecha-reembolso** posterior a **fecha-saldo-actual**
        LEER siguiente registro de **reembolso**
        SUMAR **monto-del-reembolso** a saldo-nuevo
        AÑADIR **reembolso** al siguiente renglón de **declaración**
    FIN_MIENTRAS
    MIENTRAS haya más **crédito** en **CREDITOS** con **identificación-cliente** = **identificación-cliente** en registro de **cliente** actual y **fecha-crédito** posterior a **fecha-saldo-actual**
        LEER siguiente registro de **crédito**
        RESTAR **monto-de-crédito** de saldo-nuevo
        AÑADIR **crédito** al siguiente renglón de **declaración**
    FIN_MIENTRAS
    MIENTRAS haya más **devolución** en **DEVOLUCIONES** con **identificación-cliente** = **identificación-cliente** en registro de cliente actual y **fecha-devolución** posterior a **fecha-saldo-actual**
        LEER siguiente registro de **devolución**
        RESTAR **valor-devolución** de saldo-nuevo
        AÑADIR **pago** al siguiente renglón de **declaración**
    FIN_MIENTRAS
AÑADIR saldo-nuevo al siguiente renglón de **declaración**
DESPLEGAR **declaración**
HACER **fecha-saldo-actual** en registro de **pedido** actual igual a fecha de hoy
HACER **saldo-actual** en registro de **pedido** actual igual a saldo-nuevo
FIN_REPITE
TERMINA

PROCESO 3.8: PRODUCIR REPORTE DE CUENTAS POR COBRAR
COMIENZA
REPITE HASTA que ya no haya **cliente** en **CLIENTES**
    LEER siguiente registro de **cliente**
    saldo-nuevo = **saldo-actual**
    MIENTRAS haya más **pedido** en **PEDIDOS** con **identificación-cliente** = **identificación-cliente** en registro de **cliente** actual y **fecha-pedido** posterior a **fecha-saldo-actual**

```
        LEER siguiente registro de pedido
        SUMAR total-del-pedido a saldo-nuevo
    FIN_MIENTRAS
    MIENTRAS haya más pago en PAGOS con identificación-
    cliente = identificación-cliente en registro de cliente actual
    y fecha-reembolso posterior a fecha-saldo-actual
        LEER siguiente registro de reembolso
        RESTAR monto-del-reembolso a saldo-nuevo
    FIN_MIENTRAS
    MIENTRAS haya más crédito en CREDITOS con
    identificación-cliente = identificación-cliente en registro actual de cliente y
    fecha-crédito posterior a fecha-saldo-actual
        LEER siguiente registro de crédito
    RESTAR monto-de-crédito de saldo-nuevo
    AÑADIR crédito al siguiente renglón de declaración
    FIN_MIENTRAS
    MIENTRAS haya más devolución en DEVOLUCIONES con
    identificación-cliente= identificación-cliente en registro de cliente actual y
    fecha-devolución posterior a fecha-saldo-actual
        LEER siguiente registro de devolución
        RESTAR valor-devolución de saldo-nuevo
    FIN_MIENTRAS
FIN_REPITE
AÑADIR identificación-cliente, saldo-nuevo al siguiente
        renglón de informe-cuentas-por-pagar
DESPLEGAR reporte-cuentas-por-cobrar
TERMINA

PROCESO 4.1: ACEPTAR COTIZACION DE LA IMPRENTA
COMIENZA
ACEPTAR (de imprenta) identificación-imprenta, cotización-imprenta
DESPLEGAR (a administración) identificación-imprenta, cotización-imprenta
TERMINA

PROCESO 4.2: HACER PEDIDO DE IMPRESION
COMIENZA
ENCONTRAR libro en LIBROS con clave-libro que corresponda con
clave-libro en instrucciones-pedido-impresión
SI no se encuentra registro
    respuesta-pedido-impresión = "No existe tal libro"
    DESPLEGAR respuesta-pedido-impresión
OTRO
    SI tiraje < 0
        respuesta-pedido-impresión = "Cantidad de impresión inválida"
        DESPLEGAR respuesta-pedido-impresión
```



```
    OTRO
        respuesta-pedido-impresión = "Pedido de impresión aceptado"
    DESPLEGAR respuesta-pedido-impresión
    HACER cantidad-sobre-pedido en libro igual a tiraje
    HACER fecha-disponibilidad en libro igual a fecha-
            disponibilidad en instrucciones-pedido-impresión
    ESCRIBIR registro de libro
    pedido-impresión = clave-libro + tiraje
    DESPLEGAR pedido-impresión, identificación-imprenta
    FIN_SI
FIN_SI
TERMINA

PROCESO 4.3: REVISAR PEDIDO DE LIBROS
COMIENZA
ENCONTRAR libro en LIBROS con clave-libro que corresponda con clave-libro
            en pedido-impresión-modificado
SI no se encuentra registro
    respuesta-pedido-impresión-modificado = "No existe tal libro"
    DESPLEGAR respuesta-pedido-impresión-modificado
OTRO
    LEER registro de libro
    HACER cantidad-sobre-pedido igual a cantidad-modificada
    HACER fecha-disponibilidad igual a fecha-modificada
    ESCRIBIR registro de libro en LIBROS
    respuesta-pedido-impresión-modificado = "Pedido de libros
    revisado correcto"
    DESPLEGAR respuesta-pedido-impresión-modificado
FIN_SI
TERMINA

PROCESO 4.4: PROCESAR FACTURA IMPRENTA
COMIENZA
ENCONTRAR libro en LIBROS con clave-libro que corresponda con
            clave-libro en factura-imprenta
SI no se encuentra registro
    respuesta-factura-imprenta = "No existen pedidos pendientes para este libro"
    DESPLEGAR respuesta-factura-imprenta
OTRO
    DESPLEGAR factura-imprenta (a administración para su aprobación)
    ACEPTAR autorización-factura-imprenta
    SI autorización-factura-imprenta = "NO"
        respuesta-factura-imprenta = "Factura rechazada;
        Favor de comunicarse con la administración para discutirlo"
        DESPLEGAR respuesta-factura-imprenta
```

```
        OTRO
            respuesta-factura-imprenta = "Factura aceptada"
            DESPLEGAR respuesta-factura-imprenta
            factura-imprenta-aprobada = factura-imprenta
            DESPLEGAR factura-imprenta-aprobada
        FIN_SI
FIN_SI
TERMINA
```

PROCESO 4.5: PEDIR COTIZACION IMPRENTA

```
COMIENZA
MIENTRAS haya más imprenta en IMPRENTAS
        LEER siguiente registro de imprenta
        SI imprenta corresponde con alguna de las identificación-imprenta en la
        entrada a este proceso solicitud-cotización = clave-libro + {cantidad}
            DESPLEGAR solicitud-cotización
        FIN_SI
FIN_MIENTRAS
TERMINA
```

PROCESO 5: CREAR NUEVO REGISTRO DE LIBROS

```
COMIENZA
DESPLEGAR (para administración) título-libro + solicitud-clave-libro
ACEPTAR (de administración) clave-libro
DESPLEGAR (para adquisiciones) título-libro + solicitud-tasa- regalías
ACEPTAR (de adquisiciones) tasa-regalías
DESPLEGAR (para mercadeo) título-libro + solicitud-precio-unitario
ACEPTAR (de mercadeo) precio-unitario
libro = clave-libro + título-libro + identificación-autor + total-regalías
HACER total-en-existencia igual a cero
HACER fecha-disponibilidad igual a fecha-en-existencia
AÑADIR libro a LIBROS
TERMINA
```

PROCESO 6.1: PRODUCIR ETIQUETAS DE ENVIO

```
COMIENZA
ORDENAR CLIENTES por código-postal en etiquetas-envío
DESPLEGAR etiquetas-envío
TERMINA
```

PROCESO 6.2: PRODUCIR ESTADISTICAS DE VENTAS

```
COMIENZA
REPITE hasta que no haya más libro en LIBROS
```



```
        ingresos-venta = 0
        devoluciones-ventas = 0
        créditos-venta = 0
    MIENTRAS haya más pedido en PEDIDOS con fecha-pedido
    posterior a fecha-venta
        LEER siguiente registro de pedido
        MIENTRAS haya más artículo-pedido en el registro de
        pedidos actual con clave-libro = clave-libro en registro actual de libro
            LEER siguiente registro de artículo-pedido
            SUMAR (cantidad-pedida * precio-unitario *
                descuento) a ingresos-venta
            FIN_MIENTRAS
        FIN_MIENTRAS
        MIENTRAS haya más devolución en DEVOLUCIONES con fecha-
        devolución posterior a fecha-venta y clave-libro = clave-libro en registro de
        libro actual
            SUMAR valor-devolución a devoluciones-ventas
        FIN_MIENTRAS
        MIENTRAS haya más crédito en CREDITOS con fecha-crédito
        posterior a fecha-venta y clave-libro = clave-libro en
        registro de libro actual
            SUMAR monto-de-crédito a créditos-ventas
        FIN_MIENTRAS
        AÑADIR clave-libro, ingresos-venta, devoluciones-ventas, créditos-venta,
        al siguiente renglón de estadísticas-venta
    FIN_REPITE
    DESPLEGAR estadísticas-ventas
    TERMINA
```

PROCESO 6.3: PRODUCIR FECHA DE EXISTENCIAS

```
COMIENZA
ENCONTRAR libro en LIBROS que corresponda con clave-libro
SI no se encuentra registro
        respuesta-de-existencias = "No existe tal libro "
        DESPLEGAR respuesta-de-existencias
OTRO
        LEER registro de libro
        SI fecha-disponibilidad = "nulo"
            respuesta-de-existencias = "No existen envíos pendientes"
        OTRO respuesta-de-existencias = fecha-disponibilidad
            DESPLEGAR respuesta-de-existencias
        FIN_SI
FIN_SI
TERMINA
```

PROCESO 6.4: ELIMINAR LIBRO
COMIENZA
ENCONTRAR **libro** en **LIBROS** que corresponda con **clave-libro**
SI no se encuentra registro
    respuesta-libro-agotado = "No existe tal libro"
    DESPLEGAR **respuesta-libro-agotado**
OTRO
    LEER registro de **libro**
    HACER **indicador-libro-agotado** igual a SI
    ESCRIBIR registro de **libro**
    respuesta-libro-agotado = "El libro se ha declarado agotado"
    DESPLEGAR **respuesta-libro-agotado**
FIN_SI
TERMINA

