

ASEGURAMIENTO DE LA CALIDAD MEDIANTE INGENIERÍA DE SOFTWARE

16

OBJETIVOS DE APRENDIZAJE

Una vez que haya dominado el material de este capítulo, podrá:

1. Reconocer la importancia de adoptar un enfoque de calidad total para todo el SDLC.
2. Crear diagramas de estructura para diseñar sistemas modulares con un enfoque descendente (de arriba a abajo).
3. Usar diversas técnicas para mejorar la calidad del diseño y mantenimiento del software.
4. Entender la importancia de ejecutar una variedad de pruebas durante el desarrollo de sistemas para identificar problemas desconocidos.

La calidad ha sido durante mucho tiempo una preocupación para las empresas, como lo debe ser para los analistas de sistemas en el análisis y diseño de sistemas de información. Es demasiado arriesgado emprender todo el proceso de análisis y diseño sin usar un enfoque de aseguramiento de la calidad. Los tres enfoques para el aseguramiento de la calidad mediante ingeniería de software son: (1) garantizar el aseguramiento de la calidad total diseñando sistemas y software con un enfoque modular, descendente (de arriba a abajo); (2) documentar el software con las herramientas adecuadas, y (3) probar, mantener y auditar el software.

Dos propósitos guían el aseguramiento de la calidad. El primero es que el usuario del sistema de información es el factor individual más importante en establecer y evaluar su calidad. El segundo es que es mucho menos costoso corregir los problemas en sus fases iniciales que esperar hasta que un problema se manifieste a través de las quejas o crisis del usuario.

Ya hemos aprendido acerca de la enorme inversión de mano de obra y otros recursos empresariales que se requieren para desarrollar con éxito un sistema. El uso del aseguramiento de la calidad a lo largo del proceso es una forma de reducir los riesgos, ayuda a garantizar que el sistema resultante será lo que se necesita y desea, y mejorará notablemente algún aspecto del desempeño del negocio. Este capítulo proporciona al analista tres enfoques principales para la calidad.

ENFOQUE DE ADMINISTRACIÓN DE LA CALIDAD TOTAL

La administración de la calidad total (TQM, por sus siglas en inglés) es esencial a lo largo de todos los pasos del desarrollo de sistemas. Según Dean y Evans (1994), los principales elementos de la TQM sólo son significativos cuando se presentan en un contexto organizacio-

nal que favorece un esfuerzo integral por la calidad. Es en este contexto donde los elementos de enfoque en el cliente, planificación estratégica y liderazgo, mejora continua, facultar al empleado y trabajo en equipo se unifican con el propósito de cambiar el comportamiento de los empleados y, en consecuencia, el curso de la organización. Observe que el concepto de calidad se ha ampliado con el paso de los años para reflejar un enfoque en toda la organización, y no tan sólo en la producción. En lugar de concebir a la calidad como un control del número de artículos defectuosos que se producen, ahora se considera como un proceso evolutivo hacia la perfección que se denomina administración de la calidad total.

Los analistas de sistemas deben estar conscientes de los factores que despiertan el interés en la calidad. Es importante comprender que el creciente compromiso de las empresas hacia la TQM encaja sumamente bien en los objetivos generales del análisis y diseño de sistemas.

SEIS SIGMA

La llegada de Seis Sigma ha cambiado el enfoque de la administración de la calidad. Cada analista de sistemas necesita estar consciente de Seis Sigma y aplicar algunos de los principios a sus proyectos de análisis de sistemas. Originalmente desarrollado por Motorola en la década de 1980, Seis Sigma es más que una metodología; es una cultura basada en la calidad. La meta de Seis Sigma es eliminar todos los defectos. Esto se aplica a cualquier producto, servicio o proceso. En los libros de texto de administración de operaciones que se publicaron a partir de la década de 1970 y hasta fines del siglo pasado, el control de calidad se expresó en términos de tres desviaciones estándar de la media, o tres sigma, lo cual es equivalente a aproximadamente 67,000 defectos por millón de oportunidades. Seis Sigma implica una meta de sólo 3.4 defectos por millón de oportunidades.

Seis Sigma es un enfoque descendente de arriba a abajo. Se requiere que un CEO adopte la filosofía y un ejecutivo funja como campeón de proyecto. Un líder de proyecto de Seis Sigma se denomina *Black Belt* (cinta negra). Las personas escogidas para ser *Black Belts* pueden provenir de diferentes niveles e incluso diferentes niveles salariales, pero deben tener experiencia en el proyecto y contar con capacitación especial. Los *Black Belts* se certifican después que han liderado proyectos de manera exitosa. Los miembros del proyecto se denominan *Green Belts* (cintas verdes). Los *Black Belts* maestros son los *Black Belts* que han trabajado en muchos proyectos y están disponibles como un recurso para los equipos de proyectos. (La metáfora de *Black Belt* viene del sistema de clasificación de capacidades en las artes marciales. Resalta la importancia de la disciplina en todos los ámbitos.)

Seis Sigma se puede resumir como una metodología. En la figura 16.1 se muestran los pasos de Seis Sigma. Sin embargo, Seis Sigma es mucho más que una metodología; es una filosofía y una cultura.

Para más información sobre Seis Sigma y administración de la calidad, visite el sitio Web del Juran Center en la Carlson School of Management de la University of Minnesota en Twin Cities (www.csom.umn.edu). En 2002 el Juran Center emitió un manifiesto para apoyar y fomentar la calidad. Los autores de este libro firmaron el manifiesto en ese momento y sinceramente estamos de acuerdo con sus principios.

Joseph M. Juran dijo: “Toda mejora de la calidad ocurre proyecto tras proyecto y de ninguna otra forma” (Juran, 1964). Los analistas de sistemas y gerentes de proyecto deben tomar muy en serio esta afirmación.

RESPONSABILIDAD DE LA ADMINISTRACIÓN DE LA CALIDAD TOTAL

En términos prácticos, gran parte de la responsabilidad por la calidad de los sistemas de información recae en los usuarios de éstos y en los directivos. Para que la TQM se vuelva una realidad en los proyectos de sistemas, deben darse dos condiciones. Primera, debe existir un apoyo organizacional incondicional por parte de los directivos, lo cual es distinto a simplemente respaldar el nuevo proyecto de los directivos. Este apoyo significa establecer un con-

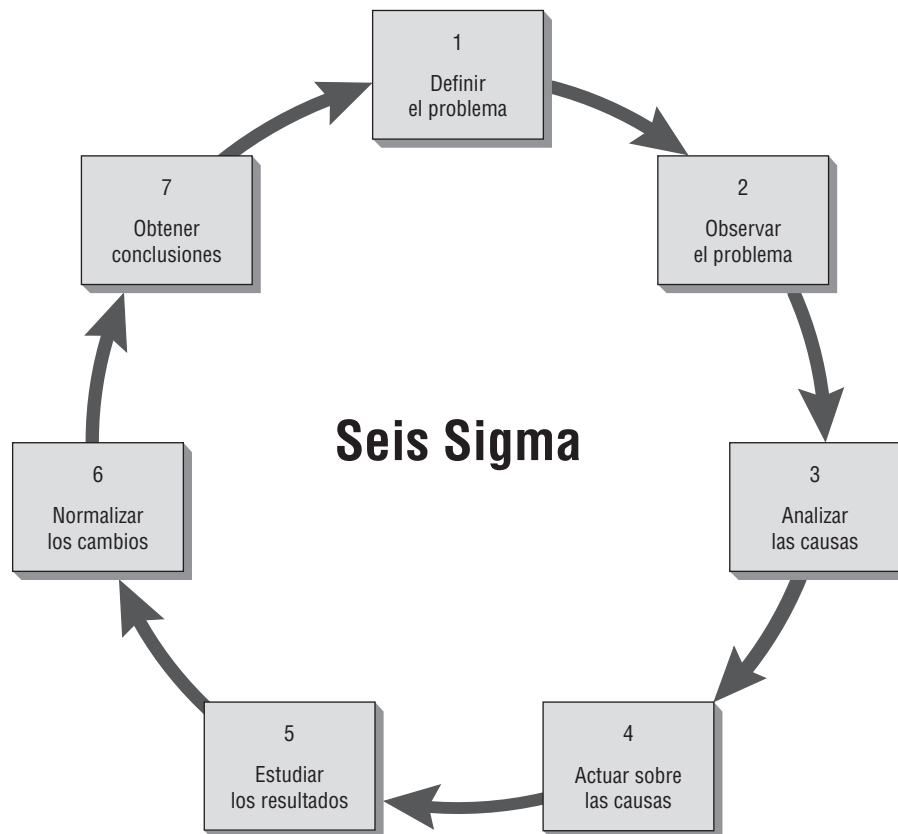


FIGURA 16.1

Cada analista de sistemas debe entender la metodología y filosofía de Seis Sigma.

texto para que los directivos consideren seriamente cómo afecta su trabajo la calidad de los sistemas de información y la información misma.

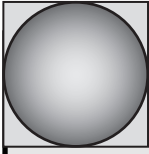
Es necesario que tanto el analista como la empresa se comprometan desde el principio con la calidad para lograr la meta de calidad. Este compromiso da como resultado un esfuerzo uniformemente controlado hacia la calidad durante todo el ciclo de vida del desarrollo de sistemas, y está en marcado contraste con tener que dedicar gran cantidad de esfuerzo para resolver problemas al final del proyecto.

El apoyo organizacional para conseguir calidad en sistemas de información de administración se puede lograr al proporcionar tiempo en el trabajo para los círculos de calidad de SI, los cuales consisten de seis a ocho pares organizacionales específicamente responsables de considerar cómo mejorar los sistemas de información y cómo implementar las mejoras.

Mediante el trabajo en los círculos de calidad de SI o a través de otros mecanismos ya colocados, la administración y usuarios deben desarrollar lineamientos para los estándares de calidad de sistemas de información. Preferentemente, los estándares se rediseñarán cada vez que un nuevo sistema o una modificación mayor se proponen formalmente por el equipo de análisis de sistemas.

No es fácil crear los estándares de calidad, pero es posible y se ha hecho. Parte del trabajo del analista de sistemas es alentar a usuarios a que cristalicen sus expectativas acerca de los sistemas de información y sus interacciones con éstos.

Los estándares de calidad departamentales se deben comunicar mediante retroalimentación para el equipo de análisis de sistemas. Normalmente el equipo está sorprendido por lo que se ha desarrollado. Las expectativas generalmente son menos complejas de lo que analistas experimentados saben se podría hacer con un sistema. Además, los problemas humanos que se han omitido o menospreciado por el equipo del analista se podrían diseñar como extremadamente urgentes en los estándares de calidad que fijen los usuarios. Involucrar a los usuarios en explicar los estándares de calidad para los sistemas de información ayudarán al analista a evitar errores costosos en el desarrollo de sistemas no deseados o innecesarios.



LA CALIDAD DE MIS NO ES OBLIGATORIA

“Merle, ven aquí y echa un vistazo a estos informes de fin de semana”, suplica Portia. Como uno de los gerentes del comité de aseguramiento de la calidad/grupo de trabajo de SI, compuesto por seis integrantes, Portia ha estado examinando la salida del sistema producida por el prototipo para su departamento de marketing. El equipo de análisis de sistemas le ha pedido que revise la salida.

Merle Chant se encamina hacia el escritorio de Portia y da una mirada a los documentos que ella le extiende. “¿Por qué?, ¿cuál es el problema?”, pregunta. “A mí me parece que están bien. Creo que le estás dando demasiada importancia a este grupo de trabajo. Se supone que también debemos hacer nuestro otro trabajo, ya sabes.” Merle da la vuelta y regresa a su escritorio ligeramente perturbado por la interrupción.

“Merle, ten un poco de compasión. Es verdaderamente tonto soportar estos informes tal como están. No puedo encontrar nada de lo que necesito, y se supone que tengo que indicarle a todos los demás en el departamento qué parte del informe deben leer. Por una parte, estoy decepcionada. Este informe es descuidado. No le encuentro ningún sentido. Es tan sólo una copia de la salida que estamos recibiendo ahora. De hecho, parece peor. Lo voy a presentar en la próxima reunión del grupo de trabajo”, manifiesta insistentemente Portia.

Merle voltea a verla y dice: “La calidad es su responsabilidad, Portia. Si el sistema no está dándonos buenos informes, ellos lo arregla-

rán cuando todo esté junto. Nada más estás provocando problemas. Estás actuando como si ellos valoraran realmente nuestra información. Yo no les dedicaría tiempo de mi día, mucho menos haría su trabajo. Ellos son inteligentes, dales la oportunidad de que deduzcan lo que necesitamos.”

Portia mira a Merle inexpressivamente, y poco a poco comienza a enfadarse. “Nosotros hemos estado en el grupo de trabajo durante cuatro semanas”, dice. “Tú has asistido a cuatro reuniones. Nosotros somos los únicos que conocemos el negocio. La idea esencial de TQM es decirles lo que necesitamos, lo que nos satisface. Si no les decimos lo que necesitamos, entonces no podemos quejarnos. Esto lo plantearé la próxima vez que nos reunamos.”

¿Qué tan eficaz cree que será Merle para comunicar sus normas de calidad al equipo de análisis de sistemas y a los miembros del grupo de trabajo de SI? Responda en un párrafo. ¿Si los analistas de sistemas pueden percibir la renuencia de Merle para trabajar con el grupo de trabajo en el desarrollo de las normas de calidad, qué le diría para convencerlo de la importancia de la participación de los usuarios en la TQM? Haga una lista de argumentos que apoyen el uso de la TQM. ¿Cómo puede responder el equipo de análisis de sistemas a las inquietudes que plantea Portia? Explique su respuesta en un párrafo.

REPASO ESTRUCTURADO

Una de las acciones de administración de la calidad más eficaces que puede emprender el equipo de análisis de sistemas es hacer repasos estructurados de manera rutinaria. Los repasos estructurados son una forma de usar expertos para monitorear la programación y el desarrollo general del sistema, señalar los problemas y permitir al programador o analista responsable de dicha parte del sistema hacer los cambios correspondientes.

Los repasos estructurados involucran por lo menos a cuatro personas: la persona responsable de la parte del sistema o subsistema que se revisará (un programador o analista), un coordinador del repaso, un programador o analista experto y un experto que toma notas acerca de las sugerencias.

Cada persona que participa en un repaso tiene un papel especial que cumplir. El coordinador se encarga de asegurar que otros cumplan los papeles que se les asigne y de que se realicen las actividades establecidas. El programador o analista está para escuchar, no para defender su punto de vista, racionalizar o discutir un problema. El programador o analista experto tiene que señalar los errores o problemas potenciales, sin especificar cómo se deben resolver. El tomador de notas registra lo que se dice con el fin de que los demás participantes puedan interactuar sin ningún problema.

Los repasos estructurados se pueden hacer siempre que una parte de la codificación, de un subsistema o de un sistema esté terminada. Simplemente asegúrese de que el subsistema bajo revisión sea comprensible fuera del contexto al que pertenece. Los repasos estructurados son especialmente adecuados en un enfoque de administración de la calidad total cuando se realizan durante todo el ciclo de vida del desarrollo de sistemas. El tiempo para realizarlos debe ser de media hora a una hora cuando mucho, lo cual implica que deben coordinarse muy bien. En la figura 16.2 se muestra un formulario útil para organizar el repaso estructurado e informar sus resultados. Debido a que los repasos estructurados toman tiempo, no abuse de ellos.

**Informe para la gerencia
sobre el repaso estructurado**

Fecha del repaso: / /
Hora:

Nombre del proyecto:

Número del proyecto:

Parte (descripción) del trabajo examinado:

Coordinador del repaso:
Lista de participantes:

Comentarios:

Firma del coordinador:
Fecha en que se archiva
el informe:
/ /

Acción recomendada (marque una):
☐ ACEPTAR EL TRABAJO TAL Y COMO ESTÁ
☐ MODIFICAR EL TRABAJO
☐ MODIFICAR EL TRABAJO Y REALIZAR
UN REPASO DE SEGUIMIENTO
☐ RECHAZAR EL TRABAJO

FIGURA 16.2

Formulario para documentar repastos estructurados; los repastos se pueden hacer siempre que se finalice una parte de la codificación, de un sistema o de un subsistema.

Utilice los repastos estructurados para obtener (y después emprender acciones acordes con) retroalimentación valiosa desde una perspectiva que le falte. Al igual que con todas las medidas de aseguramiento de la calidad, el propósito de los repastos es evaluar el producto sistemáticamente de manera continua en lugar de esperar hasta la terminación del sistema.

DISEÑO Y DESARROLLO DE SISTEMAS

En esta sección definimos los diseños de sistemas ascendente (de abajo a arriba o *bottom-up*) y descendente (de arriba abajo o *top-down*), así como también el enfoque modular para la programación. Discutimos las ventajas de cada uno, así como también las precauciones que se deben observar al emplear un enfoque descendente o uno modular. También discutimos las propiedades de los enfoques descendente y modular para ayudar en el aseguramiento de la calidad de los proyectos de sistemas.

Diseño ascendente Este diseño se refiere a identificar los procesos que necesitan computarizarse conforme surgen, analizarlos como sistemas y codificar los procesos o comprar software empaquetado para resolver el problema inmediato. Los problemas que requieren computarizarse normalmente se encuentran en el nivel más bajo de la organización. Con frecuencia este tipo de problemas son estructurados y por lo tanto son más sensibles a la computarización; también son los más rentables. Por lo tanto, el nombre *ascendente* se refiere al nivel inferior en el cual se introduce primero la computarización. Por ejemplo, con frecuencia los negocios toman este enfoque para el desarrollo de sistemas al adquirir software comercial para la contabilidad, un paquete diferente para la programación de producción y otro para el marketing.

Cuando la programación interna se hace con un enfoque de ascendente, es difícil interconectar los subsistemas de manera que se desempeñen fácilmente como un sistema. Es muy costoso corregir las fallas de la interconexión y muchas de ellas no se descubren sino hasta que se completa la programación, cuando los analistas intentan reunir el sistema en la fecha límite señalada para la entrega. En esta situación, hay poco tiempo, presupuesto o paciencia del usuario para la depuración de interconexiones delicadas que se han ignorado.

Aunque cada subsistema aparenta conseguir lo que quiere, al considerar el sistema global hay serias limitantes para tomar un enfoque de ascendente. Una es que hay duplicidad de esfuerzo en comprar software e incluso en introducir los datos. Otra es que se introducen datos inválidos en el sistema. Una tercera, y quizás la desventaja más seria del enfoque ascendente, es que no se consideran los objetivos organizacionales globales, y por lo tanto dichos objetivos no se pueden cumplir.

Diseño descendente Es fácil visualizar este enfoque; como se muestra en la figura 16.3, significa ver una descripción amplia del sistema y después dividirla en partes más pequeñas o subsistemas. El diseño descendente permite a los analistas de sistemas determinar primero los objetivos organizacionales globales, así como también determinar cómo se reúnen mejor en un sistema global. Después el analista divide dicho sistema en subsistemas y sus requerimientos.

El diseño descendente es compatible con el pensamiento general de sistemas que se discutió en el capítulo 2. Cuando los analistas de sistemas utilizan un enfoque descendente,

FIGURA 16.3

Uso de un enfoque descendente para determinar primero los objetivos organizacionales generales.

