UNIDAD 6: DISEÑO

Qué es el diseño?. Objetivos del diseño de un sistema. Características de un buen diseño. Técnicas de diseño. Evaluación y validación del diseño. Diseño de las entradas. Diseño de las salidas. Diseño de la interfaz de usuario. Reglas de oro. Definición de objetos y acciones de la interfaz. Definición de la arquitectura del sistema. Definición de estándares de diseño. Diseño de archivos y base de datos. Diseño de los procesos. PARADIGMA ESTRUCTURADO: Diseño modular:

Descripción y objetivos

El objetivo del proceso de Diseño del Sistema Sistema de Información Información (DSI) es la definición de la arquitectura del sistema y del entorno tecnológico que le va a dar soporte, junto con la especificación detallada de los componentes del sistema de información.

Diseño de las SALIDAS

El diseño de software agrupa el conjunto de principios, conceptos y prácticas que llevan al desarrollo de un sistema o producto de alta calidad. Los principios de diseño establecen una filosofía general que guía el trabajo de diseño que debe ejecutarse. Deben entenderse los conceptos de diseño antes de aplicar la mecánica de éste, y la práctica del diseño en sí lleva a la creación de distintas representaciones del software que sirve como guía para la actividad de construcción que siga.

El diseño es crucial para el éxito de la ingeniería de software. A principios de la década de 1990, Mitch Kapor, creador de Lotus 1-2-3, publicó en *Dr. Dobbs Journal* un "manifiesto del diseño de software". Decía lo siguiente:

> ¿Qué es el diseño? Es donde se está con un pie en dos mundos —el de la tecnología y el de las personas y los propósitos humanos— que tratan de unificarse...
>
> Vitruvio, romano crítico de arquitectura, afirmaba que los edificios bien diseñados eran aquellos que tenían resistencia, funcionalidad y belleza. Lo mismo se aplica al buen software. *Resistencia:* un programa no debe tener ningún error que impida su funcionamiento. *Funcionalidad:* un programa debe se apropiado para los fines que persigue. *Belleza:* la experiencia de usar el programa debe ser placentera. Éstos son los comienzos de una teoría del diseño de software.

El objetivo del diseño es producir un modelo o representación que tenga resistencia, funcionalidad y belleza. Para lograrlo, debe practicarse la diversificación y luego la convergencia. Belady

UNA MIRADA RÁPIDA

¿Qué es? El diseño es lo que casi todo ingeniero quiere hacer. Es el lugar en el que las reglas de la creatividad —los requerimientos de los participantes, las necesidades del negocio y las consideraciones técnicas— se unen para formular un producto o sistema. El diseño crea una representación o modelo del software, pero, a diferencia del modelo de los requerimientos (que se centra en describir los datos que se necesitan, la función y el comportamiento), el modelo de diseño proporciona detalles sobre arquitectura del software, estructuras de datos, interfaces y componentes que se necesitan para implementar el sistema.

¿Quién lo hace? Ingenieros de software llevan a cabo cada una de las tareas del diseño.

¿Por qué es importante? El diseño permite modelar el sistema o producto que se va a construir. Este modelo se evalúa respecto de la calidad y su mejora antes de generar código; después, se efectúan pruebas y se involucra a muchos usuarios finales. El diseño es el lugar en el que se establece la calidad del software.

¿Cuáles son los pasos? El diseño representa al software de varias maneras. En primer lugar, debe representarse la arquitectura del sistema o producto. Después se modelan las interfaces que conectan al software con los usuarios finales, con otros sistemas y dispositivos, y con sus propios componentes constitutivos. Por último, se diseñan los componentes del software que se utilizan para construir el sistema. Cada una de estas perspectivas representa una acción de diseño distinta, pero todas deben apegarse a un conjunto básico de conceptos de diseño que guíe el trabajo de producción de software.

¿Cuál es el producto final? El trabajo principal que se produce durante el diseño del software es un modelo de diseño que agrupa las representaciones arquitectónicas, interfaces en el nivel de componente y despliegue.

¿Cómo me aseguro de que lo hice bien? El modelo de diseño es evaluado por el equipo de software en un esfuerzo por determinar si contiene errores, inconsistencias u omisiones, si existen mejores alternativas y si es posible implementar el modelo dentro de las restricciones, plazo y costo que se hayan establecido.

8.1 DISEÑO EN EL CONTEXTO DE LA INGENIERÍA DE SOFTWARE

Cita:

"El milagro más común de la ingeniería de software es la transición del análisis al diseño y de éste al código."

Richard Due'

El diseño de software se ubica en el área técnica de la ingeniería de software y se aplica sin importar el modelo del proceso que se utilice. El diseño del software comienza una vez que se han analizado y modelado los requerimientos, es la última acción de la ingeniería de software dentro de la actividad de modelado y prepara la etapa de **construcción** (generación y prueba de código).