PROCESO 7.1: PRODUCIR DOCUMENTOS DE ENVIO
COMIENZA
REPITE HASTA que ya no haya **bodega** en **BODEGAS**
    LEER siguiente registro de **bodega**
    nota: esta parte produce la lista de selección para la bodega
    REPITE HASTA que ya no haya **libro** en **LIBROS**
        libros a seleccionar = 0
        MIENTRAS haya más **pedido** en **PEDIDOS** con **fecha-pedido** = fecha de hoy y **identificación-bodega** que corresponda con **identificación-bodega** en registro actual de **bodega**
            LEER siguiente registro de **pedido**
        MIENTRAS haya más **artículo-pedido** con **número-factura = número-factura** en registro de **pedido**
            LEER siguiente registro de **artículo-pedido**
            SUMAR **cantidad-pedida** a libros por seleccionar
        FIN_MIENTRAS
    FIN_MIENTRAS
    AÑADIR **título-libro,** libros-por-seleccionar al siguiente renglón de **documentos-envío**
FIN_REPITE
nota: esta parte produce las etiquetas de envío
MIENTRAS haya más **pedido** en **PEDIDOS** con **fecha-pedido** = fecha de hoy y **identificación-bodega = identificación-bodega** en registro de **bodega** actual
    LEER siguiente registro de **pedido**
    AÑADIR **nombre-cliente, domicilio-cliente, número-factura** a siguiente renglón de **documentos-envío**
FIN_MIENTRAS
nota: esta parte produce una copia del pedido original para la bodega

MIENTRAS haya más **pedido** en **PEDIDOS** con **fecha-pedido** = fecha de hoy y **identificación-bodega = identificación-bodega** en registro de **bodega** actual
        LEER siguiente registro de **pedido**
        AÑADIR **identificación-cliente, fecha-pedido, cargos-envío, impuesto-venta** al siguiente renglón de **documentos-envío**
        REPITE HASTA que ya no haya **artículo-pedido** en **ARTICULOS-PEDIDOS** con **número-factura** que corresponda con **número-factura** en registro actual de **pedido**
            AÑADIR **artículo-pedido** al siguiente renglón de **documentos-envío**
        FIN_REPITE
    FIN_MIENTRAS
FIN_REPITE
TERMINA

PROCESO 7.2: REGISTRAR ENVIOS DE IMPRENTA
COMIENZA
ENCONTRAR **libro** en **LIBROS** que corresponda con **título-libro**
SI no se encuentra registro
    notificación-envío = "No existe tal libro"
    DESPLEGAR **notificación-envío**
OTRO
    notificación-envío ="Recibido de la imprenta" + clave-libro + cantidad-recibida
    DESPLEGAR **notificación-envío**
    LEER registro de **libro**
    SUMAR **cantidad-recibida** a **total-en-existencia**
    HACER **cantidad-sobre-pedido** igual a cero
    ESCRIBIR registro de **libro** en **LIBROS**
    LEER **artículo-inventario** en **ARTICULOS-INVENTARIO** con **identificación-bodega** = "YONKERS" y que corresponda con **clave-libro**
    SUMAR **cantidad-recibida** a **cantidad-inventario**
    ESCRIBIR registro de **artículo-inventario**
FIN_SI
TERMINA

PROCESO 7.3: REGISTRAR DEVOLUCIONES DE LIBROS POR CLIENTES
COMIENZA
ENCONTRAR **cliente** en **CLIENTES** con **identificación-cliente** que corresponda con **identificación-cliente** en **información-devolución-libro** o con **nombre-cliente** que corresponda con **nombre-cliente** en **información-devolución-libro**

```
SI no se encuentra registro
        instrucciones-devolución = "No se puede identificar
                cliente; aceptar libros de cualquier forma"
        DESPLEGAR instrucciones-devolución
OTRO
        ENCONTRAR devolución-autorizada en DEVOLUCIONES-
                AUTORIZADAS con #-autorización-devoluciones que
                corresponda con #-autorización-devoluciones en
                información-devolución-libro
        SI no se encuentra registro
                instrucciones-devolución = "Esta devolución no fue
                        autorizada; favor de regresarla"
                DESPLEGAR instrucciones-devolución
                aprobación-devolución-libro = "Esta devolución no
                        fue autorizada"
                DESPLEGAR (para cliente) aprobación-devolución-libro
        FIN_SI
FIN_SI
REPITE HASTA que ya no haya clave-libro en información-devolución-libro
        ENCONTRAR artículo-inventario en ARTICULOS-INVENTARIO
                que corresponda con identificación-bodega y clave-libro que
                corresponda con clave-libro en información-devolución-libro
        SI no se encuentra registro
                instrucciones-devolución = "No existe tal libro"
                DESPLEGAR instrucciones-devolución
        OTRO
                LEER registro de artículo-inventario
                SUMAR cantidad-devuelta a cantidad-inventario
                ESCRIBIR registro de artículo-inventario
                ENCONTRAR libro en LIBROS que corresponda con
                        identificación-libro
                LEER registro de libro
                SUMAR cantidad-devuelta a total-en-existencia
                ESCRIBIR registro de libro
        FIN_SI
FIN_REPITE
AÑADIR información-devolución-libro a DEVOLUCIONES
FIN
```

PROCESO 7.4: REGISTRAR INVENTARIO FISICO

```
COMIENZA
ENCONTRAR bodega en BODEGAS que corresponda con
                identificación-bodega
```



```
SI no se encuentra registro
        respuesta-inventario-físico = "No existe tal bodega"
        DESPLEGAR respuesta-inventario-físico
OTRO
        REPITE HASTA que ya no haya detalles-pedido en cuenta-inventario
                ENCONTRAR artículo-inventario en ARTICULOS-
                        INVENTARIO que corresponda con
                        identificación-bodega y clave-libro
                SI no se encuentra registro
                        respuesta-inventario-físico = "clave de libro inválida"
                        DESPLEGAR respuesta-inventario-físico
                OTRO
                        variación = cantidad-inventario - cuenta-física
                        HACER cantidad-inventario igual a cuenta-ísica
                        ENCONTRAR libro en LIBROS que corresponda con clave-libro
                        LEER registro de libro
                        RESTAR variación de total-en-existencia
                        MIENTRAS haya más pedido en PEDIDOS con
                        fecha-pedido posterior a a-fecha y que corresponda
                        con identificación-bodega
                                LEER registro de pedido
                                MIENTRAS haya más artículo-pedido en
                                ARTICULOS-PEDIDOS con clave-libro =
                                clave-libro en detalles-pedido y número-
                                factura = número-factura en pedido
                                        LEER artículo-pedido
                                        RESTAR cantidad-pedida de cantidad-inventario
                                        RESTAR cantidad-pedida de total-en-existencia
                                FIN_MIENTRAS
                        FIN_MIENTRAS
                        ESCRIBIR registro de artículo-inventario
                        ESCRIBIR registro de libro
                FIN_SI
        FIN_REPITE
FIN_SI
TERMINA
```

PROCESO 7.5: REGISTRAR ENVIO REAL

```
COMIENZA
ENCONTRAR pedido en PEDIDOS que corresponda con número-factura
SI no se encuentra registro
        respuesta-envío = "No se puede encontrar tal pedido"
        DESPLEGAR respuesta-envío
```

OTRO
    LEER registro de **pedido**
    FIJAR **fecha-envío** a fecha actual
    ESCRIBIR registro de pedido
FIN_SI
TERMINA

PROCESO 7.6: RESPONDER A NO EN EXISTENCIAS
COMIENZA
ENCONTRAR **libro** en **LIBROS** que corresponda con título-libro
SI no se encuentra registro
    respuesta-no-en-existencia = "No existe tal libro"
    DESPLEGAR **respuesta-no-en-existencia**
OTRO
    LEER registro de libro para recuperar **clave-libro**
    ENCONTRAR **artículo-inventario** en **ARTICULOS-INVENTARIO**
            que corresponda con **identificación-bodega** y con **clave-libro**
    SI no se encuentra el registro
        respuesta-no-en-existencia = "Error: no se puede
            encontrar artículo del inventario"
    OTRO
        LEER **artículo-inventario**
        HACER **cantidad-inventario** igual a cero
        ESCRIBIR **artículo-inventario**
        respuesta-no-en-existencia = "Mensaje de no en existencia aceptado"
    FIN_SI
FIN_SI
TERMINA

PROCESO 8.1: PRODUCIR INFORME DE REGALIAS DE AUTOR
COMIENZA
MIENTRAS haya más **libro** en **LIBROS**
    total-libros = 0
    total-ventas = 0
    total-regalías = 0
    LEER siguiente registro de **libro**
    MIENTRAS haya más **pedido** en **PEDIDOS** que corresponda con
    fecha-pedido de este trimestre
        LEER siguiente registro de **pedido**
        MIENTRAS haya más **artículo-pedido** en **ARTICULOS-PEDIDOS** con
        número-factura que corresponda con **número-factura** en registro de
        pedido actual y **clave-libro** que corresponda con **clave-libro** en registro
        de **libro** actual
            LEER siguiente **artículo-pedido**
            SUMAR **cantidad-pedida** a total-libros

            esta-venta = **cantidad-pedida** * **precio-unitario** * **descuento**
            SUMAR esta-venta a total-ventas
            SUMAR (esta-venta * **tasa-regalías**) a total-regalías
        FIN_MIENTRAS
    FIN_MIENTRAS
    MIENTRAS haya más **crédito** en **CREDITOS** con **clave-libro** que corresponda
    con **clave-libro** en registro libro actual y con **fecha-crédito** en este trimestre
        LEER siguiente **crédito**
        RESTAR **cantidad-libros-a-crédito** de total-libros
        RESTAR **monto-de-crédito** de total-ventas
        RESTAR (**monto-de-crédito** * **tasa-regalías**) de **total-regalías**
    FIN_MIENTRAS
    MIENTRAS haya más **devolución** en **DEVOLUCIONES** con **clave-libro** que corresponda con **clave-libro** en registro de **libro** actual y fecha de devolución en este trimestre
        LEER siguiente **devolución**
        RESTAR **cantidad-devuelta** de total-libros
        RESTAR **valor-devolución** de total-ventas
        RESTAR (**valor-devolución** * **tasa-regalías**) de total- regalías
    FIN_MIENTRAS
    AÑADIR total-libros,total-ventas,total-regalías al
            siguiente renglón de **informe-regalías-autor**
FIN_MIENTRAS
DESPLEGAR **informe-regalías-autor**
TERMINA