NIVEL DE OBJETIVOS ORGANIZACIONALES

(coordinar los sistemas para conocer los objetivos de la compañía)

NIVEL DE SISTEMAS FUNCIONALES

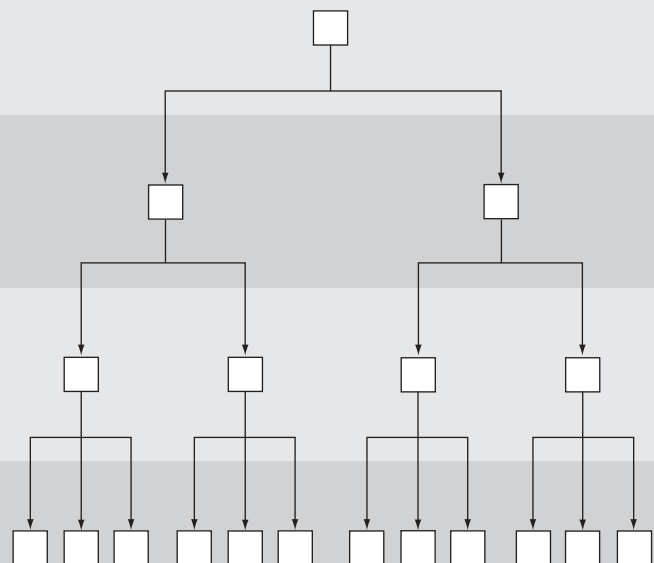
(por ejemplo, nómina, contabilidad y sistemas de producción)

NIVEL DE SISTEMAS OPERACIONALES

(por ejemplo, administración de edición, actualización e impresión)

NIVEL DE MÓDULO DE PROGRAMA

(por ejemplo, leer, clasificar, escribir en archivos e imprimir datos)



están pensando en la manera en que las interrelaciones e interdependencias de subsistemas se adaptan a la organización existente. El enfoque descendente también proporciona el énfasis deseable en la colaboración o interconexiones que los sistemas y sus subsistemas necesitan, el cual falta en el enfoque ascendente.

Las ventajas de usar un enfoque descendente para el diseño de sistemas incluyen evitar el caos de intentar diseñar un sistema de repente. Como hemos visto, planear e implementar sistemas de información de administración es increíblemente complejo. Intentar colocar todos los subsistemas en su lugar y ejecutarlos en seguida es casi un fracaso seguro.

Una segunda ventaja de tomar un enfoque descendente para diseñar es que permite separar a los equipos de análisis de sistemas para trabajar en paralelo en diferentes subsistemas, lo cual puede ahorrar mucho tiempo. El uso de equipos para el diseño de subsistemas se ajusta particularmente bien a un enfoque de control de calidad total.

Una tercera ventaja es que un enfoque descendente evita un problema mayor asociado con un enfoque ascendente; evita que los analistas de sistemas se metan tanto en los detalles que pierdan de vista lo que se supone que el sistema hace.

Hay algunas dificultades con el diseño descendente que el analista de sistemas necesita saber. La primera es el riesgo de que el sistema se divida en subsistemas “erróneos”. Se debe poner atención a las necesidades que se traslapen y a la compartición de recursos de manera que la partición en subsistemas tenga sentido para todos los sistemas. Además, es importante que cada subsistema solucione el problema correcto.

Un segundo riesgo es que una vez que se hacen las divisiones de un subsistema, sus interfaces se podrían descuidar o ignorar. Es necesario detallar de quién es la responsabilidad de las interfaces.

Una tercera advertencia que acompaña al uso de un diseño descendente es que los subsistemas se deben reintegrar eventualmente. Los mecanismos para la reintegración se necesitan poner en funcionamiento desde el principio. Una sugerencia es negociar información regular entre los equipos del subsistema; otra es usar herramientas que permiten flexibilidad si se requieren cambios para los subsistemas interrelacionados.

La administración de la calidad total y el enfoque descendente para diseñar pueden estar relacionados. El enfoque descendente proporciona el grupo de sistemas con una división más natural de usuarios en grupos de trabajo para subsistemas. Los grupos de trabajo establecidos de esta forma después pueden servir a una función dual como círculos de calidad para el sistema de información de administración. La estructura necesaria para el control de calidad está en el lugar, así como la motivación apropiada para obtener el subsistema para lograr las metas departamentales que son importante para los usuarios involucrados.

DESARROLLO MODULAR

Una vez que se toma el enfoque del diseño descendente, el enfoque modular es útil en la programación. Este enfoque implica dividir la programación en partes lógicas y manejables llamadas módulos. Este tipo de programación funciona bien con el diseño descendente porque da énfasis a las interfaces entre los módulos y no los descuida hasta el final del desarrollo de sistemas. Idealmente, cada módulo individual debe ser funcionalmente cohesivo de manera que se encargue de realizar una sola función.

El diseño de programa modular tiene tres ventajas principales. Primero, los módulos son más fáciles de escribir y de depurar porque prácticamente son independientes. Rastrear un error en un módulo es menos complicado, debido a que un problema en un módulo no debe causar problemas en otros.

Una segunda ventaja del diseño modular es que los módulos son más fáciles de mantener. Normalmente las modificaciones se limitarán a unos módulos y no seguirán en todo el programa.

Una tercer ventaja del diseño modular es que los módulos son más fáciles de entender, debido a que son subsistemas independientes. Por lo tanto, un lector puede adquirir una lista del código de un módulo y entender su función.

Algunos lineamientos para la programación modular incluyen lo siguiente:

1. Mantener cada módulo de un tamaño manejable (incluir a la perfección una sola función).
2. Poner particular atención a las interfaces críticas (los datos y variables de control que se pasan a otros módulos).
3. Minimizar el número de módulos que el usuario debe modificar al hacer los cambios.
4. Mantener las relaciones jerárquicas establecidas en las fases descendentes.

MODULARIDAD EN EL ENTORNO DE WINDOWS

La modularidad se está volviendo muy importante. Microsoft desarrolló dos sistemas para vincular los programas en su entorno de Windows. El primero se llama intercambio dinámico de datos (DDE), el cual comparte código al usar archivos de biblioteca de vínculos dinámicos (DLL). Al usar DDE, un usuario puede almacenar datos en un programa —quizás en una hoja de cálculo tal como Excel— y después usar dichos datos en otro programa, por decir, en un paquete de procesamiento de texto tal como Word para Windows. El programa que contiene los datos originales se denomina servidor y el programa que los usa se denomina cliente. El vínculo de DDE se puede establecer de manera que cuando se abra el archivo de procesamiento de texto del cliente, los datos se actualicen automáticamente y se reflejen los cambios hechos en el archivo de hoja de cálculo del servidor desde la última vez que se abrió dicho archivo de procesamiento de texto. (Véase el capítulo 17 para una discusión amplia del modelo cliente/servidor.)

Uno de los archivos de la DLL normalmente usados es COMMDLG.DLL, el cual contiene los cuadros de diálogo de Windows para **Abrir Archivos**, **Guardar Archivos**, **Buscar** e **Imprimir**. Una ventaja de usar este archivo es que los programas tendrán la misma apariencia y funcionamiento que otros programas de Windows. También acelera el desarrollo, debido a que los programadores no tienen que escribir el código contenido en los archivos DLL más comunes.

Un segundo enfoque para vincular programas en Windows se denomina vinculación e incrustación de objetos (OLE). Este método de vincular programas es superior a DDE porque está ligado a los datos y gráficos de la aplicación. Mientras que DDE utiliza un enfoque de cortar y pegar para vincular datos y no retiene el formato, OLE retiene todas las propiedades de los datos creados originalmente. Este enfoque orientado a objetos (véase el capítulo 18 para una discusión de principios orientados a objetos) permite al usuario final permanecer en la aplicación del cliente y editar los datos originales en la aplicación del servidor. Con OLE, cuando un usuario final hace clic en el objeto incrustado, se despliega una barra de herramientas que permite la edición visual.

USO DE DIAGRAMAS DE ESTRUCTURA PARA DISEÑAR SISTEMAS

La herramienta recomendada para diseñar un sistema modular descendente se denomina diagrama de estructura. Este gráfico simplemente es un diagrama que consiste de cuadros rectangulares, los cuales representan los módulos, y de flechas de conexión.

La figura 16.4 muestra tres módulos que se etiquetan como 000, 100 y 200 y se conectan mediante líneas de ángulo recto. Los módulos de nivel superior se numeran por 100s o 1,000s y los módulos de nivel inferior se numeran por 10s o 100s. Esta enumeración permite a programadores insertar módulos que usan un número entre los números de módulo adyacentes. Por ejemplo, un módulo insertado entre los módulos 110 y 120 recibiría el número 115. Si se insertaran dos módulos, los números podrían ser 114 y 117. Estos esquemas de numeración varían, dependiendo de los estándares organizacionales usados.

A los lados de las líneas de conexión, se dibujan dos tipos de flechas. Las flechas con los círculos vacíos se denominan parejas de datos y las flechas con los círculos rellenos se denominan banderas de control o interruptores. Un interruptor es lo mismo que una bandera de control excepto por que está limitado por dos valores: sí o no. Estas flechas indican que algo se pasa hacia abajo al módulo inferior o hacia arriba al superior.

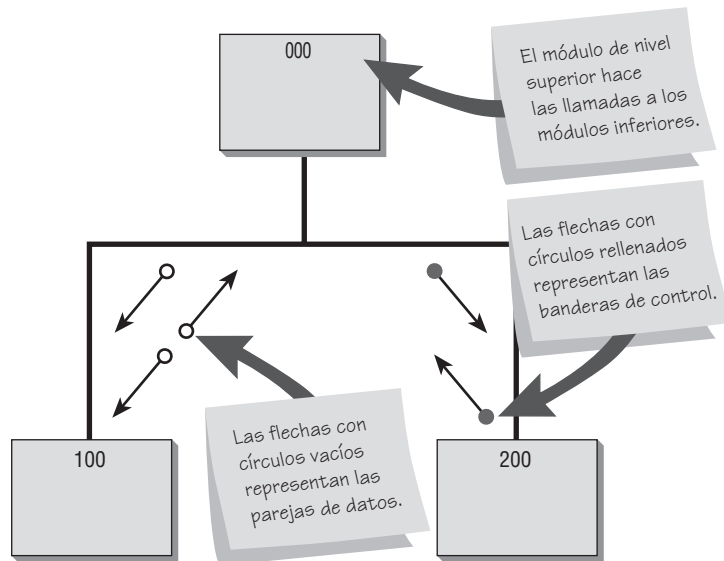


FIGURA 16.4

Un diagrama de estructura favorece el diseño descendente mediante módulos.

El analista debe mantener a la perfección este acoplamiento al mínimo. Cuando hay pocas parejas de datos y banderas de control en el sistema, lo más fácil es cambiar el sistema. Cuando finalmente se programan estos módulos, es importante pasar el menor número de parejas de datos entre los módulos.

Aún más importante es que se deben evitar las banderas de control numerosas. El control se diseña para ser pasado de los módulos de nivel inferior a los de nivel superior en la estructura. Sin embargo, en raras ocasiones será necesario pasar el control hacia abajo en la estructura. Las banderas de control deciden qué parte de un módulo se ejecuta y están asociadas con las instrucciones IF...THEN...ELSE... y otros tipos similares de instrucciones. Cuando el control se pasa en forma descendente, se permite que un módulo de nivel inferior tome una decisión y el resultado es un módulo que desempeña dos tareas diferentes. Este resultado rompe con el modelo de un módulo funcional: un módulo sólo debe desempeñar una tarea.

La figura 16.5 ilustra una parte de un diagrama de estructura para agregar nuevos empleados. El programa lee un archivo de TRANSACCIÓN DE EMPLEADO y verifica que cada registro en el archivo únicamente contenga datos aceptables. Los informes se imprimen por separado para los registros válidos e inválidos, proporcionando un rastro para auditoría de todas las transacciones. El informe que contiene los registros inválidos se envía al

FIGURA 16.5

Este diagrama de estructura muestra el control que se mueve de forma descendente y también muestra los módulos no funcionales.

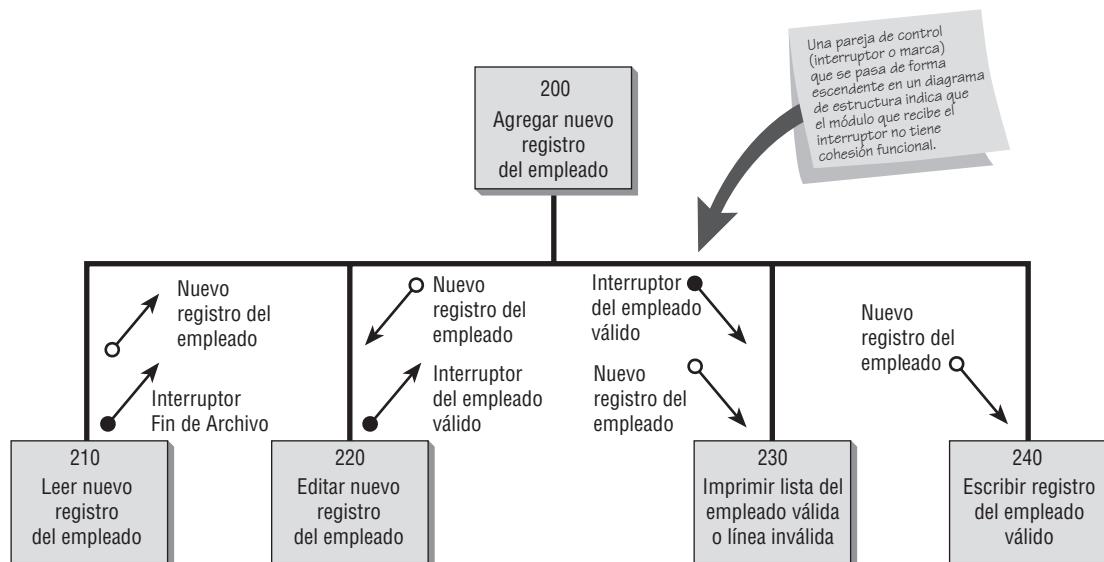


FIGURA 16.6

El pseudocódigo para el módulo 230 ilustra el efecto de pasar de forma descendente un interruptor.

```

Módulo 230—Imprimir lista del empleado válida o línea inválida
IF INTERRUPTOR DEL EMPLEADO VÁLIDO = 'Y'
  Mover NÚMERO DEL EMPLEADO a LISTA DEL EMPLEADO VÁLIDA
  Mover NOMBRE DEL EMPLEADO a LISTA DEL EMPLEADO VÁLIDA
  Mover TRABAJO DEL EMPLEADO a LISTA DEL EMPLEADO VÁLIDA
  Mover DEPARTAMENTO DEL EMPLEADO a LISTA DEL EMPLEADO VÁLIDA
  Mover NÚMERO TELEFÓNICO DEL TRABAJO DEL EMPLEADO a LISTA DEL
  EMPLEADO VÁLIDA
  Mover FECHA DE CONTRATO DEL EMPLEADO a LISTA DEL EMPLEADO VÁLIDA
  DO IMPRIMIR LISTA DEL EMPLEADO VÁLIDA
ELSE
  Mover NÚMERO DEL EMPLEADO a LÍNEA INVÁLIDA
  Mover NOMBRE DEL EMPLEADO a LÍNEA INVÁLIDA
  Mover TRABAJO DEL EMPLEADO a LÍNEA INVÁLIDA
  Mover DEPARTAMENTO DEL EMPLEADO a LÍNEA INVÁLIDA
  Mover NÚMERO TELEFÓNICO DEL TRABAJO DEL EMPLEADO a LÍNEA INVÁLIDA
  Mover FECHA DE CONTRATO DEL EMPLEADO a LÍNEA INVÁLIDA
  DO IMPRIMIR LÍNEA INVÁLIDA
ENDIF

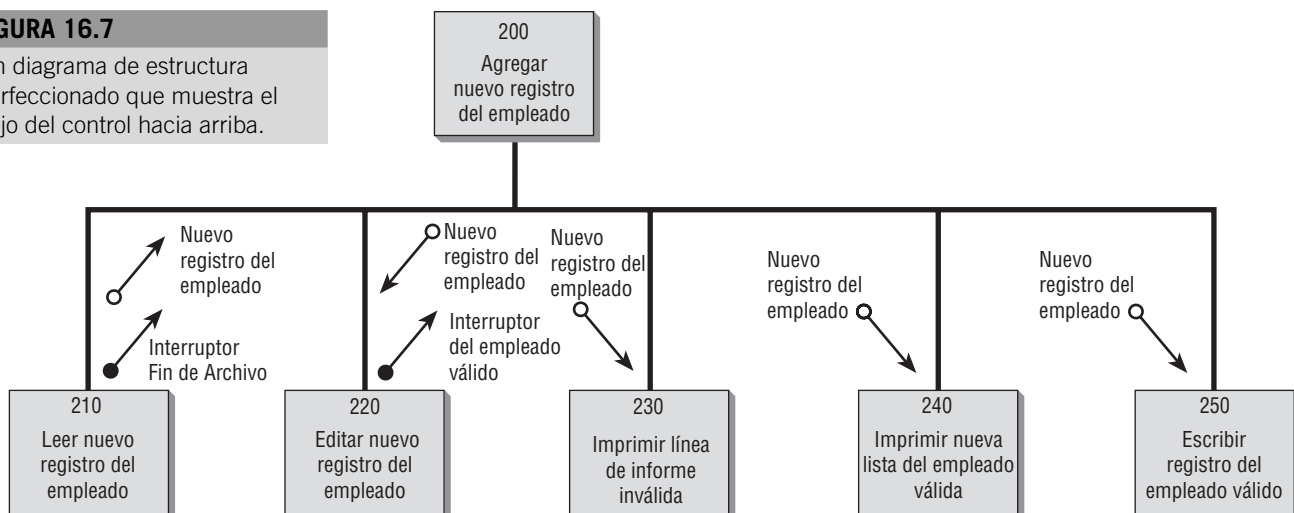
```

usuario para la corrección de errores. Los registros que son válidos se ponen en un archivo de transacción válido, el cual se pasa a un programa separado para actualizar el archivo MAESTRO DE EMPLEADOS. El módulo 200, AGREGAR NUEVO REGISTRO DEL EMPLEADO, representa la lógica de agregar un registro. Debido a que el módulo 230 se usa para imprimir ambos informes, se debe enviar una bandera de control hacia abajo para indicar al módulo que informe imprimir. De esta manera la lógica del módulo 230 se controla por completo mediante una instrucción IF, la cual se ilustra en la figura 16.6.

La figura 16.7 muestra la forma correcta de diseñar la estructura por debajo del módulo 200, AGREGAR NUEVO REGISTRO DEL EMPLEADO. Aquí, cada función de impre-

FIGURA 16.7

Un diagrama de estructura perfeccionado que muestra el flujo del control hacia arriba.