Cada uno de los elementos del modelo de requerimientos (capítulos 6 y 7) proporciona información necesaria para crear los cuatro modelos de diseño necesarios para la especificación completa del diseño. En la figura 8.1 se ilustra el flujo de la información durante el diseño del software. El trabajo de diseño es alimentado por el modelo de requerimientos, manifestado por elementos basados en el escenario, en la clase, orientados al flujo, y del comportamiento. El empleo de la notación y de los métodos de diseño estudiados en los últimos capítulos produce diseños de los datos o clases, de la arquitectura, de la interfaz y de los componentes.

CONSEJO

El diseño del software siempre debe comenzar con el análisis de los datos, pues son el fundamento de todos los demás elementos del diseño. Una vez obtenido el fundamento, se obtiene la arquitectura. Sólo entonces deben realizarse otros trabajos del diseño.

El diseño de datos o clases transforma los modelos de clases (capítulo 6) en realizaciones de clases de diseño y en las estructuras de datos que se requieren para implementar el software. Los objetos y relaciones definidos en el diagrama CRC y el contenido detallado de los datos ilustrado por los atributos de clase y otros tipos de notación dan la base para el diseño de los datos. Parte del diseño de clase puede llevarse a cabo junto con el diseño de la arquitectura del software. Un diseño más detallado de las clases tiene lugar cuando se diseña cada componente del software.

El diseño de la arquitectura define la relación entre los elementos principales de la estructura del software, los estilos y patrones de diseño de la arquitectura que pueden usarse para alcanzar

los requerimientos definidos por el sistema y las restricciones que afectan la forma en la que se implementa la arquitectura [Sha96]. La representación del diseño de la arquitectura —el marco de un sistema basado en computadora— se obtiene del modelo de los requerimientos.

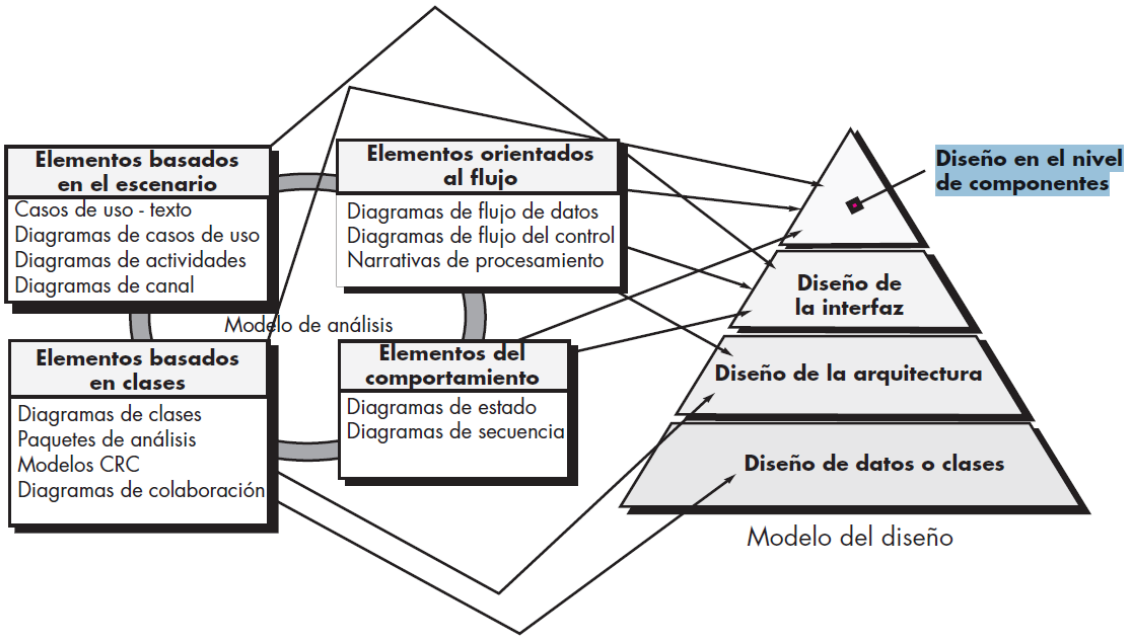
El diseño de la interfaz describe la forma en la que el software se comunica con los sistemas que interactúan con él y con los humanos que lo utilizan. Una interfaz implica un flujo de información (por ejemplo, datos o control) y un tipo específico de comportamiento. Entonces, los modelos de escenarios de uso y de comportamiento dan mucha de la información requerida para diseñar la interfaz.