PROCESO 8.2: AVISAR A CONTABILIDAD SOBRE ADELANTO DE REGALIAS
COMIENZA
ENCONTRAR **autor** en **AUTORES** con **identificación-autor** que
            corresponda con **identificación-autor** en **solicitud-adelanto-regalías**
SI no se encuentra registro
    respuesta-adelanto = "No existe tal autor"
DESPLEGAR **respuesta-adelanto**
OTRO
    ENCONTRAR **libro** en **LIBROS** con **clave-libro** que
            corresponda con **clave-libro** en **solicitud-adelanto-regalías**
    SI no se encuentra registro
        respuesta-adelanto = "No existe tal libro"
        DESPLEGAR **respuesta-adelanto**
    OTRO
        solicitud-autorización-adelanto = **solicitud-adelanto-regalías**
        DESPLEGAR (para administración) **solicitud-autorización-adelanto**
        ACEPTAR (de la administración) **respuesta-autorización-adelanto**

```
            SI respuesta-adelanto = "SI"
                  respuesta-adelanto = "Adelanto aprobado"
                  DESPLEGAR respuesta-adelanto
                  solicitud-cheque-adelanto = solicitud-adelanto-regalías
                  DESPLEGAR (para contabilidad) solicitud-cheque-adelanto
                  LEER registro de autor
                  SUMAR adelanto a saldo-regalías
                  ESCRIBIR registro de autor
            OTRO
            respuesta-adelanto = "Adelanto negado"
                  DESPLEGAR respuesta-adelanto
            FIN_SI
      FIN_SI
FIN_SI
TERMINA
```

PROCESO 8.3: CAMBIAR DETALLES AUTOR

```
COMIENZA
ENCONTRAR autor en AUTORES con identificación-autor correspondiente
SI se encuentra registro
      ESCRIBIR detalles-autor en autor
FIN_SI
TERMINA
```

APENDICE G

CASO DE ESTUDIO: EL PROBLEMA DEL ELEVADOR

G.1 INTRODUCCION

Este apéndice muestra el modelo esencial para el controlador de un elevador. Su propósito principal es usar los modelos del análisis estructurado para sistemas de tiempo real; se verán ejemplos de flujos de control, procesos de control y diagramas de entidad-relación que normalmente no se usarían en un sistema orientado a los negocios.

En la siguiente sección se da una descripción narrativa del problema. Enseguida hay varios diagramas que conforman el modelo esencial, además del diccionario de datos y las especificaciones del proceso. Observe que la mayoría de las especificaciones de proceso usan el enfoque de precondición/postcondición que se discute en el capítulo 11.

El problema del elevador se usó en un curso patrocinado por la sección Washington, D.C., de la ACM en 1986. Los modelos que se proporcionan aquí originalmente los desarrolló Dennis Stipe, ex-empleado de YOURDON, inc. Los diagramas de flujo de datos y el diccionario de datos se hicieron con ayuda de una computadora Macintosh II con el software *MacBubbles* de StarSys, Inc.; los diagramas de transición de estados se hicieron con *MacDraw.*

Es importante que vea lo mucho que difieren los diagramas de este capítulo de los diagramas del Apéndice F, que se produjo con *Design* de Meta Software. *MacBubbles* es un producto CASE específicamente hecho para el dibujo de diagramas de flujo de datos (con capacidad de balancear diagramas padres e hijos, etc.). *Design* es un programa de dibujo de propósito más general, orientado a objetos, que se

puede usar para dibujar diagramas de flujo, diagramas de flujo de datos, o casi cualquier otro diagrama de software. Desde un punto de vista estético, los diagramas que producen ambos son muy distintos; creo que los editores que produjeron este libro hubieran preferido un artista humano confiable a cualquiera de estos paquetes. Como se mencionó en el capítulo 9, el estilo y formato de los diagramas de flujo de datos puede ser una cuestión controvertida y delicada con muchos usuarios; cuando compare los apéndices F y G, verá por qué.

G.2 UNA DESCRIPCION NARRATIVA

El requerimiento general es diseñar e implantar un programa para controlar y programar el itinerario de cuatro elevadores en un edificio de 40 pisos. Los elevadores se usarán para transportar personas de un piso a otro en forma convencional.

Eficiencia: El programa debe dar el itinerario de los elevadores de una manera eficiente y razonable. Por ejemplo, si alguien llama a un elevador oprimiendo el botón de "bajar" en el cuarto piso, el siguiente elevador que pase por el cuarto piso y que vaya hacia abajo debe parar allí para aceptar al o los pasajeros. Por otro lado, si un elevador no tiene pasajeros (es decir, no existen solicitudes de destino pendientes), debe estacionarse en el último piso al que llegó, hasta que se vuelva a ocupar. Un elevador no debe invertir su dirección de viaje hasta que hayan llegado a su destino los pasajeros que desean viajar en la dirección en ese momento. (Como se verá a continuación, el programa en realidad no puede obtener información acerca de los pasajeros de un elevador dado; sólo se entera de que se oprimieron los botones de destino para cada elevador. Por ejemplo, si algún pasajero travieso o psicópata aborda el elevador en el primer piso y luego oprime los botones del cuarto, quinto y vigésimo pisos, el programa hará que el elevador viaje hasta éstos y se detenga en cada uno de ellos. La computadora y su programa no tienen información respecto a la entrada y salida de pasajeros de un elevador. Un elevador que esté lleno hasta su máxima capacidad no debe de responder a un nuevo llamado. (Existe un sensor de peso para cada elevador. La computadora y su programa los pueden interrogar).

Botón de destino: El interior de cada elevador tiene un tablero con 40 botones, uno para cada piso, etiquetados con los números de los pisos (1 a 40). Estos botones de destino se iluminan mediante señales enviadas al tablero por la computadora. Cuando un pasajero oprime un botón de destino *que aún no esté iluminado,* los circuitos del tablero mandan una interrupción a la computadora (hay una interrupción diferente para cada elevador). Cuando la computadora recibe una de estas interrupciones (vectoriales), su programa puede leer los registros de entrada de 8 bits de memoria asignada apropiados (hay uno para cada interrupción y, por tanto, uno para cada elevador) que contiene el número del piso correspondiente al botón de destino que ocasionó la interrupción. Desde luego, los circuitos del tablero escriben el número de piso indicado en el registro de entrada de 8 bits indicado que ocasiona la interrupción vectorial. (Como en esta aplicación hay 40 pisos, sólo se usarán los primeros 6 bits de cada registro de entrada para la implantación; pero el hardware podría servir para un edificio de hasta 256 pisos.)

Luces de botones de destino: Como se mencionó antes, los botones de destino se pueden iluminar (por medio de focos atrás del tablero). Cuando la rutina de interrupción de servicio del programa recibe la interrupción de un botón de destino, debe mandar una señal al tablero indicado para que se ilumine el botón indicado. Esta señal la envía el programa que carga el número del botón en el registro de salida apropiado (cada elevador tiene uno de estos registros). La iluminación de un botón indica al pasajero que el sistema ha tomado nota de su solicitud, y también evita más interrupciones debidas a que nuevamente se oprima (¿impacientemente?) el botón. Cuando el controlador detiene el elevador en un piso, debe enviar una señal a su tablero de destino para apagar el botón de destino de dicho piso.

Sensores de piso: Hay un interruptor de sensor de piso en cada piso para cada tiro de elevador. Cuando un elevador se acerca a 8 pulgadas de un piso, un engrane cierra el interruptor y manda una interrupción a la computadora (existe una interrupción diferente para cada juego de interruptores de cada tiro). Cuando la computadora recibe una de estas interrupciones (vectoriales), su programa puede leer el registro apropiado (existe uno para cada interrupción, y por tanto uno para cada elevador) que contiene el número de piso correspondiente al sensor que causó la interrupción.

Luces de llegada: El interior de cada elevador tiene un tablero que contiene un indicador iluminable para cada número de piso. Se localiza arriba de las puertas. Su propósito es informar a los pasajeros el número del piso al que está llegando (y en el cual es posible que se vaya a parar). El programa debe iluminar el indicador de un piso cuando llega a él y apagarlo cuando lo deja o llega a otro. Esta señal la envía cargando el número del indicador de piso en el registro de salida apropiado (hay uno para cada elevador).

Botón de llamada: En cada piso hay un tablero con boton(es) de llamada. A excepción de la planta baja (piso 1) y el piso más alto (piso 40) un piso tiene dos botones, uno de SUBIR y uno de BAJAR. En la planta baja sólo existe el de SUBIR, y en la más alta sólo existe el de BAJAR. Por ello, en total hay 78 botones de llamada: 39 de subida y 39 de bajada. Los pasajeros los pueden oprimir para llamar a un elevador. (Desde luego, no pueden llamar a *uno* en particular. El programador de itinerario decide cuál debe responder al llamado). Estos botones de llamada se pueden iluminar con señales enviadas de la computadora al tablero. Cuando un pasajero oprime un botón que *aún no esté iluminado,* los circuitos del tablero envían una interrupción vectorial a la computadora (hay una interrupción para botones de BAJAR y otra para SUBIR). Cuando la computadora recibe una de estas dos, su programa lee el registro apropiado, que contiene el número de piso correspondiente al botón de llamada que causó la interrupción. Desde luego, los circuitos del tablero escriben el número del piso en el registro apropiado cuando se causa la interrupción vectorial.

Luces de los botones de llamada: Los botones de llamada se pueden iluminar (por medio de focos detrás de los tableros). Cuando la rutina de interrupción de servicio por botón de llamada en el programa recibe una interrupción vectorial de SUBIR o BAJAR, debe mandar una señal al tablero apropiado para iluminar el botón en cuestión. El programa envía esta señal cargando el número del botón en el registro apropiado, uno para los botones de SUBIR y otro para los de BAJAR. La iluminación de un botón informa al o los pasajeros que el sistema ha tomado nota de su solicitud y también evita más interrupciones causadas por oprimir repetidamente el botón. Cuando el controlador detiene al elevador en un piso dado, manda una señal al tablero de botones de llamada de dicho piso para que se apague el botón en cuestión (SUBIR o BAJAR).