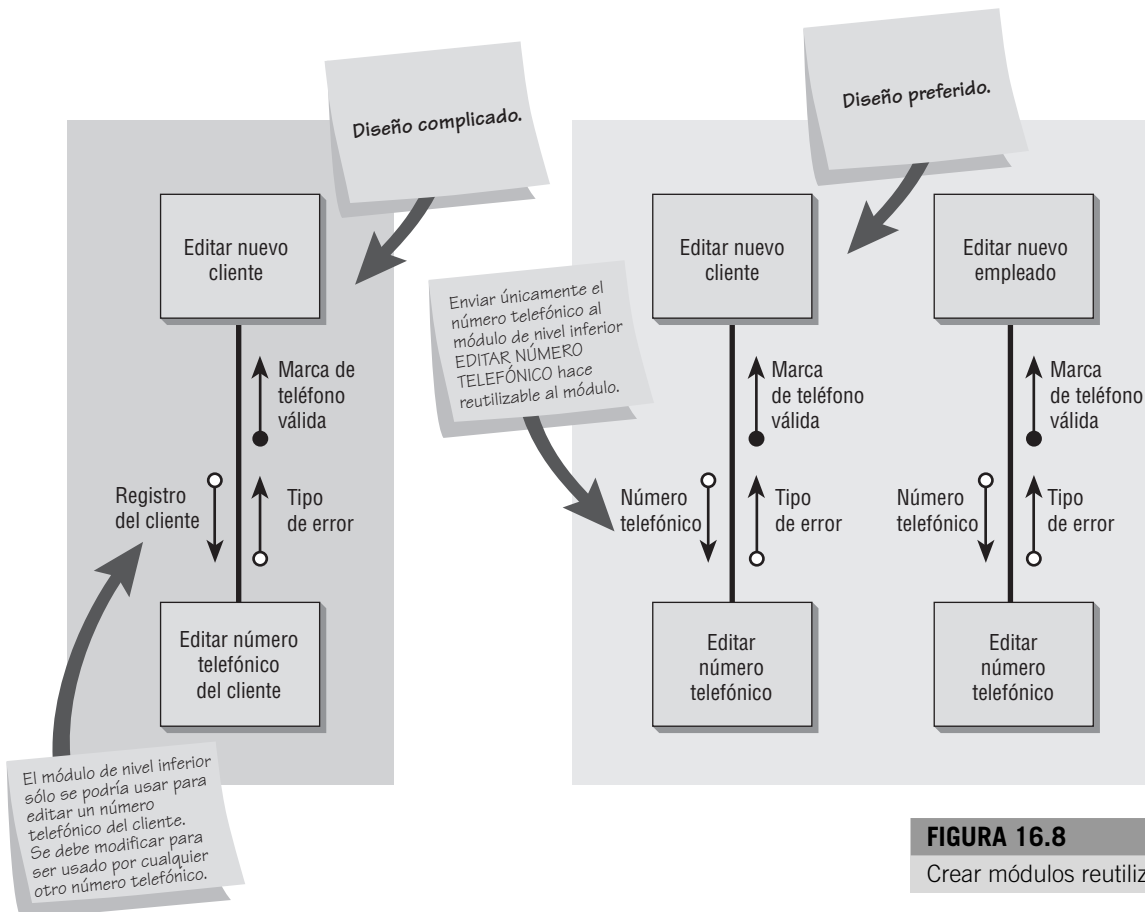


FIGURA 16.8
Crear módulos reutilizables.

sión se ha puesto en un módulo separado y las banderas de control sólo se pasan a la estructura al módulo de nivel superior.

También se deben examinar los datos que se pasan a través de las parejas de datos. Es mejor pasar sólo los datos requeridos para realizar la función del módulo. Este enfoque se denomina acoplamiento de datos. El paso excesivo de datos se denomina acoplamiento de sello, y aunque es relativamente inofensivo, reduce la posibilidad de crear un módulo reutilizable.

La figura 16.8 ilustra este concepto. Aquí, el módulo EDITAR NUEVO CLIENTE pasa el REGISTRO DEL CLIENTE al módulo EDITAR NÚMERO TELEFÓNICO DEL CLIENTE, donde NÚMERO TELEFÓNICO, un elemento encontrado en el REGISTRO DEL CLIENTE, se valida, y una bandera de control se pasa atrás al módulo EDITAR NUEVO CLIENTE. El TIPO DE ERROR (si hay alguno), uno que contiene un mensaje de error tal como "CÓDIGO DE ÁREA INVÁLIDO" o el "NÚMERO TELEFÓNICO NO ES NUMÉRICO", también se pasa hacia arriba. El mensaje se podría imprimir o desplegar en pantalla.

Aunque dichos módulos son bastante fáciles de crear y modificar cada vez que es necesario editar un número telefónico de un registro fuente diferente, se debe crear un nuevo módulo, similar al EDITAR NÚMERO TELEFÓNICO DEL CLIENTE. Además, si la forma del número telefónico está validando los cambios, como ocurre cuando se debe agregar un nuevo código de área o un código de país internacional, cada uno de estos módulos de nivel inferior se debe modificar.

Debido a que el módulo de nivel inferior no requiere ninguno de los otros elementos en el REGISTRO DEL CLIENTE, la solución es pasar sólo el NÚMERO TELEFÓNICO al módulo de nivel inferior. El nombre del módulo en este escenario cambia a EDITAR NÚ-

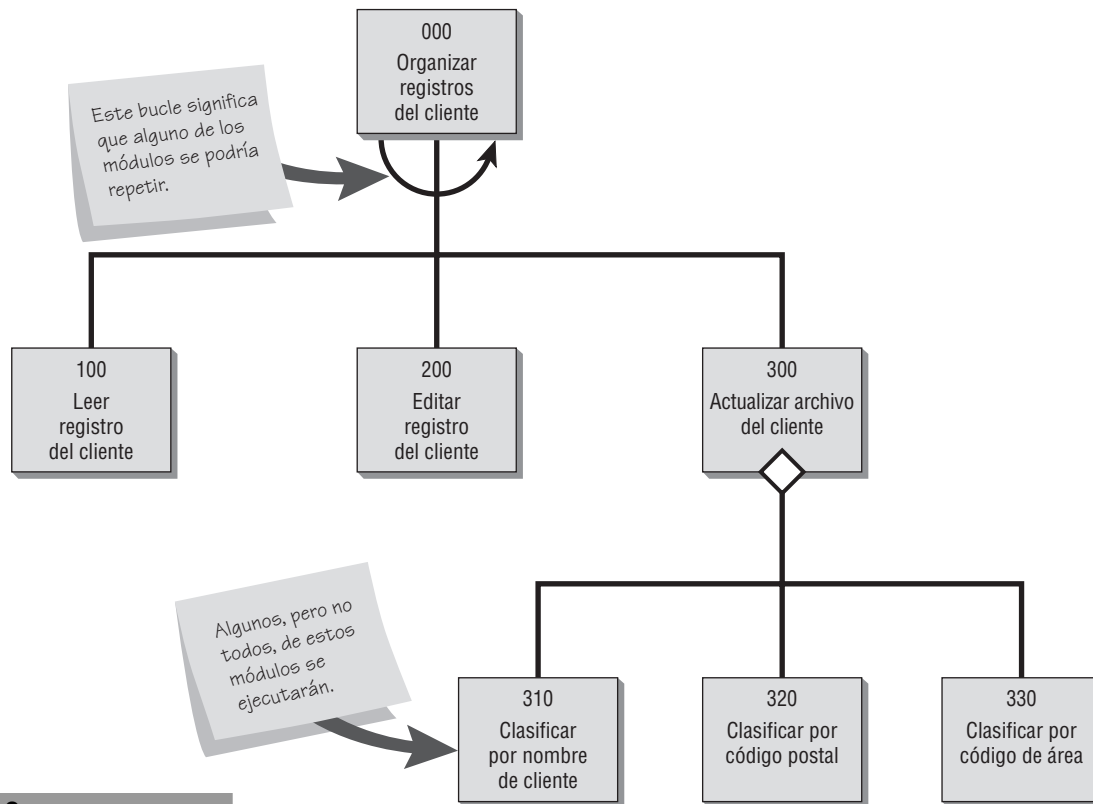


FIGURA 16.9

El bucle y el diamante son dos símbolos que indican una acción especial en un diagrama de estructura.

MERO TELEFÓNICO y se podría usar para editar cualquier número telefónico: un número telefónico del cliente o un número telefónico del empleado. Los módulos del lado derecho de la figura ilustran este concepto. Cuando cambian las reglas para validar el número telefónico, sólo se necesita modificar EDITAR NÚMERO TELEFÓNICO, sin tener en cuenta cuántos programas utilizan dicho módulo. Con frecuencia estos módulos de uso general se ponen en un programa compilado por separado denominado subprograma, función o procedimiento, dependiendo del lenguaje de programación usado.

Como se muestra en la figura 16.9, el bucle es otro símbolo usado en los diagramas de estructura. Este símbolo indica que algunos procedimientos encontrados en los módulos 100 y 200 serán repetidos hasta terminar. Este ejemplo implica que LEER REGISTRO DEL CLIENTE y EDITAR REGISTRO DEL CLIENTE se repitan hasta que todos los registros del cliente se completen. Después se clasifican en el módulo 300, pero como puede ver, se pueden clasificar de tres formas diferentes: por nombre, código postal o código de área. Para mostrar que parte, pero no toda, de la clasificación se realizará, se usa otro símbolo, un diamante. Observe que el diamante no indica cuál de los tres módulos se desempeñará, ni el bucle indica qué módulos se repetirán. Estos símbolos pretenden deliberadamente ser generales, no específicos.

DIBUJO DE UN DIAGRAMA DE ESTRUCTURA

Obviamente, los diagramas de estructura se deben dibujar de arriba hacia abajo, pero ¿dónde se empiezan a buscar los procesos que serán los módulos? Probablemente, el mejor lugar para buscar esta información es en el diagrama de flujo de datos (véase el capítulo 7).

Al transformar un diagrama de flujo de datos en un diagrama de estructura, se deben tener en cuenta varias consideraciones adicionales. El diagrama de flujo de datos indicará la secuencia de los módulos en un diagrama de estructura. Si un proceso proporciona en-

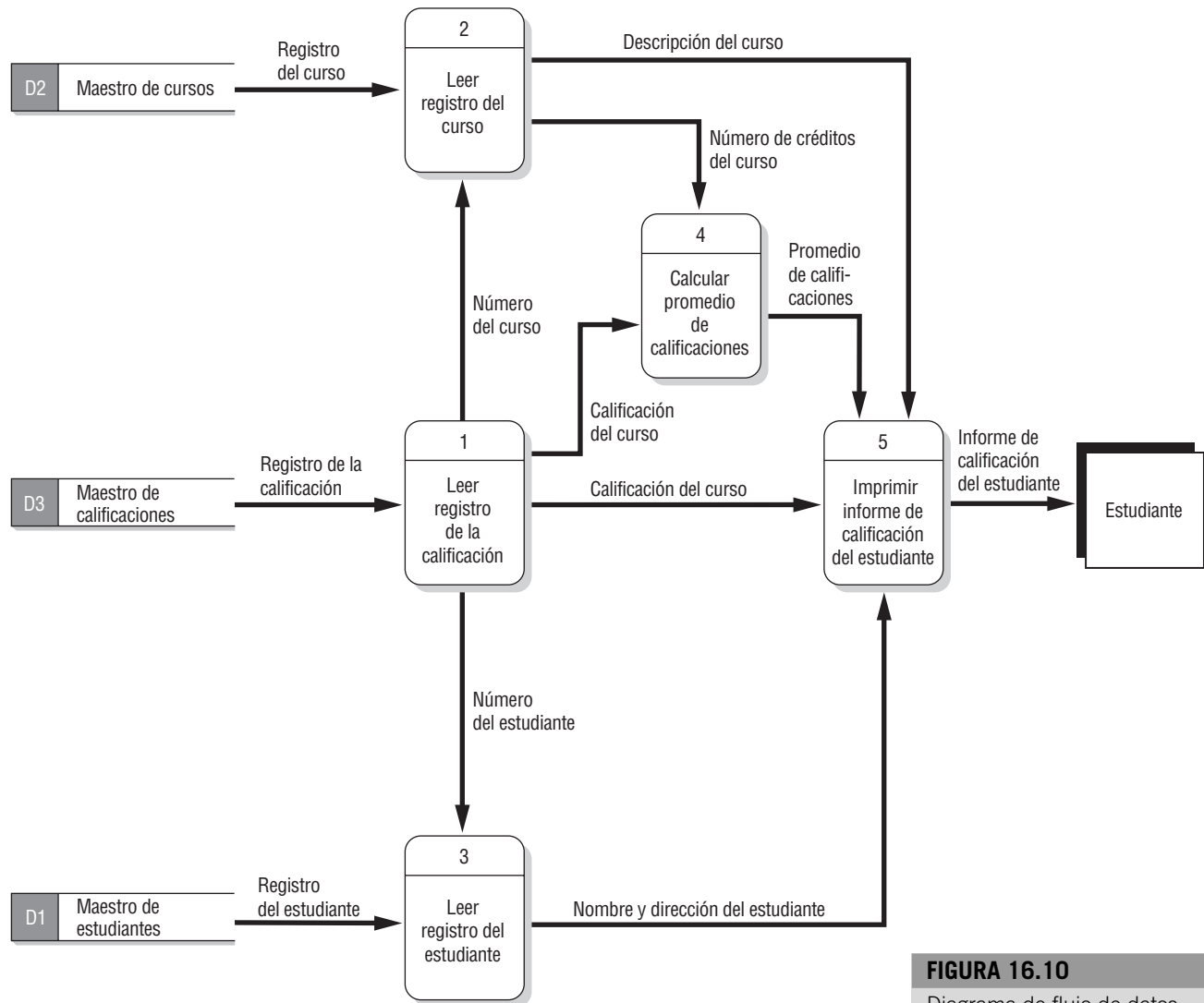


FIGURA 16.10

Diagrama de flujo de datos para imprimir un informe de calificación del estudiante.

trada a otro proceso, los módulos correspondientes se deben desempeñar en la misma secuencia. La figura 16.10 es un diagrama de flujo de datos para preparar un informe de calificación del estudiante. Observe que el proceso 1, LEER REGISTRO DE CALIFICACIÓN, proporciona entrada para el proceso 2, LEER REGISTRO DEL CURSO, y para el proceso 3, LEER REGISTRO DEL ESTUDIANTE. En la figura 16.11 se ilustra el diagrama de estructura creado para este diagrama. Observe que el módulo 110, LEER REGISTRO DE CALIFICACIÓN, se debe ejecutar primero. Después se deben ejecutar los procesos 2 y 3, pero debido a que no proporcionan entrada para otros, el orden de estos módulos (120 y 130) en el diagrama de estructura no es importante y se podría invertir sin afectar los resultados finales. Los procesos 1 y 2 proporcionan entrada para el proceso 4, CALCULAR MEDIA DE PUNTO DE CALIDAD (también conocido como módulo 140). El proceso 5, IMPRIMIR INFORME DE CALIFICACIÓN DEL ESTUDIANTE (módulo 150), recibe flujo de datos de todos los demás procesos y debe ser el último módulo en ser ejecutado.

Si un proceso se divide en un diagrama de flujo de datos hijo, el módulo correspondiente para el proceso padre tendrá módulos subordinados que correspondan a los procesos encontrados en el diagrama hijo. El proceso 5, IMPRIMIR INFORME DE CALIFICACIÓN DEL ESTUDIANTE, tiene cuatro flujos de datos de entrada y uno de salida y por ello es buen candidato para un diagrama hijo. En la figura 16.12 se ilustra el diagrama 5, los deta-

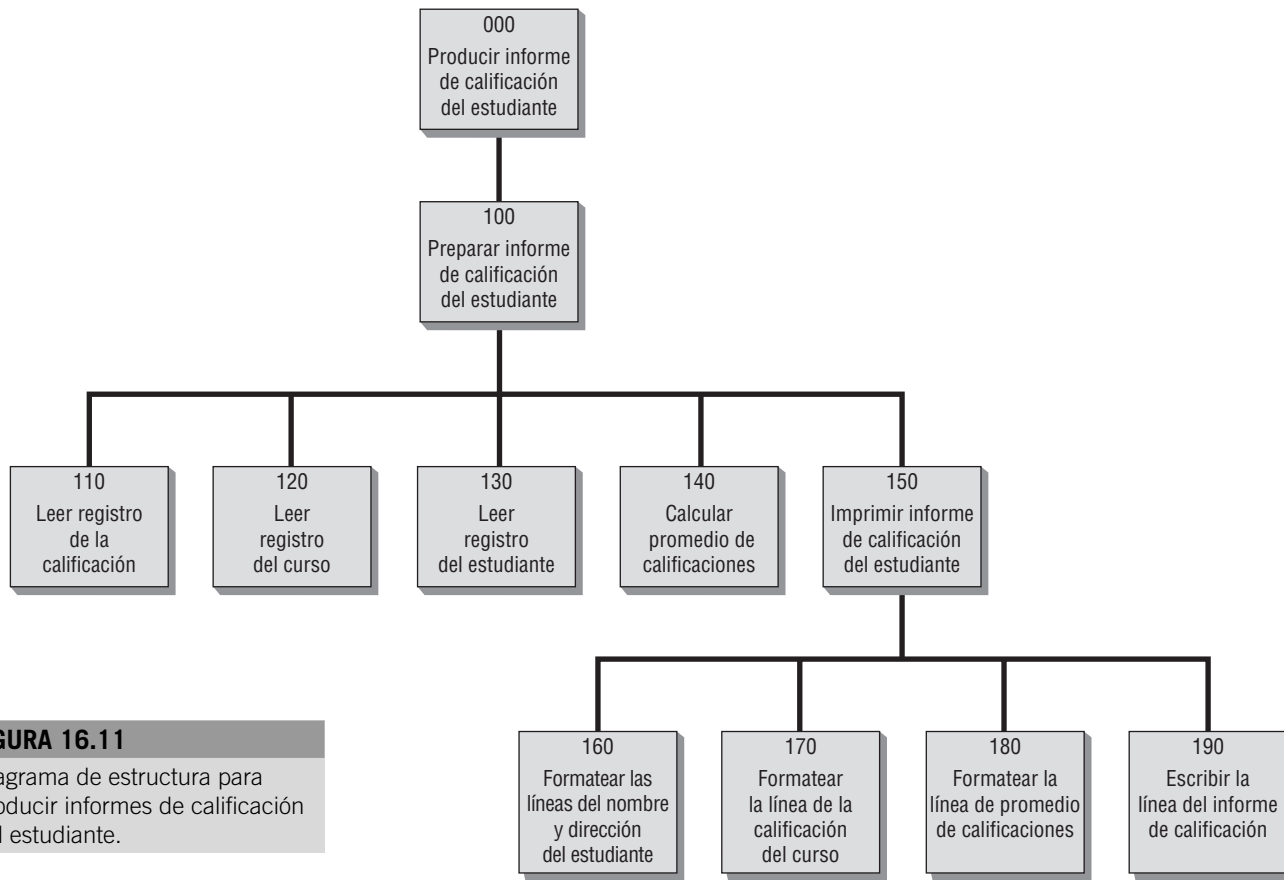


FIGURA 16.11

Diagrama de estructura para producir informes de calificación del estudiante.

lles del proceso 5. Los procesos en el diagrama 5 traducen los módulos subordinados al módulo 150, IMPRIMIR INFORME DE CALIFICACIÓN DEL ESTUDIANTE.

TIPOS DE MÓDULOS

Los módulos del diagrama de estructura entran en una de las tres categorías generales: (1) control, (2) transformacional (a veces denominado trabajador) o (3) funcional. Al producir un diagrama de estructura que es fácil de desarrollar y modificar, se debe tener cuidado de no mezclar los diferentes tipos de módulos.

Los módulos de control normalmente se encuentran cerca de la parte superior del diagrama de estructura y contienen la lógica para desempeñar los módulos de nivel inferior. Los módulos de control podrían estar, o no estar, representados en el diagrama de flujo de datos. Los tipos de instrucciones que normalmente están en los módulos de control son IF, PERFORM y DO. Las instrucciones detalladas tal como ADD y MOVE normalmente se mantienen al mínimo. Con frecuencia la lógica de control es la más difícil de diseñar; por lo tanto, los módulos de control no deben ser muy grandes. Si un módulo de control tiene más de nueve módulos subordinados, se deben crear nuevos módulos de control que sean subordinados del módulo de control original. La lógica de un módulo de control se podría determinar desde un árbol de decisión o una tabla de decisión. Una tabla de decisión con demasiadas reglas se debe dividir en varias tablas de decisión, con la primera tabla llamando a ejecución a la segunda tabla. Cada tabla de decisión produciría un módulo de control. (Véase el capítulo 9 para más información de los árboles y tablas de decisión.)