El diseño en el nivel de componente transforma los elementos estructurales de la arquitectura del software en una descripción de sus componentes en cuanto a procedimiento. La información obtenida a partir de los modelos basados en clase, flujo y comportamiento sirve como la base para diseñar los componentes.

Durante el diseño se toman decisiones que en última instancia afectarán al éxito de la construcción del software y, de igual importancia, a la facilidad con la que puede darse mantenimiento al software. Pero, ¿por qué es tan importante el diseño?

La importancia del diseño del software se resume en una palabra: *calidad*. El diseño es el sitio en el que se introduce calidad en la ingeniería de software. Da representaciones del software que pueden evaluarse en su calidad. Es la única manera de traducir con exactitud a un producto o sistema terminado los requerimientos de los participantes. Es el fundamento de toda la ingeniería de software y de las actividades que dan el apoyo que sigue. Sin diseño se corre el riesgo de obtener un sistema inestable, que falle cuando se hagan cambios pequeños, o uno que sea difícil de someter a prueba, o en el que no sea posible evaluar la calidad hasta que sea demasiado tarde en el proceso de software, cuando no queda mucho tiempo y ya se ha gastado mucho dinero.

Traducción del modelo de requerimientos al modelo de diseño



8.2 El proceso de diseño

El diseño de software es un proceso iterativo por medio del cual se traducen los requerimientos en un "plano" para construir el software. Al principio, el plano ilustra una visión holística del software. Es decir, el diseño se representa en un nivel alto de abstracción, en el que se rastrea directamente el objetivo específico del sistema y los requerimientos más detallados de datos, funcionamiento y comportamiento. A medida que tienen lugar las iteraciones del diseño, las mejoras posteriores conducen a niveles menores de abstracción. Éstos también pueden rastrearse hasta los requerimientos, pero la conexión es más sutil.

Lineamientos de la calidad. A fin de evaluar la calidad de una representación del diseño, usted y otros miembros del equipo de software deben establecer los criterios técnicos de un buen diseño. En la sección 8.3 se estudian conceptos de diseño que también sirven como criterios de calidad del software. En este momento, considere los siguientes lineamientos para el diseño:

1. Debe tener una arquitectura que 1) se haya creado con el empleo de estilos o patrones arquitectónicos reconocibles, 2) esté compuesta de componentes con buenas características de diseño (éstas se analizan más adelante, en este capítulo), y 3) se implementen en forma evolutiva,[2] de modo que faciliten la implementación y las pruebas.
2. Debe ser modular, es decir, el software debe estar dividido de manera lógica en elementos o subsistemas.
3. Debe contener distintas representaciones de datos, arquitectura, interfaces y componentes.
4. Debe conducir a estructuras de datos apropiadas para las clases que se van a implementar y que surjan de patrones reconocibles de datos.
5. Debe llevar a componentes que tengan características funcionales independientes.
6. Debe conducir a interfaces que reduzcan la complejidad de las conexiones entre los componentes y el ambiente externo.
7. Debe obtenerse con el empleo de un método repetible motivado por la información obtenida durante el análisis de los requerimientos del software.
8. Debe representarse con una notación que comunique con eficacia su significado.

Estos lineamientos de diseño no se logran por azar. Se consiguen con la aplicación de los principios de diseño fundamentales, una metodología sistemática y con revisión.

Atributos de la calidad. Hewlett-Packard [Gra87] desarrolló un conjunto de atributos de la calidad del software a los que se dio el acrónimo FURPS: funcionalidad, usabilidad, confiabilidad, rendimiento y mantenibilidad. Los atributos de calidad FURPS representan el objetivo de todo diseño de software:

- La *funcionalidad* se califica de acuerdo con el conjunto de características y capacidades del programa, la generalidad de las funciones que se entregan y la seguridad general del sistema.
- La *usabilidad* se evalúa tomando en cuenta factores humanos (véase el capítulo 11), la estética general, la consistencia y la documentación.
- La *confiabilidad* se evalúa con la medición de la frecuencia y gravedad de las fallas, la exactitud de los resultados que salen, el tiempo medio para que ocurra una falla (TMPF), la capacidad de recuperación ante ésta y lo predecible del programa.
- El *rendimiento* se mide con base en la velocidad de procesamiento, el tiempo de respuesta, el uso de recursos, el conjunto y la eficiencia.
- La *mantenibilidad* combina la capacidad del programa para ser ampliable (extensibilidad), adaptable y servicial (estos tres atributos se denotan con un término más común: *mantenibilidad*), y además que pueda probarse, ser compatible y configurable (capacidad de organizar y controlar los elementos de la configuración del software, véase el capítulo 22) y que cuente con la facilidad para instalarse en el sistema y para que se detecten los problemas.