Controles del motor del elevador (Arriba, Abajo, Parar): Existe una palabra de control para cada motor. El bit 0 de esta palabra obliga al elevador a subir, el bit 1 a bajar, y el bit 2 a detenerse en el piso cuyo interruptor sensor esté cerrado. El mecanismo del elevador no obedecerá a ninguna orden inapropiada o peligrosa. Si ningún interruptor sensor de piso está cerrado, el mecanismo ignora la señal de detenerse hasta que alguno se cierre. El programa no tiene que preocuparse por controlar las puertas de algún elevador, o de detenerlo exactamente en cierta posición ("home") respecto a un piso dado. El fabricante de los elevadores usa interruptores, relevadores, circuitos y seguros convencionales para dicho propósito, para verificar su seguridad sin hacer caso de la computadora controladora. Por ejemplo, si la computadora da una orden de detenerse a un elevador cuando éste está a 8 pulgadas de un piso (de manera que su interruptor de sensor de piso esté cerrado), el mecanismo convencional aprobado detiene y acomoda al elevador en dicho piso, abre y mantiene sus puertas abiertas de manera apropiada y luego las cierra. Si la computadora da alguna orden durante este período (por ejemplo, cuando la puerta está abierta), el mecanismo del fabricante la ignora hasta que se cumplan las condiciones para el movimiento. (Por eso no es peligroso que la computadora dé la orden de subir o bajar cuando todavía están abiertas las puertas.) Una condición para el movimiento de un elevador es que su botón de "detenerse" no esté oprimido. Cada tablero de destino de un elevador contiene un botón de éstos. No está vinculado con la computadora. Su único propósito es detener el elevador en algún piso, con la puerta abierta, mientras está estacionado allí. El interruptor rojo de emergencia detiene al elevador en el siguiente piso que pase sin considerar el itinerario de la computadora. Este interruptor puede también encender una alarma audible, y tampoco tiene conexión con la computadora.

Máquina-destino: Todo esto se puede implantar en cualquier microcomputadora contemporánea capaz de manejar esta aplicación.

G.3 EL MODELO ESENCIAL

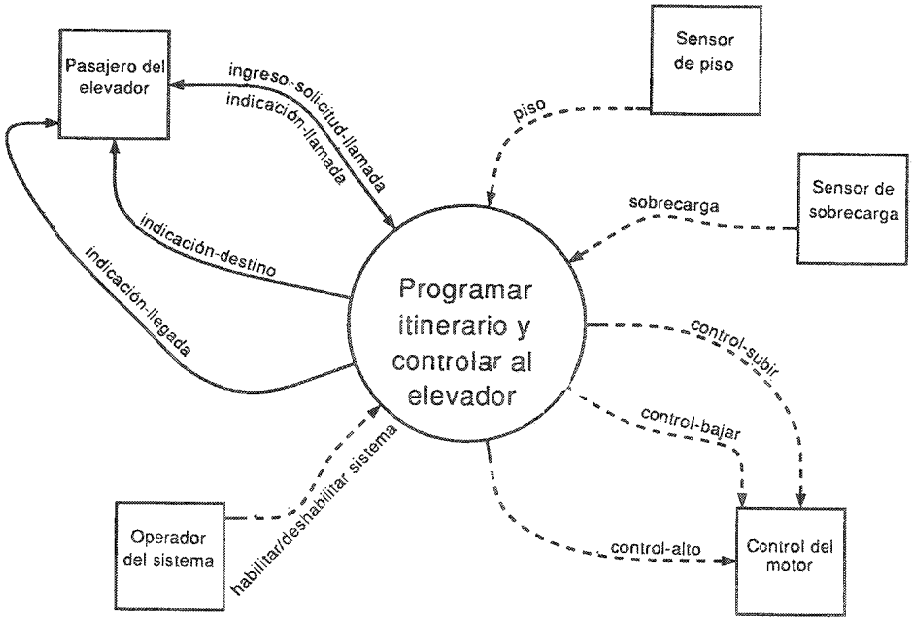


Diagrama de contexto
Modelo esencial de un elevador

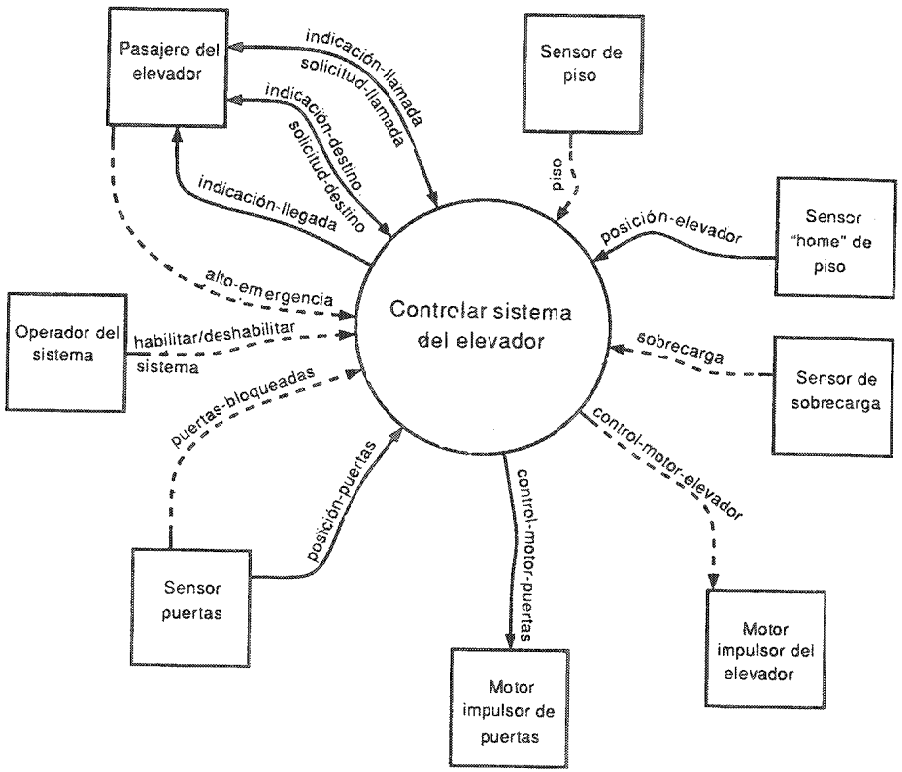


Diagrama de contexto expandido

La Lista de Acontecimientos

1. Un pasajero hace solicitud de llamada de subida.
2. Un pasajero hace solicitud de llamada de bajada.
3. El elevador llega al piso donde se le llamó.
4. El elevador no está disponible para la solicitud de llamada.
5. El elevador llega a estar dosponible para la solicitud de llamada.
6. Un pasajero hace una solicitud de destino.
7. El elevador llega al destino solicitado.
8. El elevador llega a un piso.
9. El elevador deja dicho piso.
10. El elevador no se mueve (se descompone).
11. El elevador entra en servicio nuevamente.
12. El elevador se sobrecarga.
13. Se normaliza la carga del elevador.

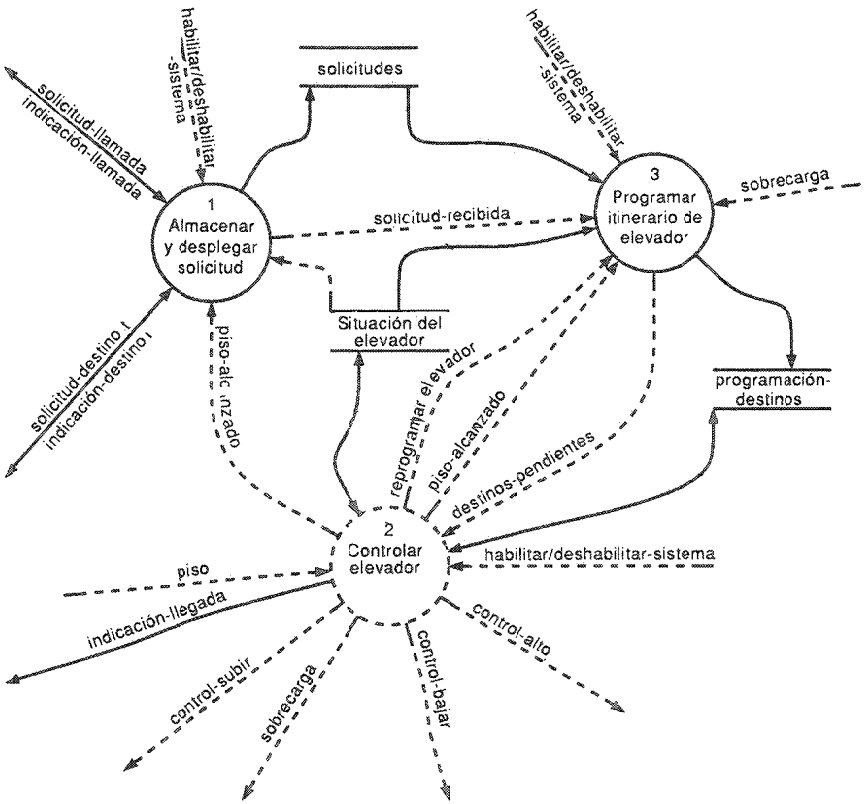


Figura 0: Programar itinerario y controlar al elevador: Modelo esencial del elevador