Los módulos transformacionales son aquellos creados de un diagrama de flujo de datos. Normalmente desempeñan una sola tarea, aunque varias tareas secundarias se podrían asociar con la principal. Por ejemplo, un módulo denominado IMPRIMIR LÍNEA TOTAL

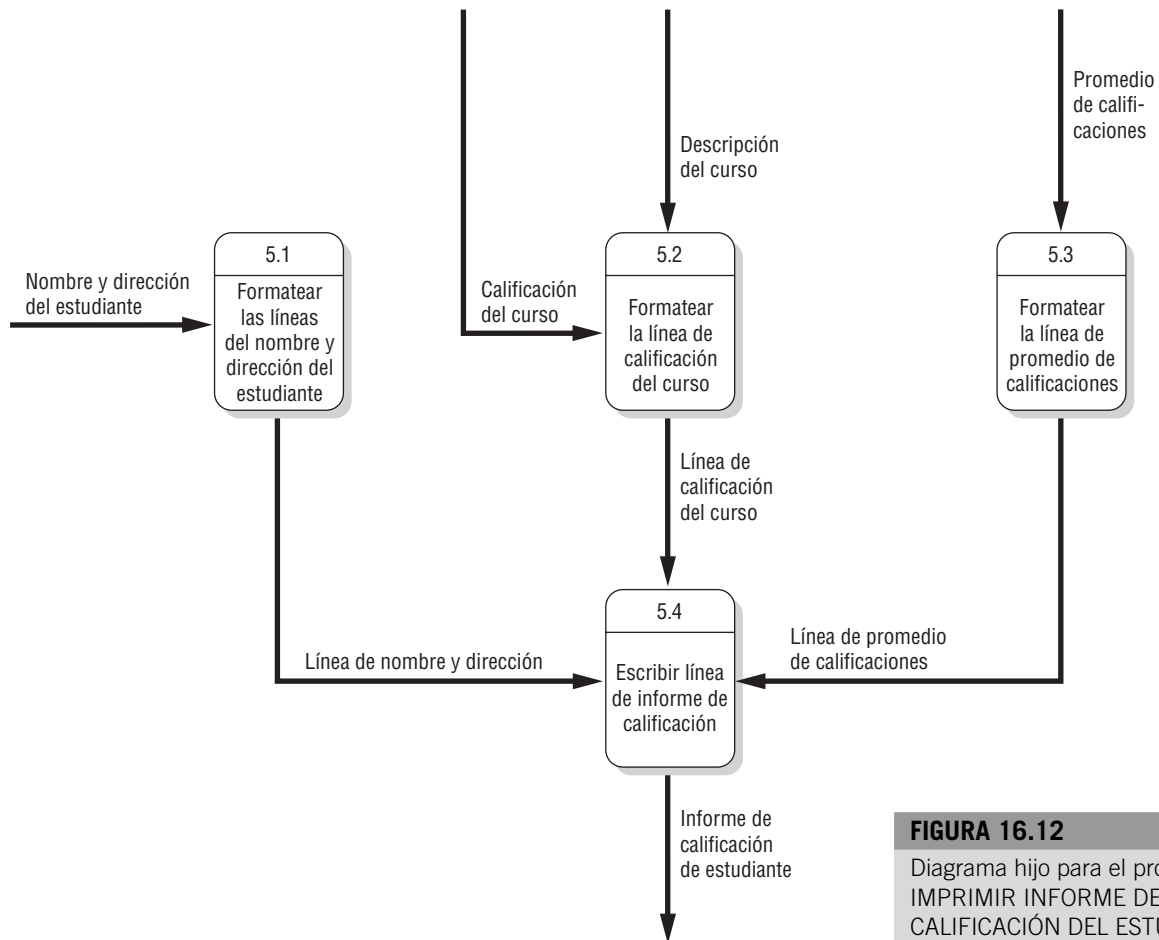


FIGURA 16.12

Diagrama hijo para el proceso 5, IMPRIMIR INFORME DE CALIFICACIÓN DEL ESTUDIANTE.

DEL CLIENTE podría formatear toda la línea, imprimir la línea, agregarla a los totales finales y establecer los totales del cliente a cero en la preparación para acumular las cantidades del siguiente cliente. Los módulos transformacionales normalmente incluyen una mezcla de instrucciones, unas cuantas instrucciones IF y PERFORM o DO y muchas instrucciones detalladas tales como MOVE y ADD. Estos módulos son inferiores en la estructura que los módulos de control.

Los módulos funcionales son los más bajos en la estructura, rara vez tienen un módulo subordinado bajo ellos. Sólo desempeñan una tarea, tal como formatear, leer, calcular o escribir. Algunos de estos módulos se encuentran en un diagrama de flujo de datos, pero otros se tendrían que agregar, tal como leer un registro o imprimir una línea de error.

La figura 16.13 representa el diagrama de estructura para agregar las reservaciones para los huéspedes de un hotel. Los módulos 000, AGREGAR RESERVACIÓN DEL HUÉSPEDE y 100, AGREGAR RESERVACIÓN DEL CUARTO, son módulos de control que representan el programa entero (módulo 000) y proporcionan el control necesario para hacer una reservación de cuarto (módulo 100). El módulo 110, DESPLEGAR PANTALLA DE RESERVACIÓN, es un módulo funcional responsable de mostrar la pantalla de reservación inicial. Los módulos 120, OBTENER RESERVACIÓN DE CUARTO VÁLIDA, y 160, CONFIRMAR RESERVACIÓN DEL CUARTO, son los módulos de control de nivel inferior.

El módulo 120, OBTENER RESERVACIÓN DE CUARTO VÁLIDA, se desempeña repetidamente hasta que los datos de la reservación sean válidos o hasta que el operador de la reservación cancele la transacción. Este tipo de módulo OBTENER RESERVACIÓN... desahoga el módulo 100 de una cantidad considerable de código complejo. Los módulos subordinados para OBTENER RESERVACIÓN DE CUARTO VÁLIDA son módulos funcio-

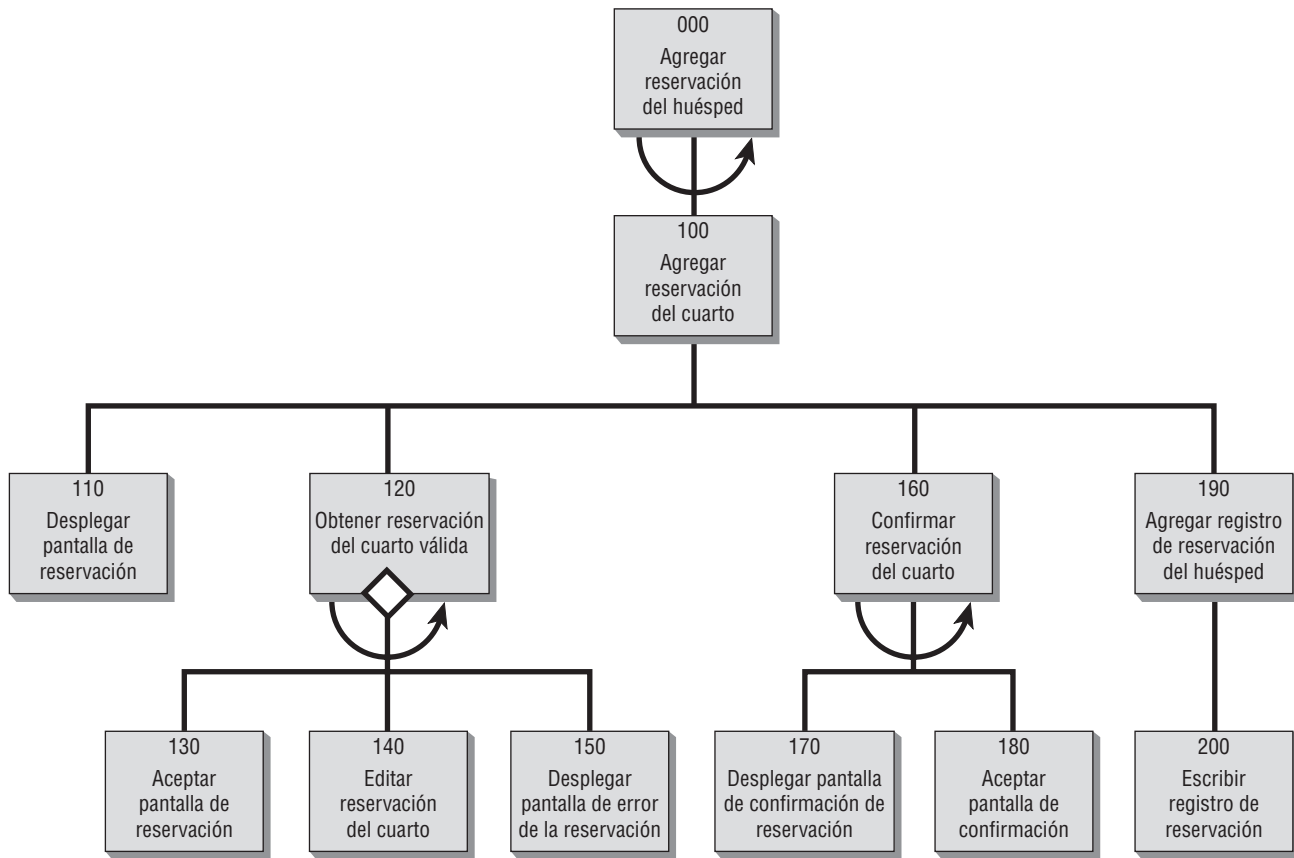


FIGURA 16.13

Diagrama de estructura para agregar, en línea, las reservaciones del huésped de un hotel.

nales responsables de recibir la pantalla de reservación, editar o validar la reservación del cuarto y mostrar una pantalla de error si los datos de entrada no son válidos. Debido a que estos módulos están en un bucle, el control permanece en esta parte de la estructura hasta que los datos de la pantalla sean válidos.

El módulo 160, CONFIRMAR RESERVACIÓN DEL CUARTO, también se desempeña repetidamente y permite al operador verificar que se haya introducido la información correcta. En esta situación, el operador inspeccionará la pantalla y presionará una tecla especificada, tal como **Enter**, si los datos son correctos, o una tecla diferente para modificar o cancelar la transacción. De nuevo, el programa permanecerá en estos módulos, repitiéndose hasta que el operador acepte o cancele la reservación.

El módulo 190 es un módulo transformacional que formatea el REGISTRO DE RESERVACIÓN y desempeña el módulo 200 para escribir el REGISTRO DE RESERVACIÓN. Los módulos 130, 140, 150, 170, 180 y 200 son módulos funcionales que desempeñan una sola tarea: aceptar una pantalla, desplegar una pantalla o editar o escribir un registro. Estos módulos son los más fáciles de codificar, depurar y mantener.

SUBORDINACIÓN DE MÓDULO

Un módulo subordinado es uno inferior en el diagrama de estructura llamado por otro módulo superior en la estructura. Cada módulo subordinado debe representar una tarea que es una parte de la función del módulo de nivel superior. Permitir que el módulo de nivel inferior desempeñe una tarea que no es requerida por el módulo que lo llama se denomina subordinación inadecuada. En tal caso, el módulo inferior se debe mover al nivel superior de la estructura.

La figura 16.14 ilustra este concepto mediante un diagrama de estructura para cambiar un archivo MAESTRO DE CLIENTES. Examine el módulo 120, LEER MAESTRO DE CLIENTES. Éste tiene la tarea de usar el NÚMERO DEL CLIENTE de CAMBIAR RE-

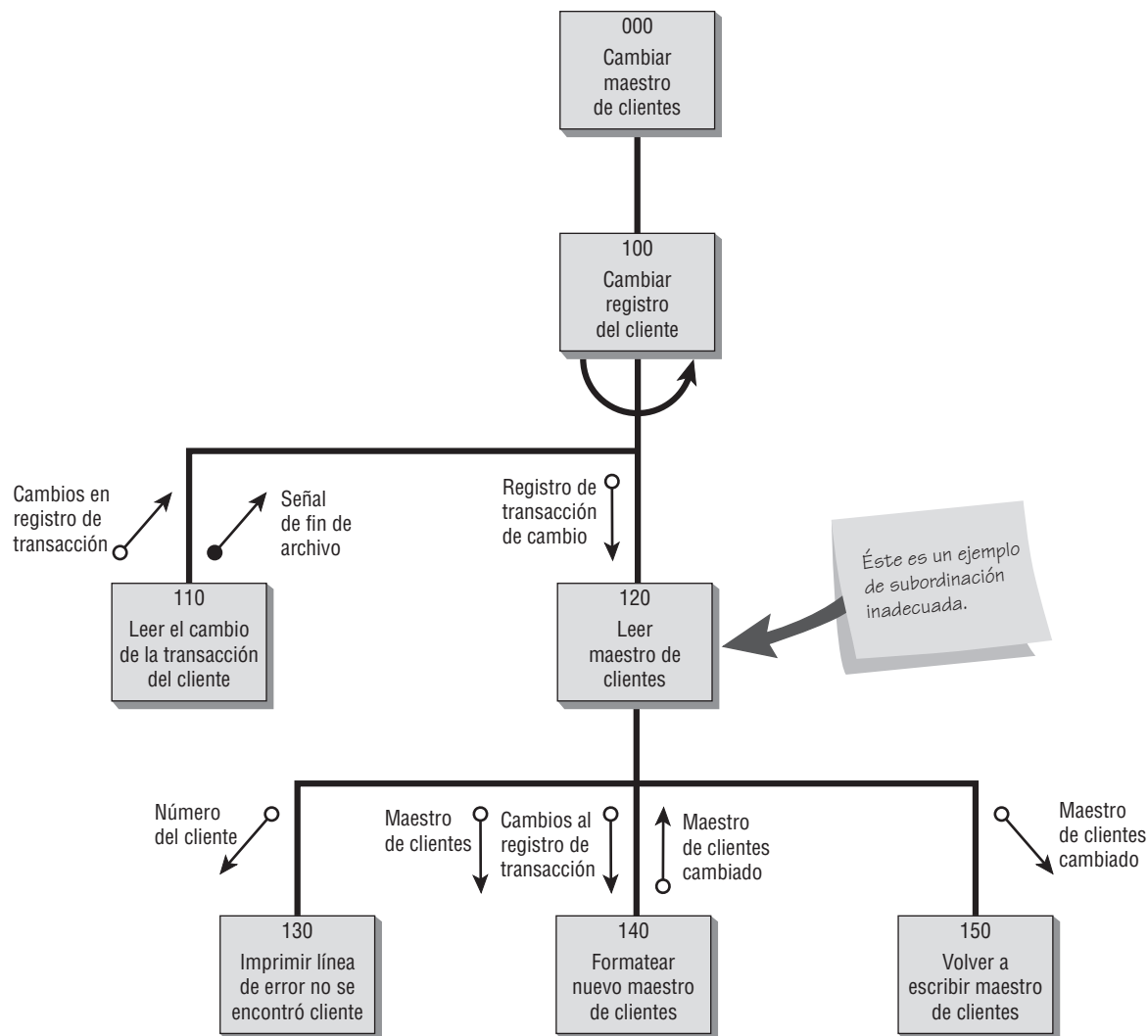


FIGURA 16.14

Diagrama de estructura que ilustra una subordinación inadecuada.

GISTRO DE TRANSACCIÓN para obtener directamente la coincidencia del REGISTRO DEL CLIENTE. Si no se encuentra el registro, se imprime una línea de error. Por otra parte, el MAESTRO DE CLIENTES se cambia y se vuelve a escribir el registro. Este módulo debe ser un módulo funcional, que simplemente lee un registro, pero en cambio tiene tres módulos subordinados. La pregunta debe ser, “¿Se debe imprimir una línea de error para lograr la lectura del MAESTRO DE CLIENTES?” Además, “¿Se debe formatear y volver a escribir el MAESTRO DE NUEVOS CLIENTES para leer el MAESTRO DE CLIENTES?” Debido a que la respuesta es no para ambas preguntas, los módulos 130, 140 y 150 no deben ser subordinados de LEER MAESTRO DE CLIENTES.

La figura 16.15 muestra el diagrama de estructura modificado. Las instrucciones de control se sacaron del registro LEER MAESTRO DE CLIENTES y se pusieron en el módulo de control principal, CAMBIAR REGISTRO DEL CLIENTE. LEER MAESTRO DE CLIENTES se vuelve un módulo funcional (módulo 120).

Aun cuando un diagrama de estructura logra todos los propósitos para los cuales fue diseñado, no puede ser la única técnica usada para diseño y documentación. Primero, no muestra el orden en que se deben ejecutar los módulos (un diagrama de flujo de datos lo hará). Segundo, no muestra suficientes detalles (un pseudocódigo lo hará). El resto de este capítulo discute estas técnicas más detalladas usando como ejemplo el problema de suscripción a un periódico presentado anteriormente, el cual se describe ahora con más detalle.

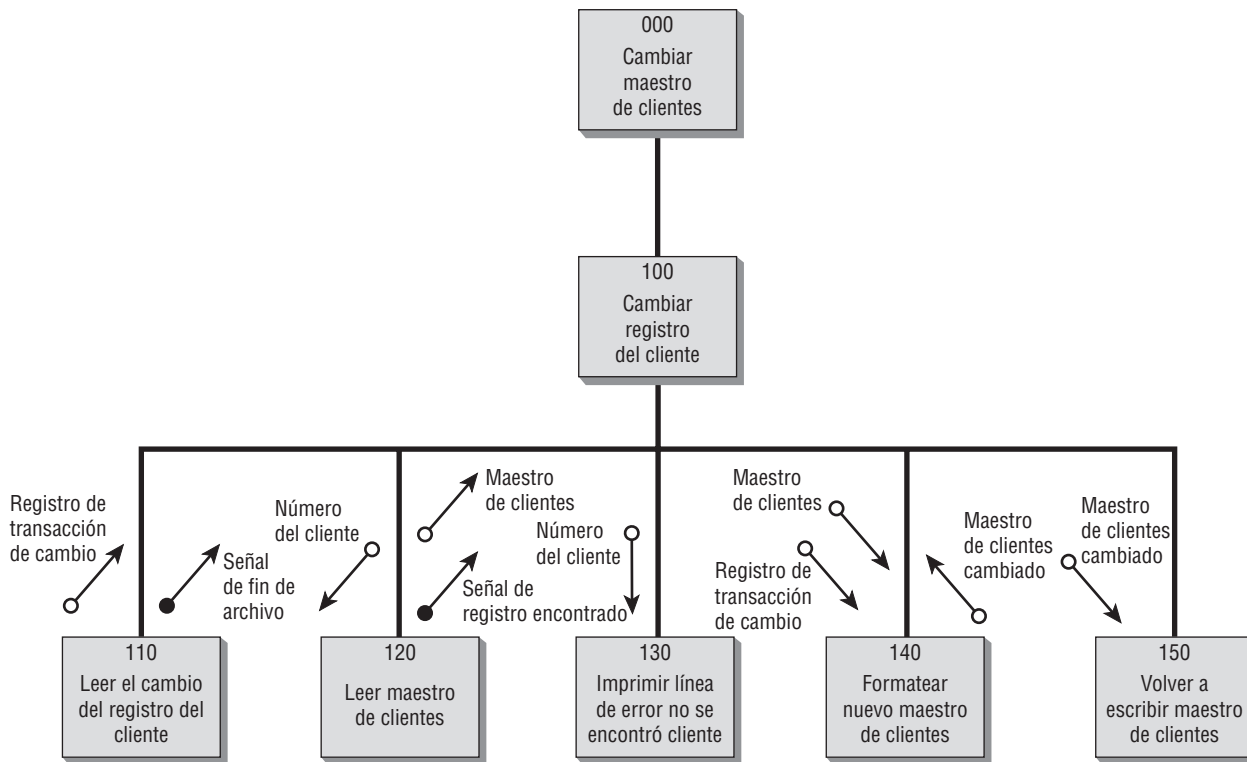


FIGURA 16.15

Diagrama de estructura modificado que muestra la subordinación adecuada.