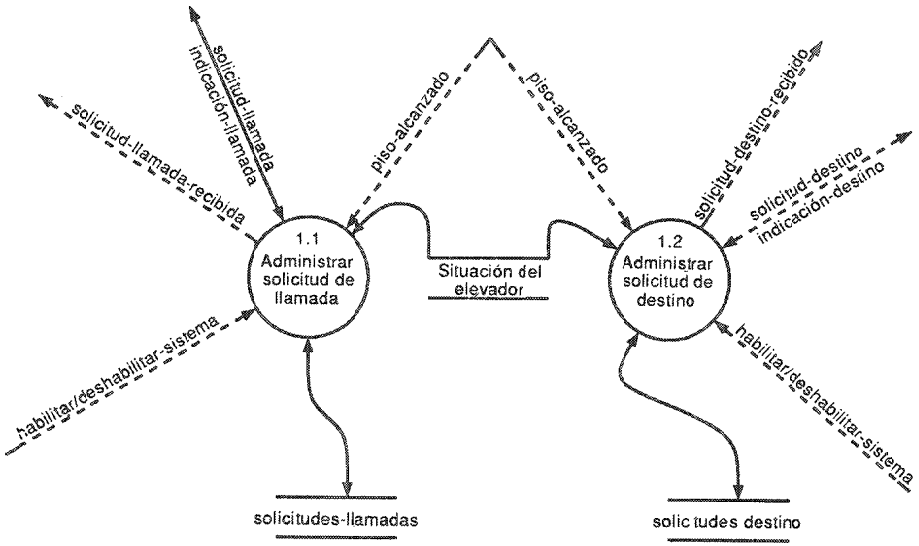


Figura 1: Almacenar y desplegar solicitud

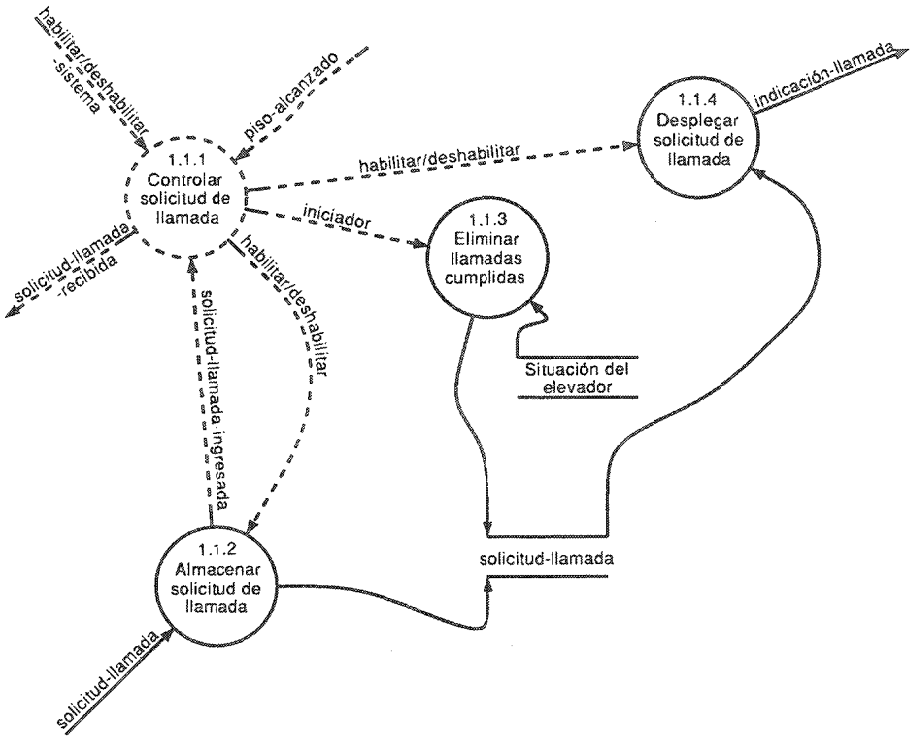


Figura 1.1: Administrar solicitud de llamada

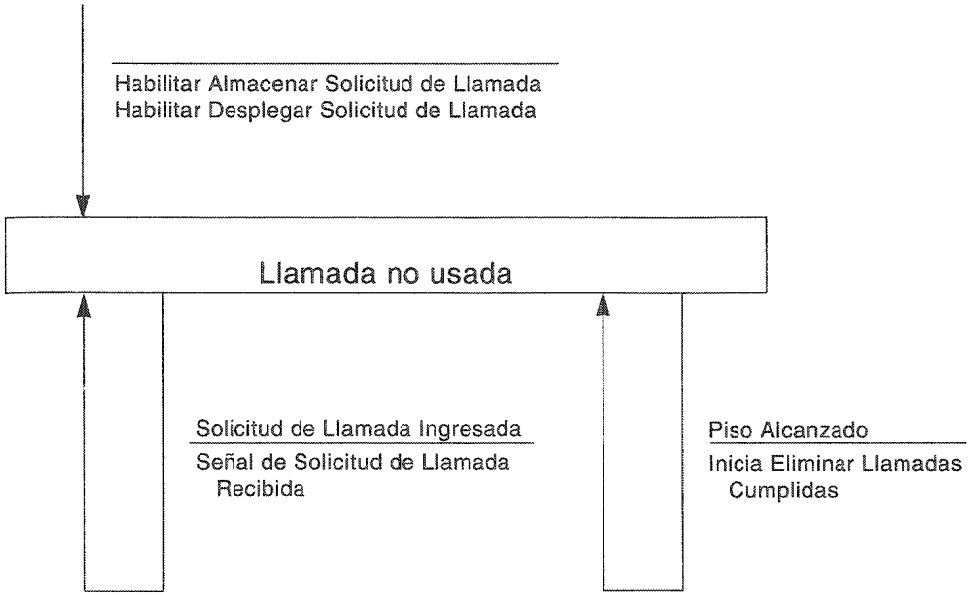


Figura 1.1.1: Controlar solicitud de llamada

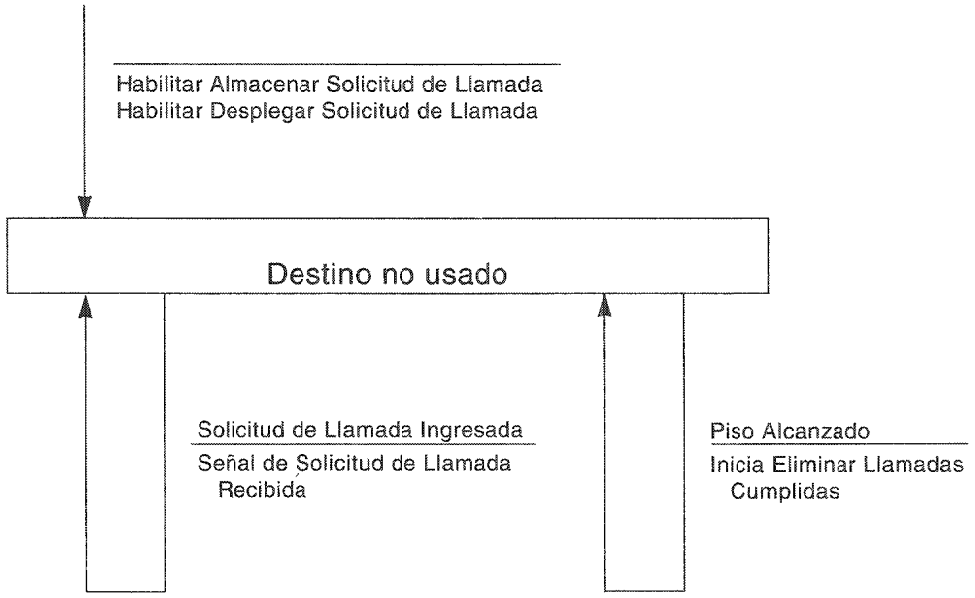


Figura 1.2.1: Controlar solicitud de destino

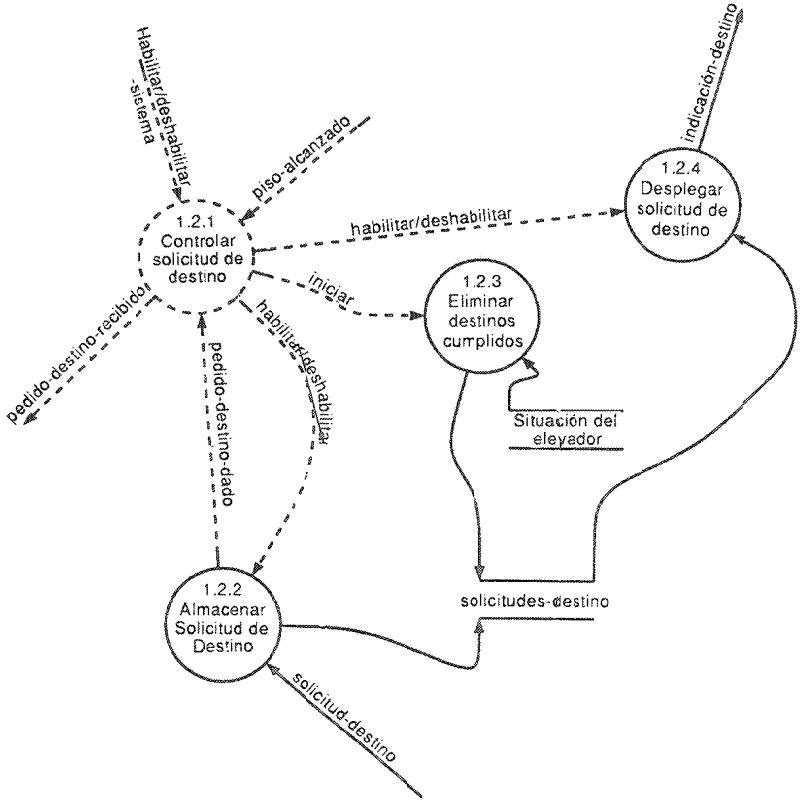


Figura 1.2: Administrar solicitud de destino

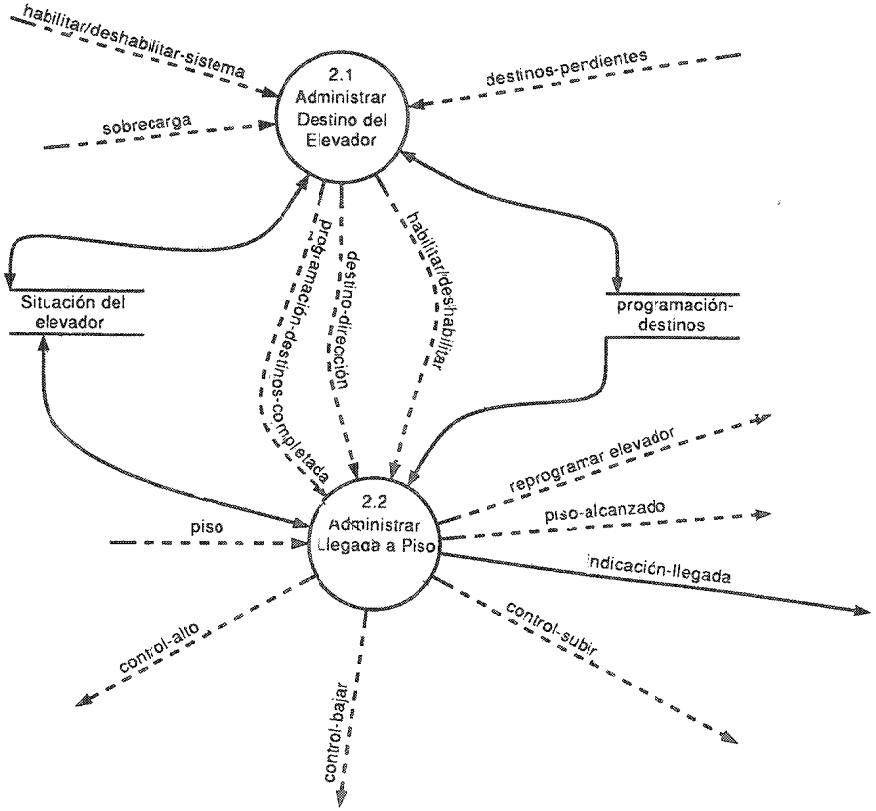


Figura 2: Controlar el elevador

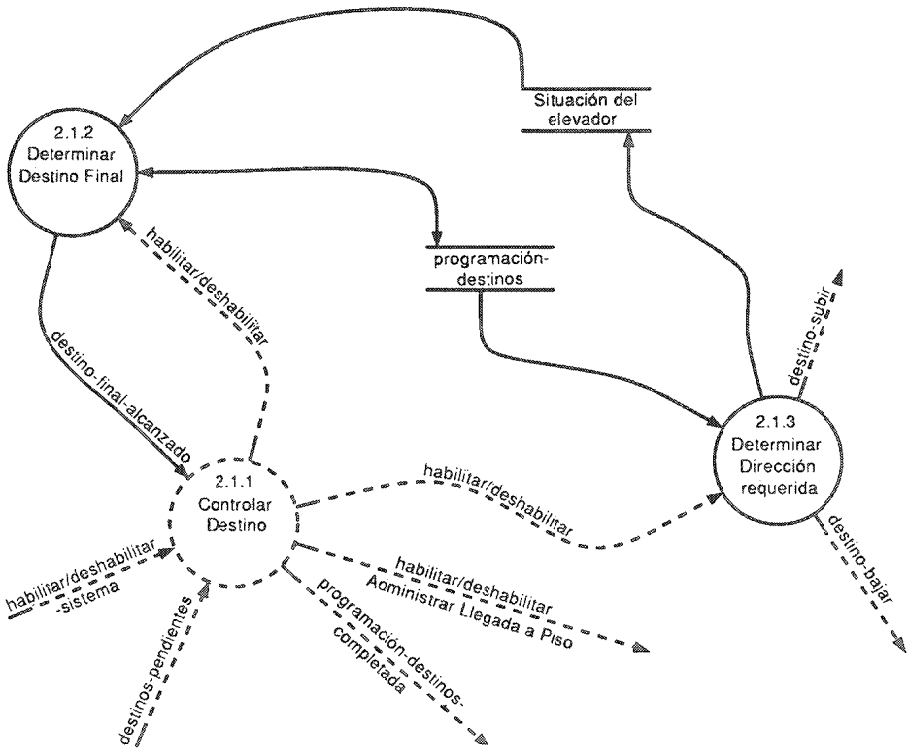


Figura 2.1: Administrar destino del elevador

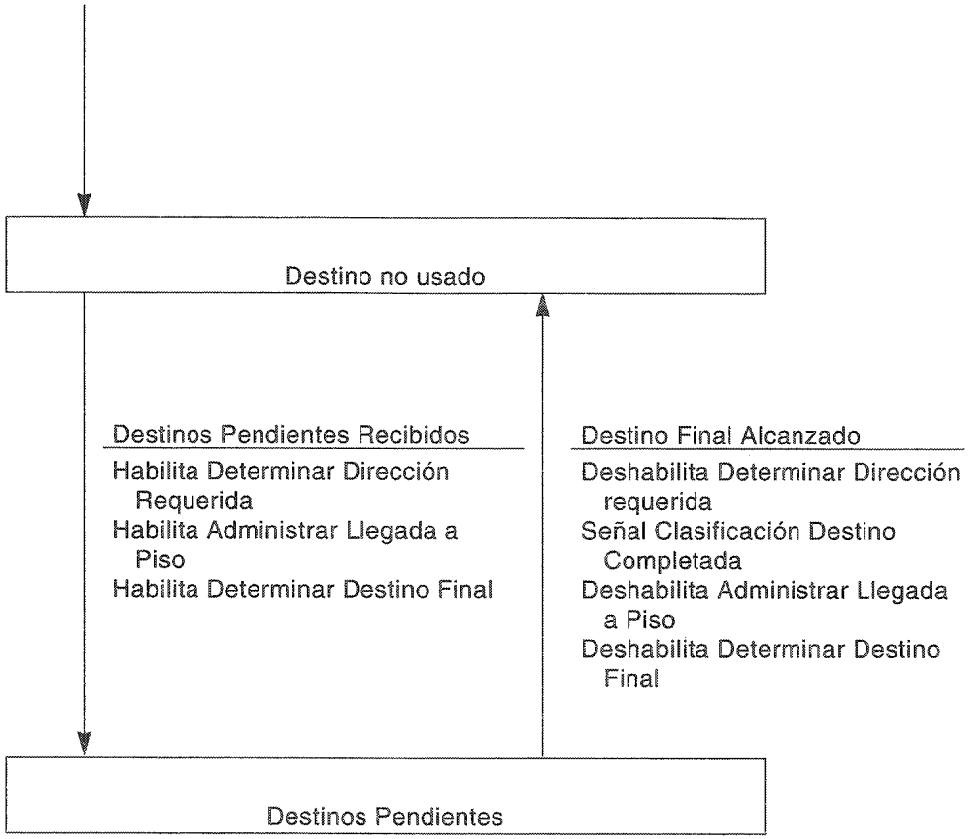


Figura 2.1.1: Controlar destino

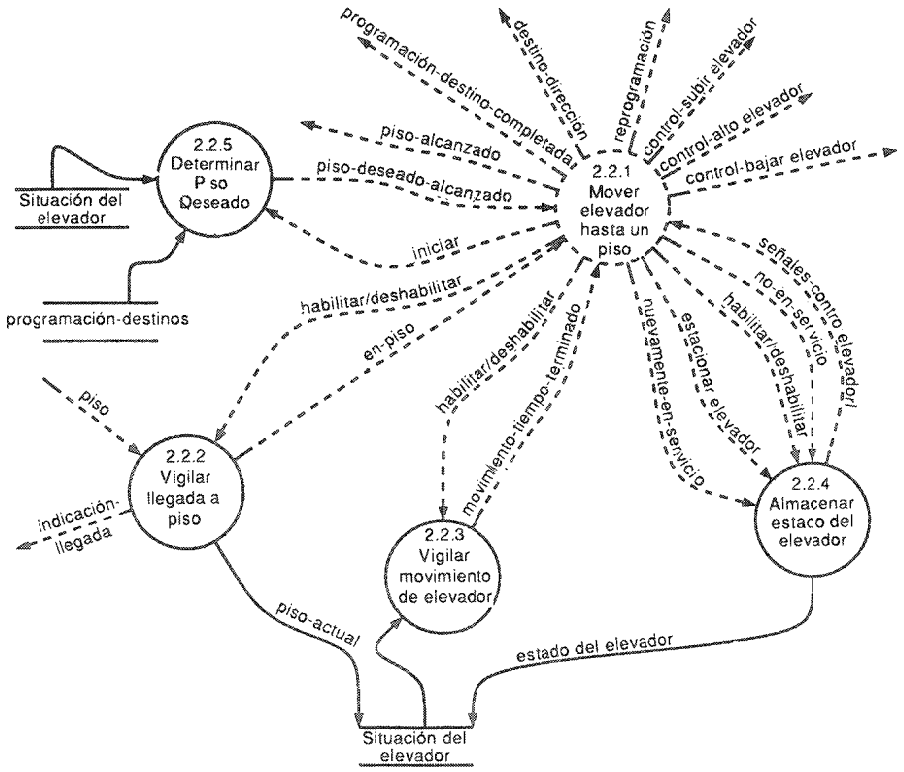


Figura 2.2: Administrar llegada a piso

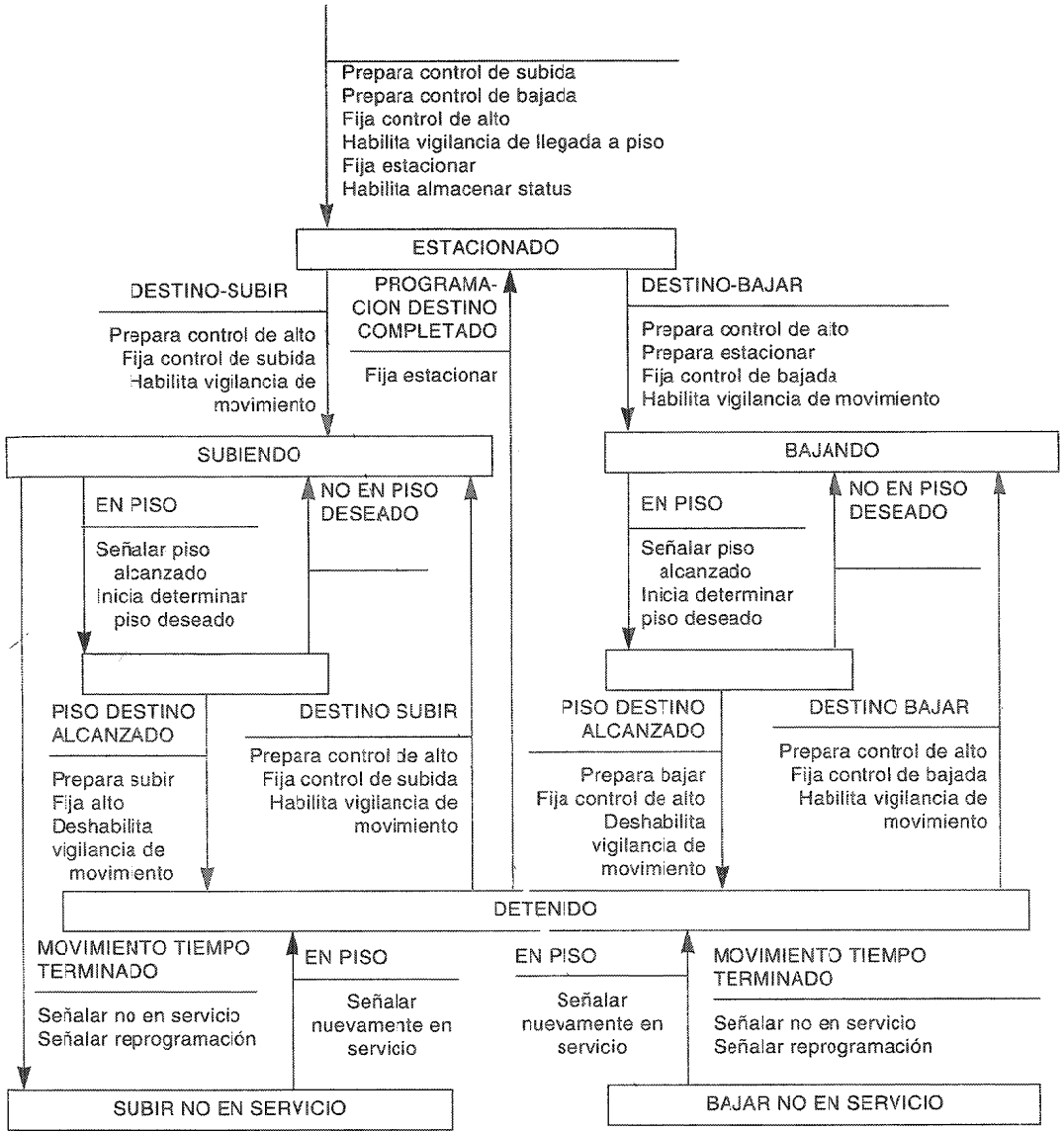


Figura 2.2.1: Mover elevador hasta piso

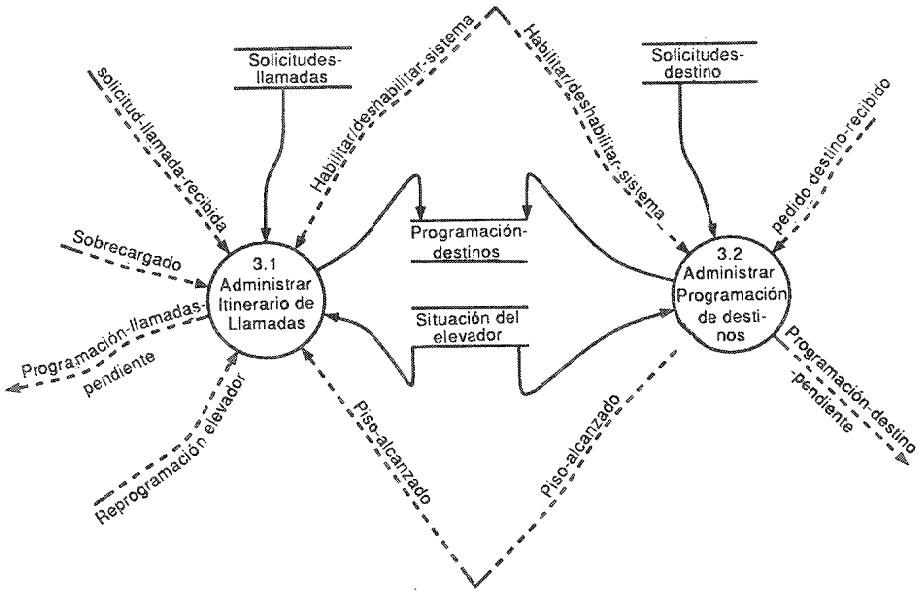


Figura 3: Programar itinerario

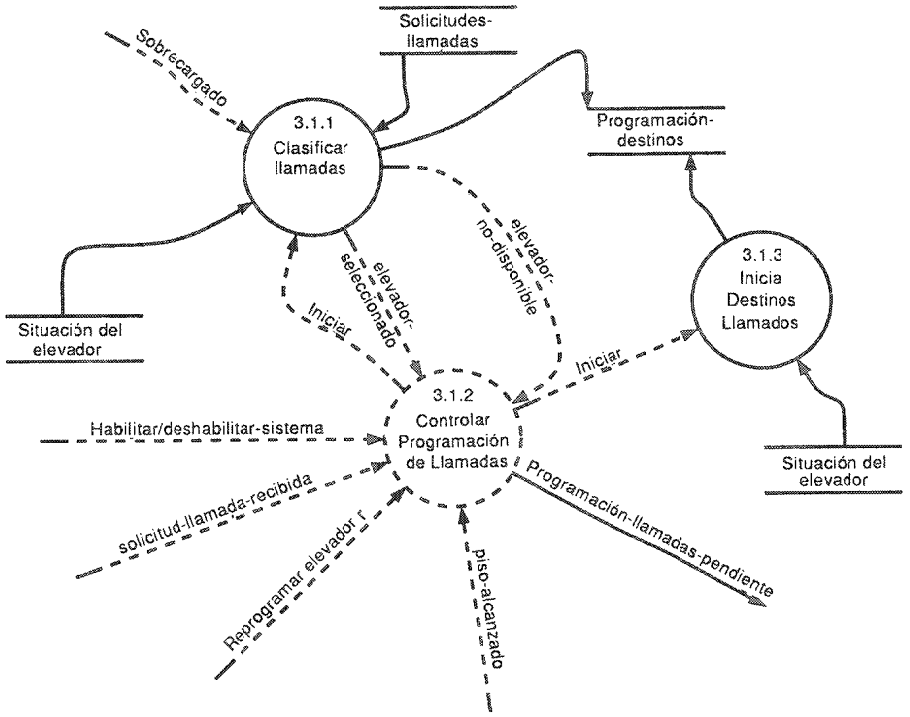


Figura 3.1: Administrar programación de llamadas

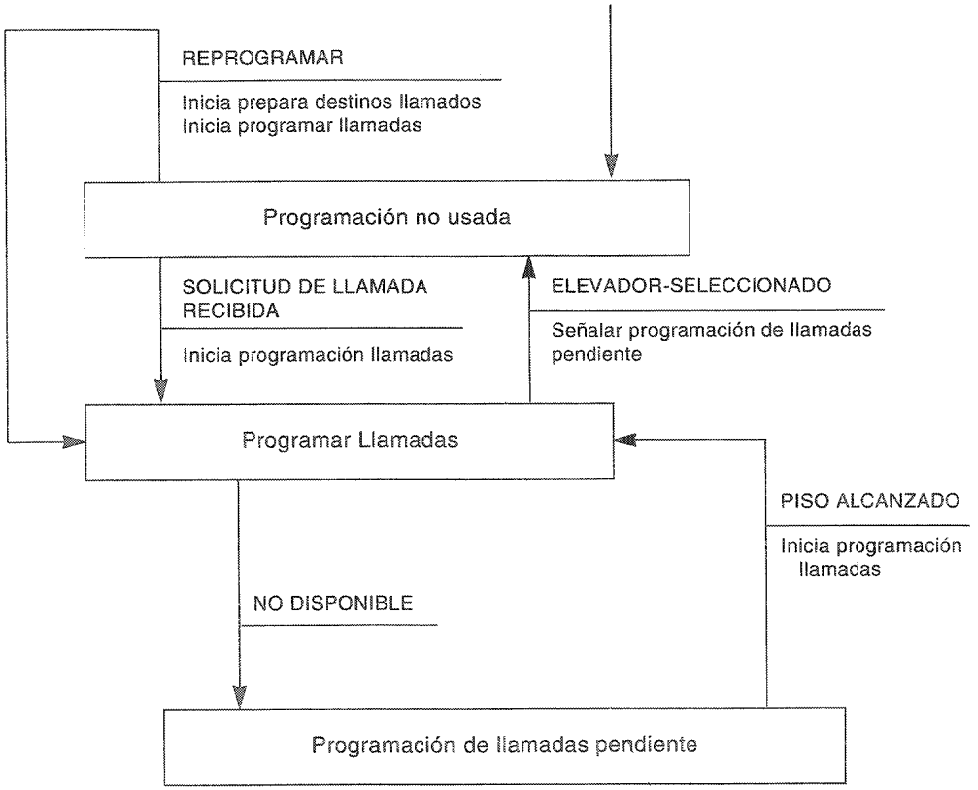


Figura 3.1.2: Controlar programación de llamadas