INGENIERÍA DE SOFTWARE Y DOCUMENTACIÓN

La planeación y control son elementos fundamentales en todo sistema exitoso. En el desarrollo de software para el sistema, el analista de sistemas debe saber que la planeación tiene lugar en el diseño, incluso antes de que empiece la programación. Necesitamos técnicas que nos ayuden a establecer los objetivos del programa, de manera que nuestros programas estén completos. También necesitamos técnicas de diseño que nos ayuden a separar el esfuerzo de programación en módulos manejables.

Sin embargo, no es satisfactorio intentar tener éxito tan sólo con las etapas de la planeación. Después de que se completan los programas, se deben mantener y los esfuerzos de mantenimiento normalmente son mayores que el esfuerzo empleado en el diseño y la programación originales.

Las técnicas descritas en la siguiente sección no sólo están hechas para usarse inicialmente en el diseño de software, sino también en su mantenimiento. Debido a que la mayoría de los sistemas no se consideran desechables, muy probablemente necesitarán ser mantenidos. El esfuerzo de aseguramiento de la calidad total requiere que los programas se documenten adecuadamente.

El software y los procedimientos se documentan de manera que se codifiquen en un formato que se pueda acceder fácilmente. El acceso a esta documentación es necesario para las nuevas personas que aprenden el sistema y como un recordatorio para aquellos que no usan el programa con frecuencia. La documentación permite a usuarios, programadores y analistas “ver” el sistema, su software y procedimientos sin tener que interactuar con él.

Cierta documentación proporciona una apreciación global del propio sistema, mientras que la documentación de procedimiento detalla lo que se debe hacer para ejecutar el software en el sistema y la documentación del programa detalla el código del programa que se usa.

La rotación del personal de servicio de información tradicionalmente ha sido alta en comparación con otros departamentos, de manera que probablemente las personas que concibieron e instalaron el sistema original no serán las mismas que las que lo mantienen.

La documentación consistente y bien actualizada acortará el número de horas requerido para que las nuevas personas aprendan el sistema antes de realizar el mantenimiento.

Hay muchas razones por las cuales los sistemas y programas no están documentados o presentan subdocumentación. Algunos de los problemas residen en los sistemas y programas, otros en los analistas de sistemas y programadores.

Algunos sistemas heredados fueron escritos antes de que un negocio estandarizara sus técnicas de documentación, pero todavía se usan (sin documentación). Muchos otros sistemas han sufrido modificaciones mayores o menores y se han actualizado durante los años, pero su documentación no se ha modificado para reflejar esto. Incluso se puede dar el caso que se hayan comprado algunos sistemas especializados para aplicaciones importantes a pesar de su falta de documentación.

Los analistas de sistemas podrían no documentar adecuadamente los sistemas debido a que no tienen tiempo o no se les paga por el tiempo usado en la documentación. Algunos analistas no documentan porque tienen miedo de hacerlo o piensan que no es su trabajo. Además, muchos analistas son reservados sobre documentar sistemas que no son suyos, quizás temen a las represalias si incluyen material incorrecto en el sistema de alguien más. La documentación lograda por medio de una herramienta CASE durante las fases del análisis puede resolver muchos de estos problemas.

Actualmente no se usa una sola técnica estándar de diseño y documentación. En las siguientes secciones, discutimos varias técnicas diferentes que actualmente se usan. Cada técnica tiene sus propias ventajas y desventajas, debido a que cada una tiene propiedades únicas.

PSEUDOCÓDIGO

En el capítulo 9, introdujimos el concepto de español estructurado como una técnica de analizar decisiones. El pseudocódigo es similar al español estructurado porque no es un tipo particular de programar código, pero se puede usar como un paso intermedio para desarrollar el código de programa.

La figura 16.16 es un ejemplo de pseudocódigo para Chenoweth Enterprises, un conglomerado del periódico que publica el *Charlie Brown's Journal*, *The Steel Pier Observer* y *Wicked*, el siempre popular periódico orientado a adolescentes. El conglomerado del periódico pasa por un proceso de actualizar, imprimir y proporcionar informes de administración para cada uno de sus periódicos. El pseudocódigo para este proceso involucra un proceso de actualizar cada lista de suscriptores al periódico. Esta estructura se puede ver fácilmente en el pseudocódigo.

En la industria es común el uso del pseudocódigo, pero la falta de estandarización evitará que sea aceptado por todos. Debido a que el pseudocódigo está tan cerca del código de programa, naturalmente es favorecido por programadores y por consiguiente no es favorecido por analistas de negocios. El pseudocódigo con frecuencia se usa para representar la lógica de cada módulo en un diagrama de estructura.

El diagrama de flujo de datos se podría usar para escribir la lógica del pseudocódigo. Al usar un nivel del programa en lugar de un nivel del sistema, el diagrama de flujo de datos podría agregar varios símbolos adicionales. El asterisco (*), que significa “y”, se usa para indicar que deben estar presentes los dos flujos de datos nombrados. Consulte la parte de un diagrama de flujo de datos que se ilustra en la figura 16.17. Si los flujos de datos de entrada son de procesos diferentes, la presencia del conector “y” significa que el proceso que recibe el flujo debe desempeñar alguna clase de coincidencia de archivo o una coincidencia secuencial, leer todos los registros de ambos archivos o una lectura indexada de un segundo archivo usando un campo importante obtenido del primer archivo.

El signo de suma encerrado en un círculo (\oplus) representa un “o” exclusivo e indica que uno u otro flujo de datos está presente en cualquier momento dado. Usar este símbolo implica que el proceso que recibe o produce el flujo de datos debe tener una instrucción correspondiente IF... THEN... ELSE...

FIGURA 16.16

Usar pseudocódigo para describir un servicio de actualización de suscripción para una compañía editorial especializada en periódicos.

```
Abrir archivos
Leer el primer Nombre.periódico
DO WHILE hay más Nombre(s).periódico
  PRINT Fecha
  PRINT nombre.periódico
  Leer el primer registro.suscriptor
  DO WHILE hay más registro(s).suscriptor
    IF Transacción = Modificar.renovación
      THEN Duración.suscrip. = Duración.suscrip. + Número.semanas
    ELSE IF Transacción = Nueva
      THEN PERFORM Agregar.suscriptor
      THEN Duración.suscrip. = Número.semanas
    ELSE IF Transacción = Modificar.dirección
      THEN PERFORM Cambiar.dirección
    ELSE IF Transacción = Eliminar.suscriptor
      THEN PERFORM Preparar.reembolsos.actualizar
      PERFORM Imprimir.reembolso.cheque
      Duración.suscrip. = 0
    ELSE PERFORM Error.en.transacción
  ENDIF
  PERFORM Preparar.suscriptor.lista
  Leer otro Registro.suscriptor
ENDDO
PERFORM Imprimir.suscriptor.lista
Obtener siguiente Nombre.periódico
ENDDO
Cerrar archivos
```

MANUALES DE PROCEDIMIENTO

Los manuales de procedimiento son documentos organizacionales comunes que la mayoría de las personas ha visto. Son el componente en Español de la documentación, aunque también podrían contener códigos de programa, diagramas de flujo, etc. Se pretende que los manuales comuniquen a aquellos que los usan. Podrían contener comentarios de fondo, los pasos requeridos para lograr diferentes transacciones, instrucciones de cómo recuperarse de los problemas y qué hacer si algo no funciona (solucionar problemas). Actualmente muchos manuales están disponibles en línea, con capacidad de hipertexto que facilita el uso.

Se desea un enfoque directo y estandarizado para crear documentación de apoyo de usuario. Para ser útil, la documentación del usuario se debe mantener actualizada. El uso de Web ha revolucionado la velocidad con que los usuarios pueden obtener asistencia. Muchos diseñadores de software están desplazando el soporte de usuario —con la lista de preguntas frecuentes (FAQ), escritorios de ayuda, soporte técnico y servicios de fax— para Web. Además, muchos vendedores de software COTS incluyen archivos “Léame” con descargas o envíos de nuevo software. Estos archivos sirven para varios propósitos: documentan cambios, ajustan o reparan fallas recientemente descubiertas en la aplicación que han ocurrido demasiado tarde en su desarrollo para poder ser incluidas en el manual del usuario.

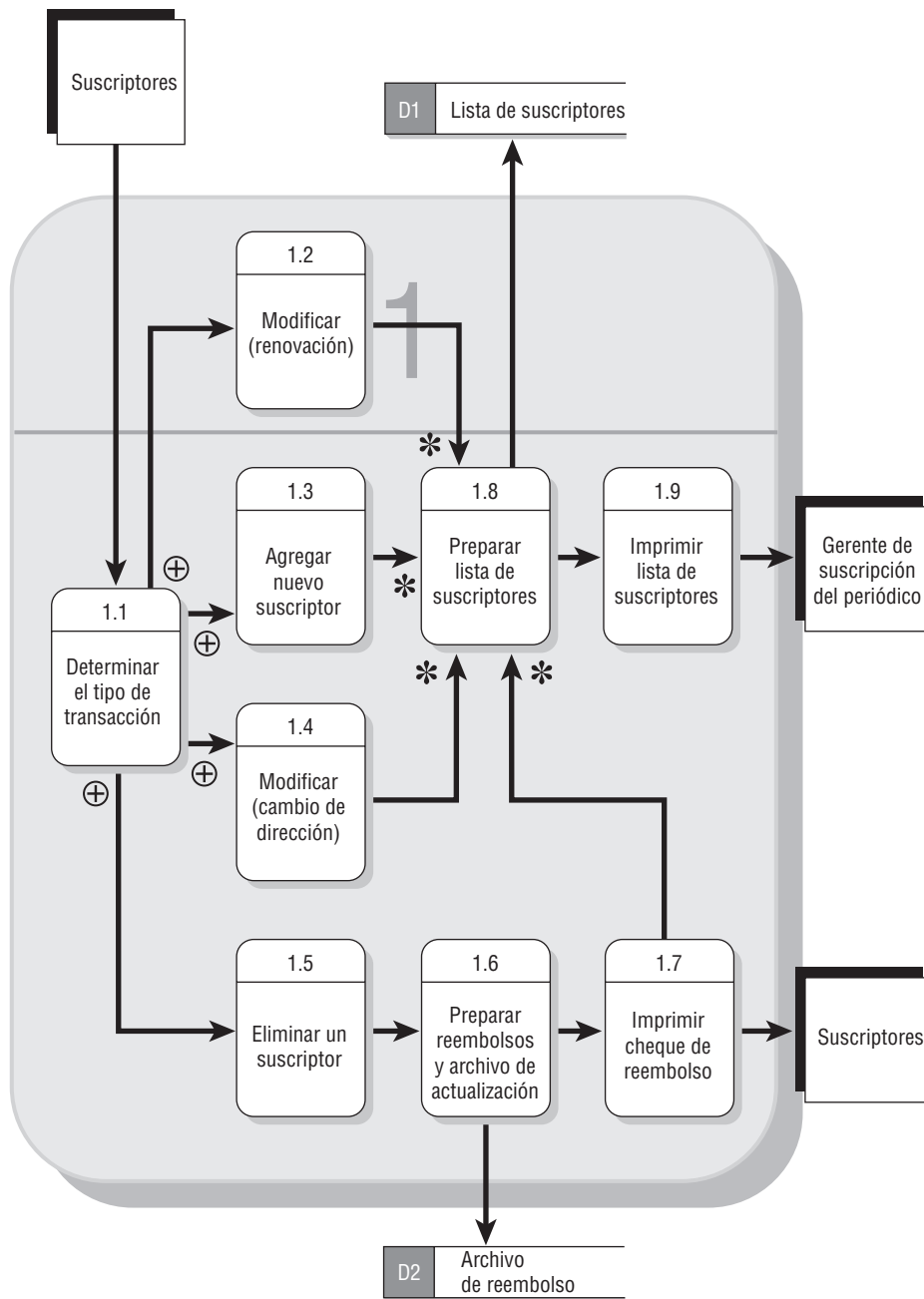


FIGURA 16.17

Se han usado símbolos especiales en el diagrama de flujo de datos del procesamiento de transacción del suscriptor del periódico para ilustrar lógica Y y O.

Las secciones importantes de un manual deben incluir una introducción, cómo usar el software, qué hacer si las cosas salen mal, una sección de referencia técnica, un índice e información de cómo contactar al fabricante. Los manuales en línea en los sitios Web también deben incluir información sobre descargar actualizaciones y una página de FAQ. Los problemas con los manuales de procedimiento son que (1) están mal organizados, (2) es difícil encontrar la información necesaria en ellos, (3) el caso específico en cuestión no aparece en el manual y (4) el manual no está escrito en español. Más adelante, en la sección donde se habla de pruebas, discutimos la importancia de tener usuarios que "prueban" los manuales de sistemas y prototipos de los sitios Web antes de que se terminen.

EL MÉTODO DE FOLKLORE

El FOLKLORE es una técnica de documentación de sistemas creada para complementar algunas de las técnicas ya tratadas. Aun con la abundancia de técnicas disponibles, muchos

FIGURA 16.18

Las costumbres, anécdotas, proverbios y formas artísticas que se usan en el método FOLKLORE de documentación se aplican a los sistemas de información.



sistemas se documentan inadecuadamente o no se documentan en absoluto. El FOLKLORE recopila información que normalmente se comparte entre los usuarios pero raramente se pone por escrito.

El FOLKLORE es una técnica sistemática, basada en métodos tradicionales usados para recopilar el folklore sobre las personas y leyendas. Este enfoque para la documentación de sistemas requiere que el analista entreviste a los usuarios, investigue la documentación existente en los archivos y observe el procesamiento de información. El objetivo es recopilar la información correspondiente a una de cuatro categorías: costumbres, anécdotas, proverbios y formas artísticas. La figura 16.18 sugiere cómo se relaciona cada categoría a la documentación de sistemas de información.

Al documentar las costumbres, el analista (u otro folklorista) intenta capturar por escrito lo que los usuarios hacen para conseguir que los programas puedan ejecutar sin problemas. Un ejemplo de una costumbre es: "Normalmente, nos toma dos días actualizar los registros mensuales porque la tarea es bastante grande. Ejecutamos cuentas comerciales en un día y guardamos las otras para el siguiente día".

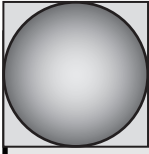
Las anécdotas son historias que los usuarios dicen respecto a cómo funcionó el sistema. Por supuesto, la exactitud de la anécdota depende de la memoria del usuario y es, en el mejor de los casos, una opinión sobre cómo funcionó el programa. Lo siguiente es un ejemplo de una anécdota:

El problema ocurrió de nuevo en 1995. Esa vez, el trabajo LIB409 (actualización mensual) sólo se ejecutó con los registros "tipo 6". Debido a este error, no había registros financieros en el archivo LIBFIN. Cuando intentamos leer el archivo vacío, éste inmediatamente se cerraba y por consiguiente los totales se reportaron como cero. Pudimos corregir este problema agregando un registro "tipo 7" y ejecutando el trabajo.

Las anécdotas normalmente tienen un principio, un cuerpo y un fin. En este caso, tenemos una historia sobre un problema (el principio), una descripción de los efectos (el cuerpo) y la solución (el fin).

Los proverbios son declaraciones breves que representan generalizaciones o consejos. Tenemos muchos proverbios en la vida cotidiana, tal como "Más vale pájaro en mano que ciento volando" o "centavo ahorrado, centavo ganado". En la documentación de sistemas, tenemos muchos proverbios, tal como "Omita esta sección de código y el programa fallará" o "Haga frecuentemente copias de seguridad". A los usuarios les gusta dar consejos y el analista debe intentar capturar dichos consejos e incluirlos en la documentación de FOLKLORE.

Recopilar formas artísticas es otra actividad importante de folkloristas tradicionales, y también el analista de sistemas debe entender su importancia. Los diagramas de flujo, diagramas y tablas que los usuarios diseñan algunas veces podrían ser mejores o más útiles que los diseñados por el autor del sistema original. Los analistas con frecuencia encontrarán tal



ESCRIBIR ES CORRECTO

“Es fácil de entender. Creo que si todos usamos pseudocódigo, no tendremos problemas, es decir, con cosas que no se estén estandarizando”, dice Al Gorithm, un nuevo programador que trabajará con su equipo de análisis de sistemas. Al participa en una reunión informal con tres miembros del equipo de análisis de sistemas, un grupo de trabajo de MIS compuesto por seis miembros del departamento de publicidad, y dos programadores, quienes trabajan en equipo para desarrollar un sistema de información para el personal de publicidad.

Philip, un ejecutivo de cuenta del área de publicidad y miembro del grupo de trabajo de MIS, pregunta sorprendido: “¿Cómo se llama este método?”. Los dos programadores contestan al mismo tiempo: “Pseudocódigo”. Philip contesta, imperturbable: “Eso no me dice nada”.

Neeva Phail, una de las analistas de sistemas, empieza a explicar: “Tal vez no importe tanto lo que utilizemos, si...”

Flo Chart, otra analista de sistemas, la interrumpe: “Yo odio el pseudocódigo”. Flo mira a los programadores en busca de apoyo. “Estoy segura de que podemos ponernos de acuerdo en una mejor técnica.”

David, un ejecutivo de publicidad más experimentado, parece un poco disgustado y dice: “Yo aprendí algo de los diagramas de flujo con los primeros analistas de sistemas que tuvimos hace años. ¿Ustedes ya no los utilizan? Creo que son una mejor opción”.

Lo que al principio era una reunión amistosa repentinamente parece haber llegado a un callejón sin salida. Los participantes se miran unos a otros con recelo. Como un analista de sistemas que ha trabajado en muchos proyectos diferentes con muchos tipos distintos de personas, usted comprende que el grupo espera que usted haga algunas sugerencias razonables.

Con base en lo que usted conoce sobre las diversas técnicas de documentación, ¿qué técnica(s) propondría a los miembros del grupo? ¿De qué manera la(s) técnica(s) que usted proponga solucionará(n) algunas de las preocupaciones que han expresado los miembros del grupo? ¿Qué proceso utilizará para elegir las técnicas apropiadas?

arte en los carteles de anuncios o podrían pedir a los usuarios vaciar sus archivos y recuperar cualquier diagrama útil.

El enfoque FOLKLORE funciona debido a que puede ayudar a reparar la falta de conocimiento cuando un autor de programa se retira. Los contribuyentes al documento FOLKLORE no tienen que documentar el sistema entero, sólo las partes que conozcan. Por último, es divertido para los usuarios contribuir, quitando así algo de carga de los analistas. Observe que la clase de sistemas de recomendación que se discutió anteriormente en el libro está muy cerca de la conceptualización de FOLKLORE. Estos sistemas amplían la idea de FOLKLORE para incluir todos los tipos de recomendaciones, tales como las calificaciones de restaurantes y películas. Usando métodos económicos o gratuitos como el correo electrónico, se han superado algunas barreras iniciales para recopilar y compartir la información informal.