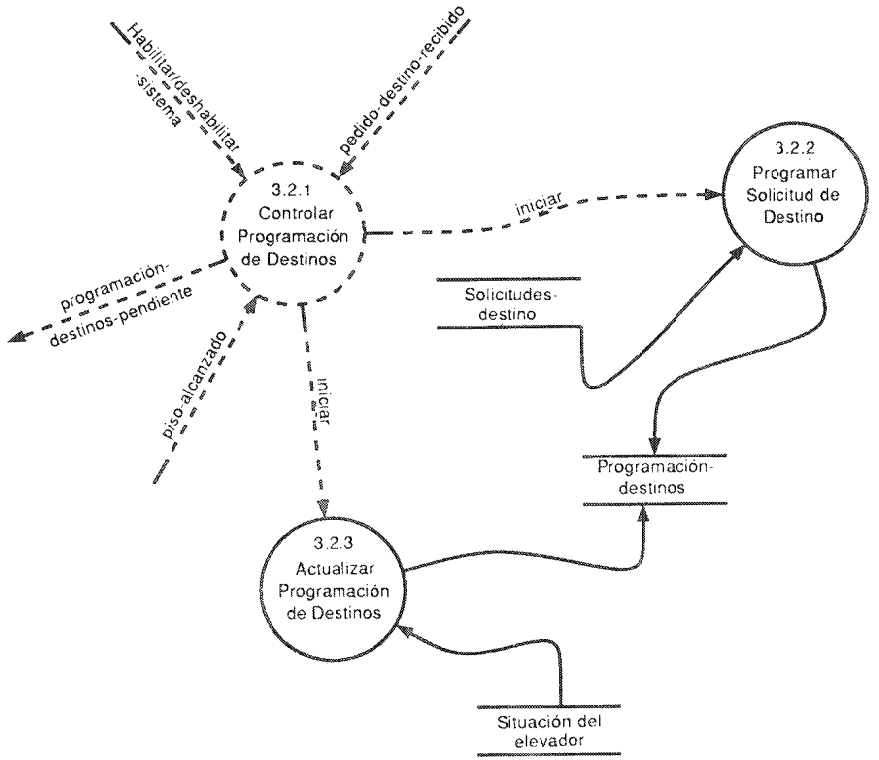


Figura 3.2: Administrar programación de destinos

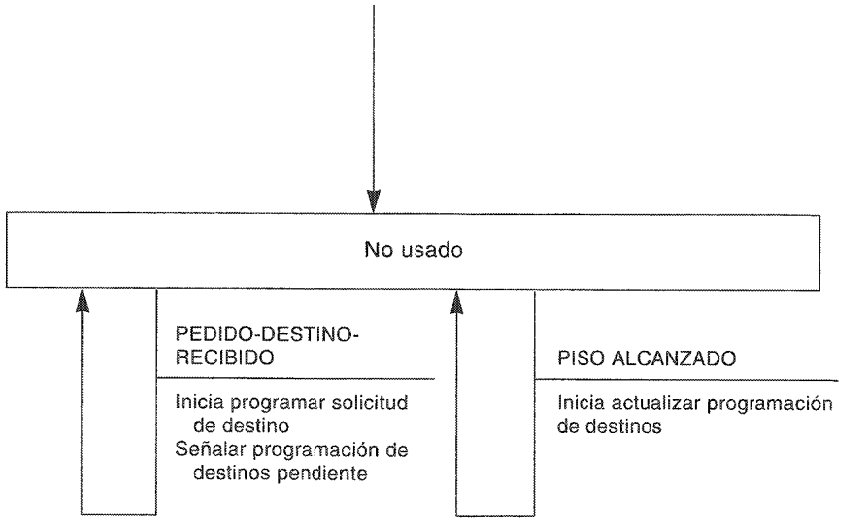


Figura 3.2.1: Controlar programación de destinos

DICCIONARIO DE DATOS
para
Sistema de Control del Elevador

apagado
NOTAS: ***Entrada Generada****

bajando
NOTAS: ***Entrada Generada****

control-alto
NOTAS: señal para hardware

control-bajar
NOTAS: señal para el hardware

control-subir
NOTAS: ***Entrada Generada****

destino-bajar
NOTAS: señala que la dirección requerida es hacia abajo

destino-dirección
NOTAS: [destino-subir
|destino-bajar]