El peligro de confiar en el FOLKLORE es que la información recopilada de los usuarios podría ser correcta, parcialmente correcta o incorrecta. Sin embargo, a menos que alguien se tome el tiempo de rehacer completamente la documentación de programa, la descripción de costumbres, anécdotas, proverbios y formas artísticas podría ser la única información escrita acerca de cómo funciona un grupo de programas.

SELECCIÓN DE UNA TÉCNICA DE DISEÑO Y DOCUMENTACIÓN

Las técnicas discutidas en este capítulo son sumamente valiosas como herramientas de diseño, ayudas de memoria, herramientas de productividad y como medios de reducir las dependencias en los miembros de personal clave. Sin embargo, el analista de sistemas se enfrenta con una decisión difícil con respecto a qué método adoptar. Lo siguiente es un grupo de lineamientos para ayudar al analista a seleccionar la técnica adecuada.

Escoja una técnica que:

1. Es compatible con la documentación existente.
2. Se entiende por otros en la organización.
3. Le permite regresar a trabajar en el sistema después de que ha estado fuera de él por un periodo.
4. Sea conveniente para el tamaño del sistema en que está trabajando.

5. Permita un enfoque de diseño estructurado si se considera como más importante que otros factores.
6. Permita fácil modificación.

CÓMO PROBAR, MANTENER Y AUDITAR

Una vez que el analista ha diseñado y codificado el sistema, probar, mantener y auditar son las primeras consideraciones.

EL PROCESO DE PROBAR

Todos los programas de aplicación del sistema recién escritos o modificados —así como también nuevos manuales de procedimiento, nuevo hardware y todas las interfaces del sistema— se deben probar completamente. Probar al azar y por tanteo no será suficiente.

Las pruebas se hacen durante todo el proceso de desarrollo de sistemas, no sólo al final. Se busca descubrir errores desconocidos hasta ahora, no demostrar la perfección de programas, manuales o equipo.

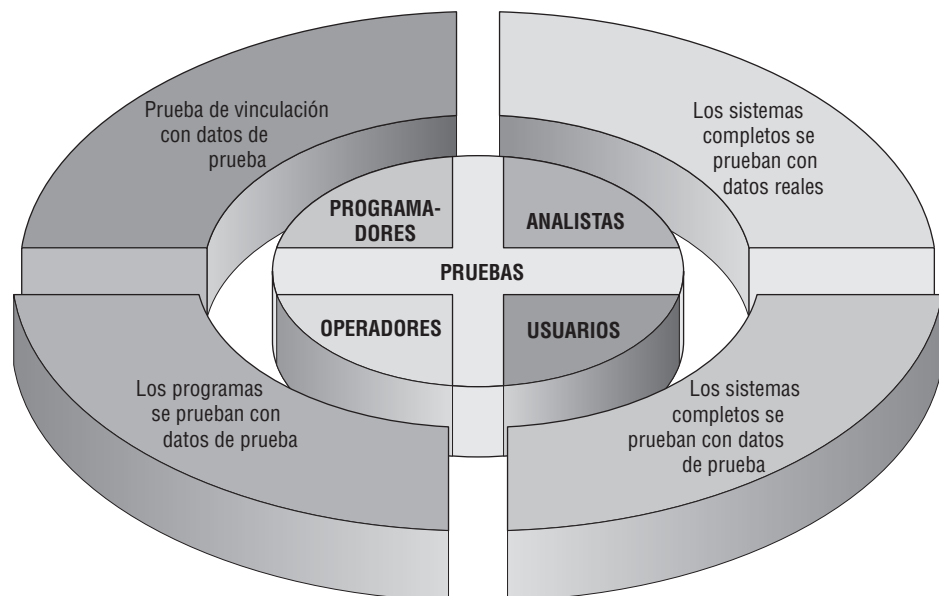
Aunque probar es tedioso, es una serie esencial de pasos que ayuda a asegurar la calidad del eventual sistema. Es mucho menos inquietante probar de antemano que tener un sistema probado deficientemente que falle después de la instalación. Las pruebas se realizan en subsistemas o módulos del programa conforme avance su desarrollo. Las pruebas se hacen en muchos niveles diferentes a varios intervalos. Antes de que el sistema se ponga en producción, todos los programas se deben verificar en el escritorio, verificar con datos de prueba y verificar para ver si los módulos trabajan entre sí como se planeó.

El sistema también se debe probar como un todo en funcionamiento. Incluso hay que probar las interfaces entre los subsistemas; la exactitud de salida; y la utilidad y entendimiento de la documentación y salida de sistemas. Como se muestra en la figura 16.19, los programadores, analistas, operadores y usuarios cumplen un papel diferente en los varios aspectos a probar. Las pruebas de hardware normalmente se proporcionan como un servicio por los vendedores de equipo quienes ejecutarán sus propias pruebas en el equipo cuando se libere en el sitio.

Pruebas de programas con datos de prueba Mucha de la responsabilidad para probar el programa radica en el autor(es) original de cada programa. El analista de sistemas sirve como consejero y coordinador para las pruebas del programa. En esta capacidad, el analista

FIGURA 16.19

Los programadores, analistas, operadores y usuarios desempeñan un papel diferente en probar el software y sistemas.



trabaja para asegurar que los programadores implementen las técnicas de prueba correctas pero probablemente no desempeñe personalmente este nivel de verificación.

En esta fase, los programadores primero deben hacer pruebas de escritorio de sus programas para verificar la forma en que funcionará el sistema. En la verificación de escritorio, el programador sigue cada paso en el programa impreso para verificar si la rutina funciona como se espera.

Luego, los programadores deben crear datos de prueba válidos e inválidos. Estos datos se ejecutan después para ver si las rutinas de base trabajan y también para descubrir errores. Si la salida de los módulos principales es satisfactoria, pueden agregar más datos de prueba para verificar otros módulos. Los datos de prueba creados deben probar posibles valores mínimos y máximos así como también todas las variaciones posibles en el formato y códigos. Los archivos de salida de los datos de prueba se deben verificar cuidadosamente. Nunca se debe asumir que los datos contenidos en un archivo son correctos sólo porque un archivo fue creado y accesado.

A lo largo de este proceso, el analista de sistemas verifica la salida en busca de errores, avisando al programador de cualesquier correcciones necesarias. El analista normalmente no recomendará o creará datos de prueba para las pruebas de programas pero podría señalar al programador las omisiones de tipos de datos a ser agregados en pruebas posteriores.

Prueba de vínculos con datos de prueba Cuando los programas pasan la verificación de escritorio y la verificación con datos de prueba, se deben pasar por las pruebas de vínculos, que también se conocen como prueba de cadena. Estas pruebas verifican si los programas que realmente son interdependientes trabajan juntos como se planeó.

Una pequeña cantidad de datos de prueba, normalmente diseñados por el analista de sistemas para probar las especificaciones del sistema así como también los programas, se usa para las pruebas de vinculación. Podría tomar varios pasos a través del sistema para probar todas las combinaciones, debido a que es inmensamente difícil resolver los problemas si intenta probar todo a la vez.

El analista crea datos de prueba especiales que cubren una variedad de situaciones de procesamiento para la prueba de vinculación. Primero, se procesan datos de prueba típicos para ver si el sistema puede manejar transacciones normales, aquellas que constituirían el mayor volumen de su carga. Si el sistema funciona con transacciones normales, se agregan las variaciones, incluyendo los datos inválidos para asegurar que el sistema puede detectar adecuadamente los errores.

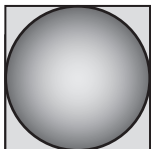
Prueba completa de sistemas con datos de prueba Cuando las pruebas de vinculación se concluyen satisfactoriamente, se debe probar el sistema como una entidad completa. En esta fase, los operadores y usuarios finales se involucran activamente en la prueba. Los datos de prueba, creados por el equipo de análisis de sistemas para el propósito expreso de probar los objetivos del sistema, se usan.

Como se puede esperar, hay varios factores a considerar cuando se prueban los sistemas con datos de prueba:

1. Examinar si los operadores tienen la documentación adecuada en los manuales de procedimiento (en papel o en línea) para asegurar un funcionamiento correcto y eficaz.
2. Verificar si los manuales de procedimiento son lo bastante claros como para comunicar cómo se deben preparar los datos para la entrada.
3. Determinar si los flujos de trabajo necesarios para el sistema nuevo o modificado realmente “fluyen”.
4. Determinar si la salida es correcta y si los usuarios entienden que esta salida es como se verá en su formulario final.

Recuerde fijar el tiempo adecuado para la prueba del sistema. Desafortunadamente, con frecuencia este paso se elimina si la instalación del sistema se retrasa de la fecha indicada.

La prueba de sistemas incluye reafirmar los estándares de calidad para el desempeño del sistema que se estableció cuando se hicieron las especificaciones iniciales del sistema. Todos los involucrados deben estar de acuerdo una vez más en cómo determinar si el sistema está haciendo lo que se supone que hace. Este paso incluirá medidas de error, oportuni-



ESTUDIANDO PARA SU PRUEBA DE SISTEMAS

“Tenemos el tiempo encima. Tan sólo mira esta proyección”, dice Lou Scuntroll, el miembro más nuevo de su equipo de análisis de sistemas, mientras le muestra el diagrama PERT que el equipo ha estado usando para proyectar la fecha en que el nuevo sistema quedaría listo. “Quizá no podamos cumplir la fecha prevista de julio para realizar las pruebas con datos reales. Estamos atrasados tres semanas debido a que el equipo se embarcó tarde.”

Como uno de los analistas de sistemas que han vivido la presión de fijar y reprogramar fechas límite en otros proyectos, usted intenta permanecer tranquilo y evaluar cuidadosamente la situación antes de hablar. Lentamente, usted plantea a Lou la posibilidad de postergar las pruebas.

Lou contesta: “Si tratamos de retrasar las pruebas hasta las primeras semanas de agosto, en esas fechas dos personas importantes de contabilidad estarán de vacaciones”. Lou está visiblemente molesto con la posibilidad de retrasar la fecha límite.

Stan Dards, otro miembro reciente de su equipo de análisis de sistemas, entra en la oficina de Lou. “Ambos tienen un aspecto pésimo. Todo está bien, ¿no es cierto? No me reasignaron a programar una aplicación de nómina, ¿o sí?”

A Lou no le hace gracia el sentido del humor de Stan ni su aparente preocupación por sí mismo. “Todo estaba bien hasta antes de que llegaras. Estábamos a punto de tomar algunas decisiones importantes sobre el calendario.” Lou le pasa a Stan el diagrama PERT para que lo revise.

“Observa la fecha de pruebas de julio. Como podrás darte cuenta, no hay manera de cumplirla. ¿Alguna brillante idea?”

Stan contempla unos instantes el diagrama, luego señala: “Algo ha pasado. Veamos aquí... quizá si movemos el módulo de contabilidad a...”

Lou lo interrumpe bruscamente: “¡No!, ya pensamos en eso, pero Stanford y Binet, de contabilidad, están de vacaciones en agosto. Quizá podríamos omitir esa parte de las pruebas. Ellos han sido muy cooperativos. No creo que pongan ninguna objeción si sacamos los sistemas y hacemos las pruebas ya que estemos en la fase de producción”.

“Creo que ésa es una buena idea, Lou”, coincide Stan, tratando de congraciarse con Lou después de sus bromas anteriores. “No hemos tenido ningún problema real con eso, y los programadores son confiables. Así podríamos mantener el calendario con todo lo demás. Yo voto por *no* probar la parte de contabilidad, y sólo darle un vistazo cuando se ponga en marcha.”

Como el miembro con más experiencia del equipo, ¿qué puede usted argumentar para convencer a Lou y Stan de la importancia de probar el módulo de contabilidad con datos reales? ¿Qué pueden hacer los analistas de sistemas para planificar sus actividades de tal manera que le dediquen un tiempo razonable a realizar las pruebas con datos reales? ¿Cuáles son algunos de los posibles problemas que los miembros del equipo podrían encontrar si no prueban el sistema completamente con datos reales antes de poner el sistema en producción? ¿Hay en el proceso de análisis y diseño de sistemas pasos que puedan omitirse con el propósito de poner al día un proyecto retrasado?

dades, facilidad de uso, clasificación apropiada de transacciones, tiempo fuera de servicio aceptable y manuales de procedimiento entendibles.

Prueba completa de sistemas con datos reales Cuando las pruebas de sistemas con datos de prueba se realizan de manera satisfactoria, es bastante recomendable probar el nuevo sistema repetidas veces con lo que se conoce como datos reales, datos que se han procesado de manera exitosa con el sistema existente. Este paso permite una comparación precisa de la salida del nuevo sistema con la salida que sabe ha sido procesada correctamente, y también es una buena opción para probar cómo se manejarán los datos reales. Obviamente, este paso no es posible al crear salidas completamente nuevas (por ejemplo, salida de una transacción de comercio electrónico de un nuevo sitio Web corporativo). Al igual que con los datos de prueba, sólo se usan pequeñas cantidades de datos reales en este tipo de prueba de sistemas.

Las pruebas constituyen un periodo importante para evaluar la manera en que los usuarios finales y los operadores interactúan realmente con el sistema. Aunque se da mucha importancia a la interacción de usuario-sistema (véase el capítulo 14), nunca podrá predecir totalmente el amplio rango de diferencias en la forma en que los usuarios interactuarán realmente con el sistema. No basta con entrevistar a los usuarios sobre cómo interactúan con el sistema; debe observarlos directamente.

Los aspectos que debe vigilar son la facilidad con que un usuario aprende el sistema y sus reacciones a la retroalimentación del sistema, incluyendo lo que hace cuando recibe un mensaje de error y cuando se le informa que el sistema está ejecutando sus comandos. Ponga especial atención a la manera en que reaccionan los usuarios al tiempo de respuesta del sistema y a la redacción de las respuestas. También escuche lo que dicen los usuarios acerca del sistema cuando trabajan en él. Es necesario resolver todos los problemas reales antes de que el sistema se ponga en producción, no sólo considerarlos como ajustes al sistema que los usuarios y operadores deben hacer por sí mismos.

Como se mencionó anteriormente, también los manuales de procedimiento necesitan ser probados. Aunque los manuales se pueden corregir por el personal de apoyo, y verificar su exactitud técnica por el equipo de análisis de sistemas, la única forma real de probarlos es tener usuarios y operadores que los prueben, de preferencia durante la prueba de sistemas completos con datos reales. Haga que usen versiones precisas, pero no necesariamente versiones finales de los manuales.

Es difícil comunicar con precisión los procedimientos. Demasiada información será como un obstáculo para el uso del sistema. El uso de documentos basados en Web puede ayudar en esta consideración. Los usuarios pueden pasar a los temas de interés y descargar e imprimir lo que quieren conservar. Considere las sugerencias del usuario e incorpórelas en las versiones finales de páginas Web, manuales impresos y otra documentación.

PRÁCTICAS DE MANTENIMIENTO

Su objetivo como analista de sistemas debe ser instalar o modificar sistemas que tienen una vida bastante útil. Quiere crear un sistema cuyo diseño es bastante comprensivo y previsorio para atender las necesidades actuales y proyectadas del usuario durante varios años. Debe usar parte de su experiencia para proyectar lo que podrían ser esas necesidades y después construir flexibilidad y adaptabilidad en el sistema. Lo mejor y más fácil del diseño de sistemas será asegurar que el negocio tendrá que gastar menos dinero en el mantenimiento.

Reducir los costos de mantenimiento es una consideración principal, debido a que el mantenimiento de software aislado puede consumir más de 50 por ciento del presupuesto de procesamiento de datos para un negocio. Los costos de mantenimiento excesivos se reflejan directamente en el diseñador del sistema, debido a que aproximadamente 70 por ciento de errores de software se han atribuido al diseño de software inadecuado. Desde una perspectiva de sistemas, tiene sentido que detectar y corregir a tiempo los errores de diseño de software es menos costoso que permitir que permanezcan inadvertidos hasta que sea necesario el mantenimiento.

Por lo regular el mantenimiento se realiza para mejorar el software existente en lugar de responder a una crisis o falla del sistema. Al igual que con el cambio de requerimientos del usuario, el software y la documentación se deben cambiar como parte del trabajo de mantenimiento. Además, los programas se podrían recodificar para mejorar la eficacia del programa original. Más de la mitad de todo el mantenimiento está compuesto de dicho trabajo de mejora.

El mantenimiento también se hace para actualizar el software en respuesta a la organización cambiante. Este trabajo no es tan sustancial como mejorar el software, pero se debe hacer. El mantenimiento de emergencia y de adaptación representa menos de la mitad de todo el mantenimiento del sistema.

Parte del trabajo del analista de sistemas es asegurar que en el lugar haya procedimientos y canales adecuados para permitir retroalimentación sobre —y respuestas subsecuentes para— las necesidades de mantenimiento. Los usuarios deben poder comunicar fácilmente los problemas y sugerencias a aquellos que estarán manteniendo el sistema. Es muy desalentador si el sistema no se mantiene adecuadamente. Las soluciones consisten en proporcionar a los usuarios acceso a correo electrónico para el soporte técnico, así como también permitirles descargar actualizaciones de producto o ajustes de Web.

El analista de sistemas también necesita establecer un esquema de clasificación para permitir a usuarios designar la importancia percibida del mantenimiento sugerido o solicitado. Clasificar las solicitudes permite a programadores de mantenimiento entender cómo estiman los usuarios la importancia de sus solicitudes. Este punto de vista, junto con otros factores, se puede tener en cuenta al establecer el mantenimiento.

CÓMO AUDITAR

Auditar es otra forma de asegurar la calidad de la información contenida en el sistema. Ampliamente definido, auditar se refiere a pedirle a un experto, que no esté involucrado en crear o usar un sistema, examinar la información para determinar su fiabilidad. Ya sea que la

información se establezca o no para ser fiable, el descubrimiento en su fiabilidad se comunica a otros con el propósito de hacer la información del sistema más útil para ellos.

Generalmente hay dos tipos de auditores para los sistemas de información: interno y externo. Determinar si ambos son necesarios para el sistema que usted diseña, dependerá de qué tipo de sistema es. Los auditores internos trabajan para la misma organización que posee el sistema de información, mientras que los externos (también llamados independientes) se contratan por fuera.

Los auditores externos se usan cuando el sistema de información procesa datos que influyen en las declaraciones financieras de una compañía. Los auditores externos auditan el sistema para asegurar la veracidad de las declaraciones financieras que se producen. También se podrían traer si ocurre algo fuera de lo normal que involucra a los empleados de la compañía, tal como la sospecha de un fraude electrónico o un desfalco.

Los auditores internos estudian los controles usados en el sistema de información para estar seguros que son adecuados y que están haciendo lo que deben hacer. También prueban la suficiencia de controles de seguridad. Aunque trabajan para la misma organización, los auditores internos no informan a las personas responsables del sistema que están auditando. El trabajo de los auditores internos con frecuencia es más detallado que el de los auditores externos.

RESUMEN

El analista de sistemas usa tres enfoques amplios de la administración de calidad total (TQM) para analizar y diseñar sistemas de información: diseñar sistemas y software con un enfoque descendente y modular; diseñar y documentar sistemas y software usando métodos sistemáticos; y probar sistemas y software de manera que se puedan mantener y auditar fácilmente.

Seis Sigma es una cultura, filosofía, metodología y enfoque para la calidad que tiene como meta la eliminación de todos los defectos. Los siete pasos de un enfoque Seis Sigma son: (1) definir el problema; (2) observar el problema; (3) analizar las causas; (4) actuar en las causas; (5) estudiar los resultados; (6) estandarizar los cambios, y (7) sacar conclusiones.