destino-pendiente = valores: [encendido|apagado]
NOTAS: indicación de que un elevador tiene destinos subsecuentes al piso actual

destinos-pendientes = [llamadas-pendientes|destino-pendiente|llamadas-pendientes + destinos-pendientes]
NOTAS:señala que existe una programación de destinos

destino-subir
NOTAS: ***Entrada Generada***

detenido
NOTAS: ***Entrada Generada****

dirección-bajar
NOTAS: ***Entrada Generada****

dirección-subir
NOTAS: ***Entrada Generada***

elevador-número = valores: 1-4

elevador-seleccionado
NOTAS: señala que se ha programado un elevador para una solicitud de llamada

encendido
NOTAS: ***Entrada Generada****

en-piso

DICCIONARIO DE DATOS Página 2
para
Sistema de Control del Elevador

NOTAS: señala que el elevador ha llegado a cierto piso

estacionado
NOTAS: señal

estado = [estacionadolsubiendolbajandoldetenidolno-en-servicio]

indicación-destino = valores:1-40
NOTAS: indicación de números de pisos en los que está programado que un elevador se detenga

indicación-llamadas = valores: 1-40
NOTAS: indicación de números de piso donde está programado que se detenga un elevador

indicación-llegada = valores: 1-40
NOTAS: indicación de piso al que ha llegado

llamadas
NOTAS: ***Entrada Generada****

llamadas-pendientes = valores: [encendidolapagado]

no-disponible
NOTAS: señala que un elevador no está disponible para satisfacer una solicitud de llamada

no-en-servicio
NOTAS: señala que un elevador no ha logrado responder a una orden de movimiento

origen-solicitud = [llamadalpiso-destinolllamada + piso-destino]

pedido-destino-dado
NOTAS: señala que el pasajero ha ingresado una petición de destino

pedido-destino-recibido
NOTAS: señala que la petición está lista para programarse

piso-actual = valores: 1-40
NOTAS: número de piso donde actualmente se encuentra un elevador

piso-deseado-alcanzado
NOTAS: señal

piso-destino = valores: 1-40

DICCIONARIO DE DATOS Página 3
para
Sistema de Control del Elevador

NOTAS: números de pisos donde está programado que un elevador se detenga

piso-elevador = valores: 1-40

piso-número = valores: 1-40
NOTAS: ***Entrada Generada****

programación-destino = @elevador-número + {piso - destino} + origen-solicitud + destino-pendiente

programación-destinos = {programación-destino} reprogramación
NOTAS: señal de iniciar reprogramación de llamadas asignadas a elevador fuera de servicio

señal-control = control-subir + control-bajar + control - alto

sobrecarga
NOTAS: señal del hardware

solicitud = solicitudes-llamada + solicitudes-destino

solicitud-destino = @número-elevador + {número-piso}

solicitudes-destino = {solicitud-destino}

solicitud-llamada = @piso-elevador + [dirección-subirl dirección-bajarldirección-subir + dirección-bajar] + elevador-número

solicitud-llamada-dada
NOTAS: señal

solicitud-llamada-recibida
NOTAS: señal

solicitudes-llamadas = {solicitud-llamada}

solicitud-recibida = solicitud-llamada-recibida + pedido- destino-recibido

status = @elevador-número + estado + piso-actual

statuses = {status}

subiendo
NOTAS: ***Entrada Generada****

valores
NOTAS: ***Entrada Generada***

Especificaciones de Proceso

1.1.2 ALMACENAR SOLICITUD DE LLAMADA
Precondición
ocurre **ingreso-solicitud-llamada**
Postcondición
se almacena **ingreso-solicitud-llamada**
se produce **ingreso-solicitud-llamada**

1.1.3 ELIMINAR LLAMADA COMPLETADA
Precondición
Existe un **elevador-número** en **statuses** que corresponde con **elevador-número-asignado** en **solicitud-llamada**
y
Existe un **piso-actual** en statuses que corresponde con **piso-número** en **solicitudes-llamada**
Postcondición
La entrada correspondiente en **solicitud-llamada** es nula

1.1.4 DESPLEGAR SOLICITUD LLAMADA
Precondición
Ninguna
Postcondición
Se despliegan **solicitudes-llamada**

1.2.3 ELIMINAR DESTINOS COMPLETADOS
Precondición
Existe un **elevador-número** en **statuses** que corresponde con **elevador-número** en **solicitudes-destino**
y
Existe un **piso-actual** en **statuses** que corresponde con **piso-actual** en **solicitudes-destino**
Postcondición
La entrada correspondiente en **solicitudes-destino** es nula

1.2.4 DESPLEGAR SOLICITUD-DESTINO
Precondición
Ninguna
Postcondición
Se despliegan **solicitudes-destino**

2.1.2 DETERMINAR DESTINO FINAL
Precondición
Existe un **elevador-número** en **statuses** que corresponde con **elevador-número** en **programación-destinos**

y
Existe un **piso-actual** en **statuses** que corresponde con **piso-destino** y **programación-destinos**
y
el correspondiente **destino-pendiente** = **apagado** en **programación-destinos**
Postcondición
Se produce **destino-final-alcanzado**

2.1.3 DETERMINAR DIRECCION REQUERIDA
El término local **correspondencia** es un **elevador-número** que corresponde con **programación-destinos** y **elevador-número** en **status**
Precondición 1
Existe **correspondencia**
y
Existe en **programación-destinos** un **piso-destino** > **piso-actual** en **status**
Postcondición 1
Se produce **destino-subir**
Precondición 2
Existe **correspondencia**
y
Existe en **programación-destinos** un **piso-destino** < **piso-actual** en **status**
Postcondición 2
Se produce **destino-bajar**

2.2.2 VIGILAR LLEGADA A PISO
Precondición 1
Ocurre **piso**
Postcondición 1
Se elimina **indicación-llegada** en piso anterior
y
se produce **indicación-llegada** para piso correspondiente
y
se produce **en-piso**
y
se actualiza **piso-actual** en **statuses**

2.2.3 VIGILAR MOVIMIENTO DE ELEVADOR
Se lee **piso-actual** de **statuses**
Precondición
Transcurren 10 segundos y **piso-actual** no cambia
Postcondición
Ocurre **movimiento-tiempo-terminado**

2.2.4 ALMACENAR STATUS
Precondición
Se recibe señal de entrada
Postcondición
Se actualiza estado en **status**

2.2.5 DETERMINAR PISO DESEADO
Precondición
Existe un **elevador-número** en **statuses** que corresponde con **elevador-número** en **statuses-destino**
y
Existe un **piso-actual** correspondiente en **statuses** que corresponde con **piso-destino** en **programación-destinos**
Postcondición
Se produce **piso-deseado-alcanzado**

3.1.1 PROGRAMAR LLAMADAS
COMIENZA
con **solicitud-llamada, status,** y **sobrecarga**
MIENTRAS **elevador-seleccionado** no se haya señalado
Encontrar **elevador** más cercano
SI **elevador** se está moviendo en la **dirección** correcta o **elevador** está **estacionado**
SI **elevador** no está **sobrecargado**
ingresar **solicitud-llamada** por medio de **elevador-número** en **clasificación-destino**
hacer **origen-solicitud** igual a llamada o **llamada + destino**
FIN_SI
SI **destino-pendiente** = **apagado**
hacer **destino-pendiente** = **encendido**
FIN_SI
señalar **elevador-seleccionado**
OTRO
encontrar **elevador** más cercano
FIN_MIENTRAS
SI no se encuentra **elevador**
Señalar **elevador** no disponible
FIN_SI
TERMINA

3.1.3 ELIMINAR DESTINOS LLAMADOS
Precondición
Existe un **elevador-número** en **statuses** que corresponde con **elevador-número** en **programación-destinos**

y
el correspondiente **estado** = **no-en-servicio** en **statuses**
y
el correspondiente **origen-solicitud** = llamada en **programación-destinos**
Postcondición
Las entradas correspondientes en **piso-destino** son nulas

3.2.2 PROGRAMAR SOLICITUD-DESTINO
Precondición
Ninguna
Postcondición
programación-destinos se actualiza con **solicitudes-destino** que corresponden con **elevador-número**
Hacer **origen-solicitud** = **destino** o **llamada** + **destino**
SI **destino-pendiente** = **apagado**
hacer **destino-pendiente** = **apagado**
FIN_SI

3.2.3 ACTUALIZAR PROGRAMACION DESTINOS
Precondición 1
Existe un **elevador-número** en **statuses** que corresponde con **elevador-número** en **programación-destinos**
y
Existe un **piso-actual** en **statuses** que corresponde con **piso-destino** en **programación-destinos**
Postcondición 1
La correspondiente entrada en **piso-destino** es nula
Precondición 2
misma condición que precondición 1
y
no existe entrada correspondiente en **piso-destino**
Postcondición 2
La entrada correspondiente en **piso-destino** es nula
y
se hace **destino-pendiente** = **apagado**

INDICE