Los usuarios son extremadamente importantes para establecer y evaluar, desde varias dimensiones, la calidad de los sistemas de información de administración y de los sistemas de apoyo a la toma de decisiones. Se pueden involucrar en la evolución entera de sistemas a través del establecimiento de fuerza de tarea de SI o círculos de calidad.

TQM se puede implementar con éxito al tomar un enfoque descendente (arriba a abajo) para diseñar. Este enfoque se refiere a observar primero los objetivos generales de la organización y después dividirlos en requerimientos manejables de subsistemas. El desarrollo modular hace la programación, depuración y mantenimiento más fácil de lograr. La programación en módulos se complementa bien con un enfoque descendente.

Dos sistemas vinculan programas en el entorno de Windows. Uno es DDE (intercambio dinámico de datos), el cual comparte código usando archivos de biblioteca de vínculos dinámicos (DLL). Al usar DDE, un usuario puede almacenar datos en un programa y después usarlos en otro. Un segundo enfoque para vincular programas en Windows es OLE (vinculación e incrustación de objetos). Debido a su enfoque orientado a objetos, este método de vinculación es superior a DDE para vincular datos y gráficos de la aplicación.

Una herramienta recomendada para diseñar un sistema con un enfoque descendente y modular se denomina diagrama de estructura. Se usan dos tipos de flechas para indicar los tipos de parámetros que se pasan entre los módulos. El primero se denomina pareja de datos y el segundo se denomina bandera de control. Los módulos de un diagrama de estructura entran en una de tres categorías: control, transformacional (a veces denominado trabajador) y funcional o especializado.



“Éste es un lugar fascinante para trabajar. Estoy seguro de que usted coincide conmigo ahora que ha tenido la oportunidad de observarnos. A veces creo que debe ser divertido ser de fuera... ¿no se siente como un antropólogo que descubre una nueva cultura? Recuerdo cuando llegué aquí por primera vez. Todo era tan nuevo, tan extraño. ¡Vaya!, incluso el idioma era diferente. No era un ‘consumidor’; era un ‘cliente’. No teníamos ‘departamentos’; teníamos ‘unidades’. No es una cafetería para empleados; es la ‘taberna’. Esto también se aplica a la manera en que trabajamos. Todos tenemos diferentes maneras de enfocar las cosas. Creo que he logrado entender lo que Snowden quiere, pero también de vez en cuando cometo algún error. Por ejemplo, si le doy trabajo en un disco, igual de sencillo es para él verlo así que en un informe impreso. ¡Por eso también tengo dos computadoras en mi escritorio! Siempre lo veo a usted tomar tantos apuntes... Sin embargo, creo que esto tiene sentido. Se supone que usted documenta lo que nosotros hacemos con nuestros sistemas e información, así como lo que su equipo hace, ¿no es así?”

PREGUNTAS DE HYPERCASE

1. Use el método FOLKLORE para completar la documentación del sistema GEMS de la Unidad de Sistemas de Información Gerencial. Asegúrese de incluir costumbres, anécdotas, proverbios y formas artísticas.
2. En dos párrafos, sugiera una manera de capturar los elementos de FOLKLORE en una PC de tal manera que no sea necesario usar un registro en papel. Asegúrese de que la solución que sugiera incluya gráficos y texto.
3. Diseñe pantallas de entrada y salida para FOLKLORE que permitan ingresar datos con facilidad, y proporcione mensajes que recuerden de inmediato los elementos de FOLKLORE.



FIGURA 16.HC1

En HyperCase puede utilizar FOLKLORE para documentar formas artísticas que los usuarios hayan creado o recopilado para darle sentido a sus sistemas.

La parte de TQM es para ver que los programas y sistemas se diseñan, documentan y mantienen adecuadamente. Algunas de las técnicas estructuradas que pueden ayudar al analista de sistemas son pseudocódigo, manuales de procedimiento y FOLKLORE. El pseudocódigo se usa con frecuencia para representar la lógica de cada módulo o diagrama de estructura. El pseudocódigo se puede usar para repastos estructurados. Los analistas de sistemas deben escoger una técnica que se adapte bien con lo que se usó previamente en la organización y que permita flexibilidad y fácil modificación.

La prueba de programas específicos, subsistemas y sistemas totales es esencial para la calidad. La prueba se hace para detectar cualesquier problemas existentes con los programas y sus interfaces antes de que el sistema se use realmente. La prueba normalmente se hace en una forma ascendente, con códigos de programa que primero se verifican en el escritorio. La prueba del sistema completo con datos reales (datos reales que se han procesado exitosamente con el sistema viejo), se logra siguiendo varios pasos intermedios de prueba. Esta prueba proporciona una oportunidad de resolver cualesquier problemas que surjan antes de que el sistema se ponga en producción.

El mantenimiento del sistema es una consideración importante. El software bien diseñado puede ayudar a reducir los costos de mantenimiento. Los analistas de sistemas necesitan establecer canales para recibir la retroalimentación del usuario en las necesidades del mantenimiento, debido a que los sistemas que no se mantienen quedarán obsoletos. Los sitios Web pueden ayudar al respecto al proporcionar acceso a las actualizaciones de productos e intercambios de correo electrónico con personal técnico.

Los auditores internos y externos se usan para determinar la fiabilidad de la información del sistema. Ellos comunican sus resultados de la auditoría a otros para mejorar la utilidad de la información del sistema.

PALABRAS Y FRASES CLAVE

acoplamiento de sello	módulo transformacional
administración de la calidad total (TQM)	parejas de datos
auditor interno	prueba completa de sistemas con datos de prueba
bandera de control (interruptor)	prueba completa de sistemas con datos reales
biblioteca de vínculos dinámicos (DLL)	prueba de programas con datos de prueba
círculo de calidad de SI	prueba de vínculos con datos de prueba (prueba de cadenas)
desarrollo modular	pseudocódigo
diagrama de estructura	repaso estructurado
diseño ascendente	Seis Sigma
diseño descendente	subordinación inadecuada
documentación de software	verificación de escritorio
FOLKLORE	vinculación e incrustación de objetos (OLE)
intercambio dinámico de datos (DDE)	
mantenimiento de software	
módulo de control	
módulo funcional	

PREGUNTAS DE REPASO

1. ¿Cuáles son los tres enfoques amplios disponibles para el analista de sistemas para lograr la calidad en los sistemas recientemente desarrollados?
2. ¿Cuál es el factor más importante para establecer y evaluar la calidad de sistemas de información o sistemas de apoyo a la toma de decisiones? ¿Por qué?
3. Defina el enfoque de administración de la calidad total (TQM) conforme se aplica al análisis y diseño de sistemas de información.
4. ¿Qué significa el término *Seis Sigma*?
5. ¿Qué es un círculo de calidad de SI?
6. Defina el significado de hacer un repaso estructurado. ¿Quién debe estar involucrado? ¿Cuándo se debe hacer un repaso estructurado?

7. Mencione las desventajas de tomar un enfoque ascendente para diseñar.
8. Mencione las ventajas de tomar un enfoque descendente para diseñar.
9. ¿Cuáles son las tres desventajas principales de tomar un enfoque de descendente para diseñar?
10. Defina el desarrollo modular.
11. Mencione cuatro lineamientos para la programación modular correcta.
12. ¿Cómo ayudan los diagramas de estructura al analista?
13. Mencione los dos tipos de flechas usados en los diagramas de estructura.
14. ¿Por qué queremos mantener el número de flechas al mínimo al usar los diagramas de estructura?
15. ¿Por qué se deben pasar hacia arriba las banderas de control en los diagramas de estructura?
16. Mencione dos formas en que el diagrama de flujo de datos ayuda a construir un diagrama de estructura.
17. Mencione las tres categorías de módulos. ¿Por qué se usan en los diagramas de estructura?
18. ¿Cómo puede ayudar un sitio Web a mantener el sistema y su documentación?
19. Proporcione dos razones que apoyen la necesidad de sistemas bien desarrollados y documentación de software.
20. Defina el pseudocódigo.
21. Mencione las cuatro quejas principales que los usuarios expresan sobre los manuales de procedimiento.
22. ¿En cuáles cuatro categorías el método de documentación de FOLKLORE recopila la información?
23. Mencione seis lineamientos para escoger una técnica de diseño y documentación.
24. ¿De quién es la responsabilidad principal para probar los programas de cómputo?
25. ¿Cuál es la diferencia entre datos de prueba y datos reales?
26. ¿Cuáles son los dos tipos de auditores de sistemas?

PROBLEMAS

1. Uno de los miembros de su equipo de análisis de sistemas ha estado desalentando los comentarios de los usuarios sobre los estándares de calidad, argumentando que debido a que ustedes son los expertos, realmente son los únicos que saben lo que constituye un sistema de calidad. En un párrafo, explique al miembro de su equipo por qué es importante obtener las opiniones de los usuarios para la calidad del sistema. Use un ejemplo.

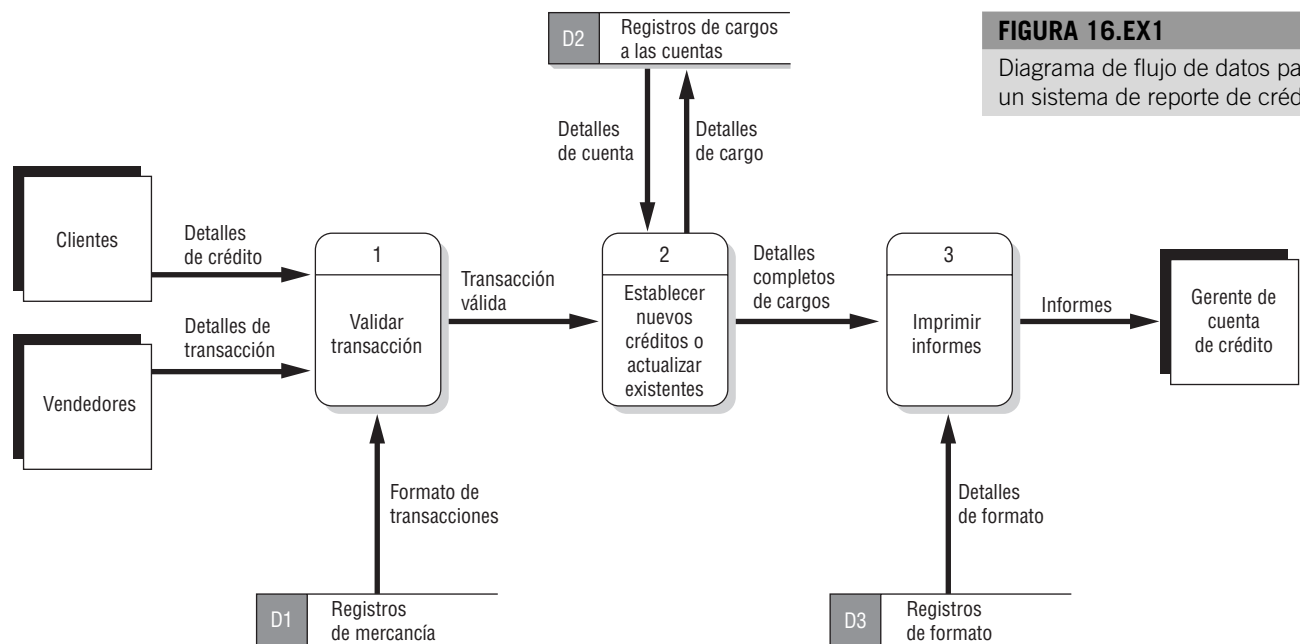


FIGURA 16.EX1

Diagrama de flujo de datos para un sistema de reporte de crédito.

2. Dibuje un diagrama de estructura para el sistema de reporte de crédito en la figura 16.EX1.
3. Escriba el pseudocódigo para el problema 2.
4. Escriba el pseudocódigo para la política de renta de Citron Car proporcionada en la Oportunidad de consultoría 9.3.
5. Escriba una tabla de contenidos detallada para un manual de procedimientos que explique a los usuarios cómo conectarse a la red de cómputo de su escuela, así como también las políticas de la red (quién es un usuario autorizado, etc.). Asegúrese de que el manual se escriba pensando en el usuario.
6. Su equipo de análisis de sistemas está cerca de completar un sistema para Meecham Feeds. Roger está bastante seguro de que los programas que ha escrito para el sistema de inventario de Meecham funcionarán como se requiere, debido a que son similares a los programas que ha hecho antes. Su equipo ha estado muy ocupado y le gustaría empezar a realizar la prueba completa de sistemas tan pronto como sea posible.

Dos miembros jóvenes de su equipo han propuesto lo siguiente:

 - a. Omitir la verificación de escritorio de los programas (debido a que programas similares se verificaron en otras instalaciones; Roger está de acuerdo).
 - b. Hacer la prueba de vínculos con grandes cantidades de datos para comprobar que el sistema funcionará.
 - c. Hacer la prueba completa de sistemas con grandes cantidades de datos reales para demostrar que el sistema está funcionando.

Responda a cada uno de los tres pasos del programa de prueba propuesto. Explique su respuesta en un párrafo.
7. Proponga un plan de pruebas modificado para Meecham Feeds (problema 6). Divida su plan en una secuencia de pasos detallados.

PROYECTOS DE GRUPO

1. Divida su grupo en dos subgrupos. Un subgrupo debe entrevistar a los miembros del otro subgrupo sobre sus experiencias al registrarse en una clase. Se deben diseñar preguntas para obtener información sobre costumbres, anécdotas, proverbios y formas artísticas que ayudarán a documentar el proceso de registro de su escuela.
2. Reúna a su grupo para desarrollar una página Web para un extracto de un manual de FOLKLORE que documente el proceso de registro para una clase, basado en el FOLKLORE que se utilizó en las entrevistas del proyecto 1. Recuerde incluir ejemplos de costumbres, anécdotas, proverbios y formas artísticas.

BIBLIOGRAFÍA SELECCIONADA

- Dean, J. W., Jr. y J. R. Evans, *Total Quality*, St. Paul, MN: West, 1994.
- Deming, W. E., *Management for Quality and Productivity*, Cambridge, MA: MIT Center for Advanced Engineering Study, 1981.
- Juran, J. M., *Managerial Breakthrough*, Nueva York: McGraw-Hill, 1964.
- Kendall, J. E. y P. Kerola, "A Foundation for the Use of Hypertext Based Documentation Techniques", *Journal of End User Computing*, vol. 6, núm. 1, invierno de 1994, pp. 4-14.
- Kendall, K. E. y R. Losee, "Information System FOLKLORE: A New Technique for System Documentation", *Information and Management*, vol. 10, núm. 2, 1986, pp. 103-111.
- Kendall, K. E. y S. Yoo, "Pseudocódigo-Box Diagrams: An Approach to More Understandable, Productive, and Adaptable Software Design and Coding", *International Journal on Policy and Information*, vol. 12, núm. 1, junio de 1988, pp. 39-51.
- Lee, S. M. y M. J. Schniederjans, *Operations Management*, Boston: Houghton-Mifflin, 1994.



ALLEN SCHMIDT, JULIE E. KENDALL Y KENNETH E. KENDALL

DIAGRAMACIÓN DE LA ESTRUCTURA

16

“Aquí las tienes, como lo prometimos”, dicen Chip y Anna triunfalmente al entregar sus especificaciones a Mack Roe, el programador del proyecto.

“Gracias”, responde Mack. “Tengo mucho trabajo por delante.”

Mack empieza a crear un diagrama de estructura para cada programa y luego pasa al diseño de cada módulo. En la figura E16.1 se muestra el diagrama de estructura PRODUCE SOFTWARE CROSS-REFERENCE REPORT. La “C” antes de cada número de módulo se

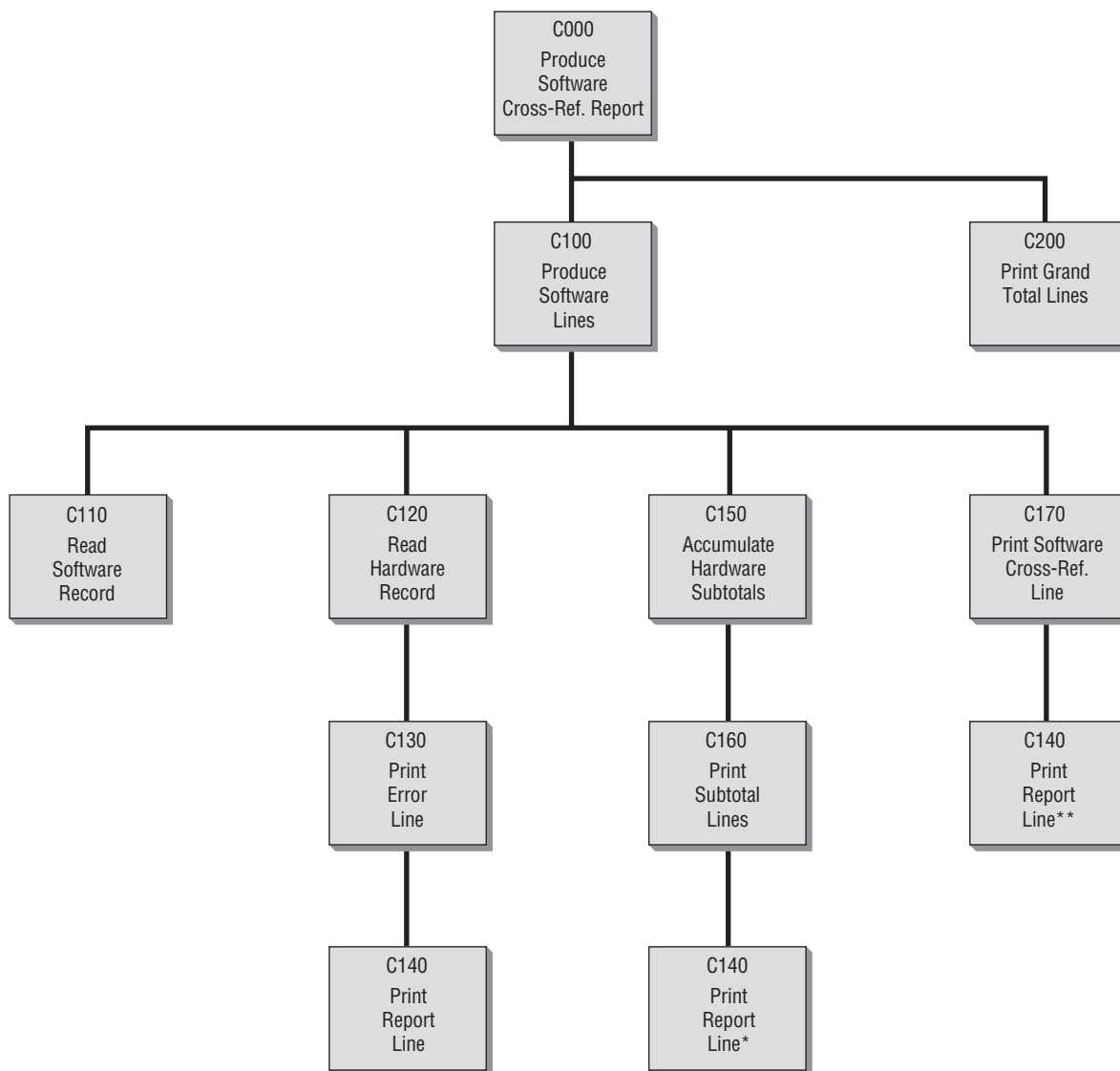


FIGURA E16.1

Diagrama de estructura PRODUCE SOFTWARE CROSS-REFERENCE REPORT. El asterisco sencillo (*) y el asterisco doble (**) indican la segunda y tercera ocurrencias del módulo C140.

16

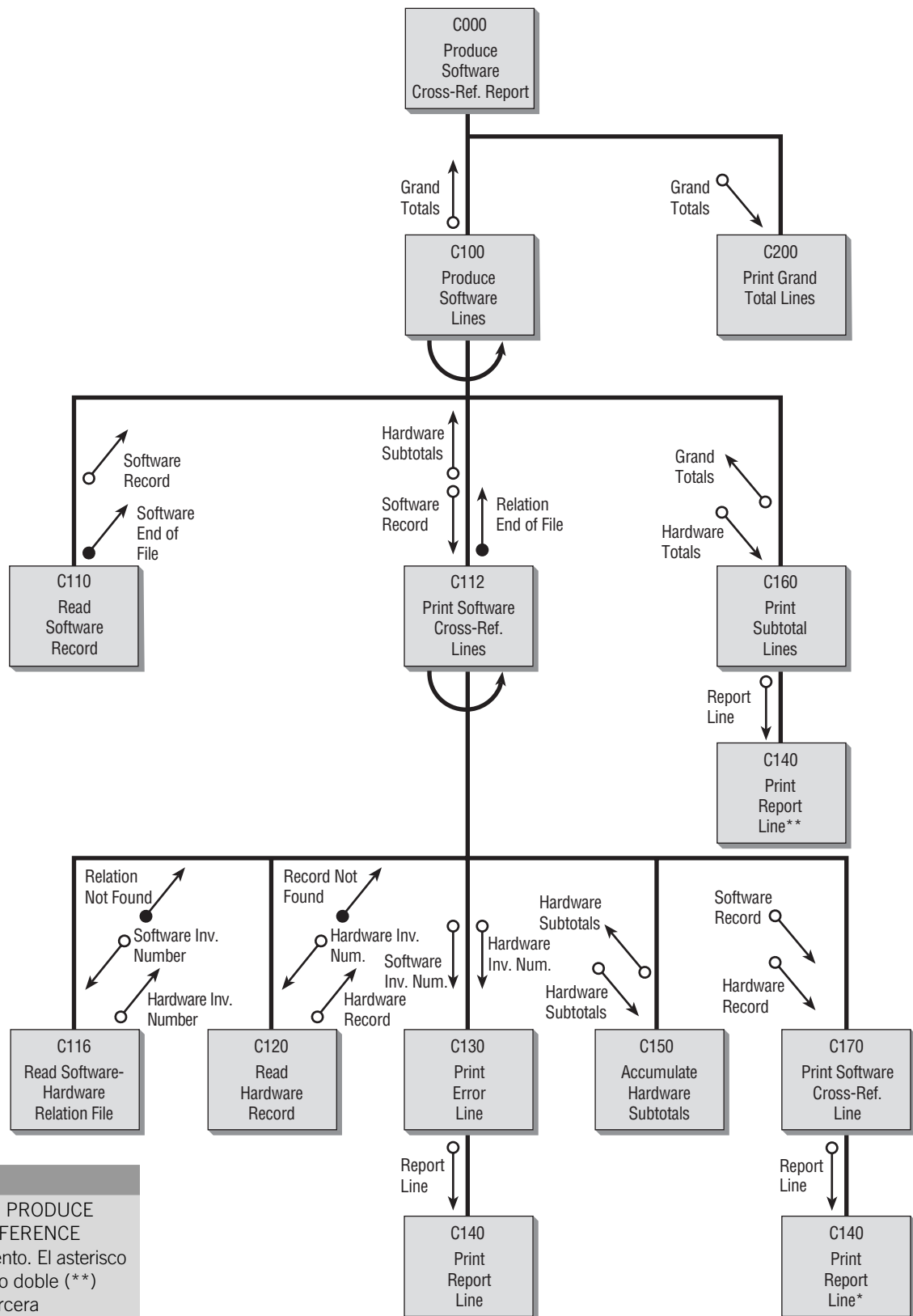
**FIGURA E16.2**

Diagrama de estructura PRODUCE SOFTWARE CROSS-REFERENCE REPORT, con acoplamiento. El asterisco sencillo (*) y el asterisco doble (**) indican la segunda y tercera ocurrencias del módulo C140.

16

refiere al CROSS-REFERENCE REPORT (Visible Analyst requiere una letra como primer carácter en el nombre de un módulo). Este borrador es el primero que Mack utiliza en un repaso estructurado con Dee Ziner, una programadora con experiencia.

Dee Ziner tiene varias sugerencias importantes para mejorar la estructura. Ella dice: “El módulo C130, PRINT ERROR LINE, está erróneamente subordinado al módulo de llamada C120, READ HARDWARE RECORD. Se podría hacer la pregunta: ‘¿El programa debe imprimir una línea de error para leer un HARDWARE RECORD?’ Puesto que la respuesta es no, el módulo debe colocarse en el mismo nivel que C120, READ HARDWARE RECORD”.

Dee continúa analizando la situación con Mack, y dice: “Lo mismo ocurre con el módulo C160, PRINT SUBTOTAL LINES. Ésta no es una función de acumulación de subtotales de hardware y por lo tanto no se debe llamar desde el módulo C150, ACCUMULATE HARDWARE SUBTOTALS”. Dee continúa el repaso y plantea la pregunta: “¿Un SOFTWARE RECORD se podría localizar en muchas máquinas?” Mack responde que eso sí es válido, y otro módulo de control, PRINT SOFTWARE CROSS-REFERENCE LINE, se incluye en el diagrama de estructura.

Mack procede a incorporar los cambios al diagrama de estructura. Cuando se establece la jerarquía correcta, se agrega el acoplamiento. Se pone especial cuidado a pasar el mínimo de datos y a pasar control sólo hacia arriba del diagrama de estructura. En la figura E16.2 se ilustra la versión final. El módulo C116 es nuevo, y se utiliza el archivo SOFTWARE/HARDWARE RELATION para enlazar un SOFTWARE RECORD a muchos HARDWARE RECORDS. El SOFTWARE INVENTORY NUMBER se pasa hacia abajo del módulo y el archivo de relación se lee de manera aleatoria. El HARDWARE INVENTORY NUMBER y el interruptor de control RELATION NOT FOUND se pasan hacia arriba de la estructura.

El diagrama de estructura final tiene una forma funcional. Unos cuantos módulos de control en la parte superior de la estructura, varios módulos trabajadores a la mitad y unos cuantos módulos especializados en la parte inferior le dan un aspecto general equilibrado. Todos los nombres de los módulos tienen la estructura verbo-adjetivo-sustantivo, y describen la tarea que se realiza una vez que termina la ejecución del módulo. Por ejemplo, el módulo C150 tiene el verbo “accumulate”, que describe la tarea que desempeña el módulo. “Subtotals”, un sustantivo, se acumula, y “hardware” describe cuáles subtotales se acumulan.

Cada uno de los módulos del diagrama de estructura se describió en el depósito. La figura E16.3 ilustra la pantalla que describe la función del módulo C100, PRODUCE SOFTWARE LINES. Observe que la descripción del módulo contiene pseudocódigo que muestra la lógica del módulo. Dado que PRODUCE SOFTWARE LINES es un módulo de control, su lógica debe consistir de bucles y toma de decisiones, con muy pocas instrucciones relativas a los detalles del procesamiento, como ADD o READ.

Cada pareja de datos y control del diagrama de estructura se podría describir también en el depósito. El área **Related to** ofrece un vínculo a la entrada del depósito que contiene los detalles de los elementos contenidos en la pareja de datos.

“Bueno, creo que estamos a punto de terminar los diagramas para los programadores”, comenta Chip.

“De crear los diagramas, sí”, repone Anna, “pero les podemos dar un poco más”.

“¿Qué quieres decir?”, pregunta Chip, un tanto desconcertado.

“Usemos Visible Analyst para generar las tablas de la base de datos para Microsoft Access”, responde Anna. “Pienso que podríamos comenzar con una de las entidades principales, como SOFTWARE MASTER, y utilizar la característica de generación de código de Visible Analyst.”

16

**FIGURA E16.3**

Pantalla del depósito para el módulo PRODUCE SOFTWARE LINES.

Anna y Chip proceden a trabajar con Visible Analyst para asegurarse de que se hayan definido todos los elementos de SOFTWARE MASTER. Anna hace clic en **Repository** y en **Generate Database Schema**. Seleccionan el diagrama COMPUTER SYSTEM y le dan el mismo nombre al esquema. Se genera el esquema completo para el sistema de cómputo. En la figura E16.4 se ilustra una parte del código que se generó.

“Voy a copiar una parte del esquema para trabajar sólo con el SOFTWARE MASTER”, comenta Anna a Chip. Anna copia el código SQL generado para el archivo SOFTWARE MASTER. El siguiente paso es crear una consulta en blanco en Microsoft Access. Anna ejecuta Microsoft Access y crea la consulta. A continuación hace clic en el botón SQL y pega el archivo SOFTWARE MASTER en la ventana de SQL.

“Tengo que cambiar el nombre de la tabla por el de SOFTWARE y cambiar el tipo de consulta a Make Table”, continúa Anna. Le da el nombre SOFTWARE a la nueva tabla. Anna hace clic en el botón **Run Query** y cierra la consulta.

“¿Qué ocurrió?”, pregunta Chip desconcertado. “No veo ninguna salida.”

16

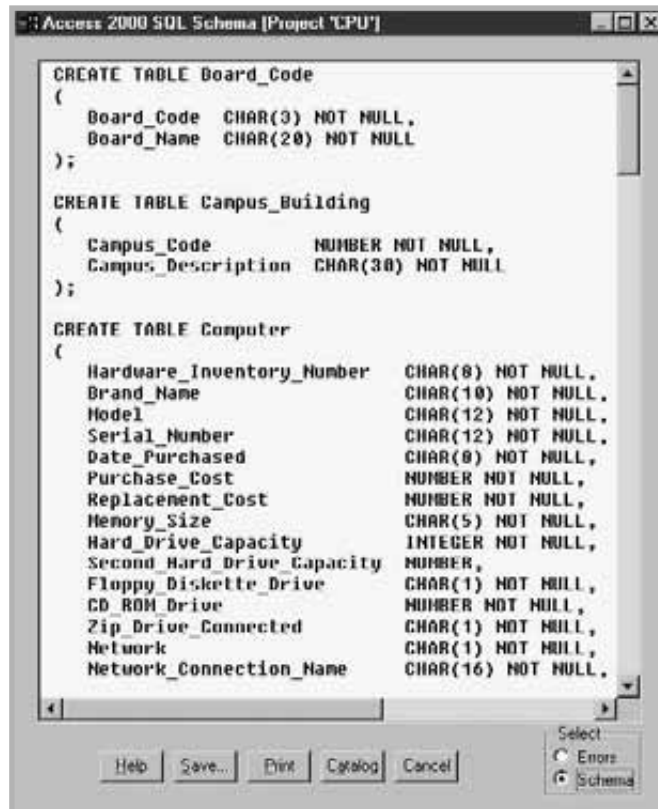


FIGURA E16.4

Ejemplo de generación de código.

Anna hace clic en el botón **Tables**. “¡Dale un vistazo a nuestra nueva tabla!”, exclama Anna. Ella hace clic en la tabla **SOFTWARE** y en la vista diseño. “Aquí tienes nuestra estructura de Visible Analyst, implementada en Microsoft Access.”

“Ahora se ve fenomenal”, dice Chip con una amplia sonrisa.




Los analistas siguen generando tablas hasta que finalizan el diseño.

“Creo que podemos dejarle el resto a los programadores”, comenta Anna. “Haríamos mejor en comenzar el desarrollo de los planes de prueba para cada programa.”

Los planes de prueba contienen detalles acerca de cómo determinar si los programas funcionan correctamente, y se los envían a Mack y Dee, quienes crearán los datos para las pruebas. En cada archivo de prueba que se utiliza en los programas por lotes se incluyen datos válidos e inválidos. Lo mismo ocurre con los sistemas interactivos, excepto que los datos de prueba se escriben en formas semejantes al diseño de las pantallas. Una vez que Mack termina de probar sus programas y se convence de que funcionan correctamente, reta a Dee a que encuentre algún error en los programas. A su vez, Dee le pide a Mack que pruebe sus programas en una especie de competencia amistosa. Ambos están conscientes de que los programadores no siempre detectan sus propios errores, porque conocen tanto sus propios programas que podrían ignorar errores sutiles de lógica.

16


EJERCICIOS

-  E-1. Vea el diagrama de estructura PRODUCE SOFTWARE CROSS-REF REP. Haga doble clic en algunos módulos para ver sus entradas en el depósito.
-  E-2. Modifique el diagrama de estructura PRODUCE HARDWARE INVESTMENT RPT. Agregue la función PRINT INVESTMENT LINE en el rectángulo vacío que se proporciona. PRINT HEADING LINES y WRITE REPORT LINE están subordinados a este módulo. Describa cada función en el depósito.
-  E-3. Modifique el diagrama de estructura del archivo CHANGE COMPUTER. Incluya el símbolo de bucle y agregue los siguientes módulos subordinados al módulo 160, CHANGE COMPUTER RECORD (también vea abajo):


- A. SHOW CHANGE DISPLAY
- B. ACCEPT COMPUTER CHANGES
- C. VALIDATE CHANGES
- D. DISPLAY ERROR MESSAGE
- E. CONFIRM CHANGES

Los siguientes módulos deben subordinarse al módulo 220, PUT COMPUTER RECORD:

- A. FORMAT COMPUTER RECORD
- B. REWRITE COMPUTER RECORD

-  E-4. Modifique el diagrama de estructura ADD SOFTWARE RECORD agregando un símbolo de bucle y acoplando las conexiones. El siguiente acoplamiento se debe colocar en la línea de conexión arriba de cada módulo (también vea abajo):

- | | |
|----------------------|--------------------------------|
| A. Módulo: | DISPLAY ADD SOFTWARE SCREEN |
| Pasado hacia arriba: | ADD SOFTWARE SCREEN |
| B. Módulo: | ACCEPT ADD SOFTWARE SCREEN |
| Pasado hacia arriba: | EXIT INDICATOR (Control) |
| | ADD SOFTWARE SCREEN DATA |
| C. Módulo: | VALIDATE ADD SOFTWARE DATA |
| Pasado hacia abajo: | ADD SOFTWARE SCREEN DATA |
| Pasado hacia arriba: | CANCEL TRANSACTION (Control) |
| | VALID ADD SOFTWARE DATA |
| D. Módulo: | READ SOFTWARE RECORD |
| Pasado hacia abajo: | SOFTWARE INVENTORY NUMBER |
| Pasado hacia arriba: | RECORD FOUND (Control) |
| E. Módulo: | VALIDATE HARDWARE REQUIREMENTS |
| Pasado hacia abajo: | ADD SOFTWARE SCREEN DATA |
| Pasado hacia arriba: | VALID DATA (Control) |
| | ERROR MESSAGE |
| F. Módulo: | DISPLAY ERROR MESSAGE |
| Pasado hacia abajo: | ERROR MESSAGE |
| G. Módulo: | PUT NEW SOFTWARE RECORD |
| Pasado hacia abajo: | VALID ADD SOFTWARE DATA |

 Los ejercicios precedidos por un icono Web indican que en el sitio Web del libro hay material de valor agregado. Los estudiantes pueden descargar una base de datos de Microsoft Access que pueden utilizar para completar los ejercicios.

16

- H. Módulo: FORMAT SOFTWARE RECORD
 - Pasado hacia abajo: VALID ADD SOFTWARE DATA
 - Pasado hacia arriba: FORMATTED SOFTWARE RECORD
- I. Módulo: WRITE SOFTWARE RECORD
 - Pasado hacia abajo: FORMATTED SOFTWARE RECORD



E-5. Elabore el diagrama de estructura PRINT PROBLEM MACHINE REPORT. A continuación se da un esquema de los módulos, con cada módulo subordinado sangrado:

```

PRINT PROBLEM MACHINE REPORT
  PRINT PROBLEM MACHINE LINES
    READ MACHINE RECORD
    DETERMINE PROBLEM MACHINE
    PRINT PROBLEM MACHINE LINE
      PRINT HEADING LINES
      WRITE REPORT LINE
    PRINT FINAL REPORT LINES
      WRITE REPORT LINE
  
```



E-6. Elabore el diagrama de estructura CHANGE SOFTWARE RECORD. Los módulos del programa se muestran con sus módulos subordinados sangrados.

```

CHANGE SOFTWARE FILE
  CHANGE SOFTWARE RECORDS
    GET SOFTWARE RECORD
      DISPLAY SOFTWARE ID SCREEN
      ACCEPT SOFTWARE ID SCREEN
      FIND SOFTWARE RECORD
      DISPLAY ERROR LINE
    OBTAIN SOFTWARE CHANGES
      DISPLAY CHANGE SCREEN
      ACCEPT SOFTWARE CHANGES
      VALIDATE CHANGES
      DISPLAY ERROR LINE
    PUT SOFTWARE RECORD
      FORMAT SOFTWARE RECORD
      REWRITE SOFTWARE RECORD
  
```



E-7. Elabore el diagrama de estructura SOFTWARE DETAILS INQUIRY. Los módulos se enlistan con sus módulos subordinados sangrados.

```

INQUIRE SOFTWARE DETAILS
  INQUIRE SOFTWARE RECORD
    GET SOFTWARE RECORD
      DISPLAY SOFTWARE ID SCREEN
      ACCEPT SOFTWARE ID SCREEN
      FIND SOFTWARE RECORD
      DISPLAY ERROR LINE
    DISPLAY INQUIRY SCREEN
      FORMAT SOFTWARE INQUIRY SCREEN
      DISPLAY SOFTWARE INQUIRY SCREEN
  
```

16



E-8. Vea el diagrama de flujo del sistema ADD COMPUTER.



E-9. Modifique el flujo del sistema ADD SOFTWARE. Agregue los siguientes rectángulos del programa abajo del proceso manual INSTALL SOFTWARE. Incluya los archivos e informes de entrada y salida especificados para cada programa.

Programa:	UPDATE SOFTWARE RELATIONAL FILE
Entrada:	UPDATE SOFTWARE INSTALLATION LIST, documento UPDATE INSTALLED SOFTWARE SCREEN, pantalla
Salida:	SOFTWARE RELATIONAL FILE, disco INSTALLED SOFTWARE TRANSACTION, disco
Programa:	PRINT USER NOTIFICATION REPORT
Entrada:	INSTALLED SOFTWARE TRANSACTION, disco
Salida:	USER NOTIFICATION REPORT, informe



- E-10. Elabore el diagrama de flujo del sistema ADD STAFF. Hay dos programas: ADD STAFF y PRINT NEW STAFF LIST. La entrada para el programa ADD STAFF es un listado NEW STAFF y un pantalla de entrada ADD NEW STAFF. El archivo STAFF MASTER se actualiza y se produce un nuevo archivo NEW STAFF LOG. Este último archivo es entrada para el programa PRINT NEW STAFF LIST, que produce el informe NEW STAFF LIST.
- E-11. Diseñe datos de prueba en papel para probar el programa ADD COMPUTER. Utilice Microsoft Access para probar la pantalla. Anote cualquier inconsistencia.
- E-12. Diseñe datos de prueba y resultados previstos para el programa ADD SOFTWARE. Utilice Microsoft Access para probar la pantalla y anote si los resultados se apegaron a sus predicciones.
- E-13. Diseñe datos de prueba y resultados previstos para el programa ADD TRAINING CLASS. Utilice Microsoft Access para probar la pantalla y anote si los resultados se apegaron a sus predicciones.
- E-14. Diseñe datos de prueba en papel para probar el programa CHANGE SOFTWARE EXPERT. Utilice Microsoft Access para probar la pantalla. Anote cualquier inconsistencia.