No todo atributo de la calidad del software se pondera por igual al diseñarlo. Una aplicación tal vez se aboque a lo funcional con énfasis en la seguridad. Otra quizá busque rendimiento con la mira puesta en la velocidad de procesamiento. En una tercera se persigue la confiabilidad. Sin importar la ponderación, es importante observar que estos atributos de la calidad deben tomarse en cuenta cuando comienza el diseño, *no* cuando haya terminado éste y la construcción se encuentre en marcha.

8.3 Conceptos de diseño

8.3.1 Abstracción

Cuando se considera una solución modular para cualquier problema, es posible plantear muchos niveles de abstracción. En el más elevado se enuncia una solución en términos gruesos con el uso del lenguaje del ambiente del problema. En niveles más bajos de abstracción se da la descripción más detallada de la solución. La terminología orientada al problema se acopla con la que se orienta a la implementación, en un esfuerzo por enunciar la solución. Por último, en el nivel de abstracción más bajo se plantea la solución, de modo que pueda implementarse directamente.

Cuando se desarrollan niveles de abstracción distintos, se trabaja para crear abstracciones tanto de procedimiento como de datos. Una *abstracción de procedimiento* es una secuencia de instrucciones que tienen una función específica y limitada. El nombre de la abstracción de procedimiento implica estas funciones, pero se omiten detalles específicos. Un ejemplo de esto sería la palabra *abrir*, en el caso de una puerta. *Abrir* implica una secuencia larga de pasos del procedimiento (caminar hacia la puerta, llegar y tomar el picaporte, girar éste y jalar la puerta, retroceder para que la puerta se abra, etcétera).[5]

Una *abstracción de datos* es un conjunto de éstos con nombre que describe a un objeto de datos. En el contexto de la abstracción de procedimiento *abrir*, puede definirse una abstracción de datos llamada **puerta**. Como cualquier objeto de datos, la abstracción de datos para **puerta** agruparía un conjunto de atributos que describirían la puerta (tipo, dirección del abatimiento, mecanismo de apertura, peso, dimensiones, etc.). Se concluye que la abstracción de procedimiento *abrir* usaría información contenida en los atributos de la abstracción de datos **puerta**.

8.3.2 Arquitectura

La *arquitectura del software* alude a "la estructura general de éste y a las formas en las que ésta da integridad conceptual a un sistema" [Sha95a]. En su forma más sencilla, la arquitectura es la estructura de organización de los componentes de un programa (módulos), la forma en la que éstos interactúan y la estructura de datos que utilizan. Sin embargo, en un sentido más amplio, los componentes se generalizan para que representen los elementos de un sistema grande y sus interacciones.

Una meta del diseño del software es obtener una aproximación arquitectónica de un sistema. Ésta sirve como estructura a partir de la cual se realizan las actividades de diseño más detalladas. Un conjunto de patrones arquitectónicos permite que el ingeniero de software resuelva problemas de diseño comunes.

Shaw y Garlan [Sha95a] describen un conjunto de propiedades que deben especificarse como parte del diseño de la arquitectura:

> **Propiedades estructurales.** Este aspecto de la representación del diseño arquitectónico define los componentes de un sistema (módulos, objetos, filtros, etc.) y la manera en la que están agrupados e interactúan unos con otros. Por ejemplo, los objetos se agrupan para que encapsulen tanto datos como el procedimiento que los manipula e interactúen invocando métodos.
>
> **Propiedades extrafuncionales.** La descripción del diseño arquitectónico debe abordar la forma en la que la arquitectura del diseño satisface los requerimientos de desempeño, capacidad, confiabilidad, seguridad y adaptabilidad, así como otras características del sistema.
>
> **Familias de sistemas relacionados.** El diseño arquitectónico debe basarse en patrones repetibles que es común encontrar en el diseño de familias de sistemas similares. En esencia, el diseño debe tener la capacidad de reutilizar bloques de construcción arquitectónica.

Dada la especificación de estas propiedades, el diseño arquitectónico se representa con el uso de uno o más de varios modelos diferentes [Gar95]. Los *modelos estructurales* representan la arquitectura como un conjunto organizado de componentes del programa. Los *modelos de marco* aumentan el nivel de abstracción del diseño, al tratar de identificar patrones de diseño arquitectónico repetibles que se encuentran en tipos similares de aplicaciones. Los *modelos dinámicos* abordan los aspectos estructurales de la arquitectura del programa e indican cómo cambia la

8.3.3 Patrones

Brad Appleton define un *patrón de diseño* de la manera siguiente: "Es una mezcla con nombre propio de puntos de vista que contienen la esencia de una solución demostrada para un problema recurrente dentro de cierto contexto de necesidades en competencia" [App00]. Dicho de otra manera, un patrón de diseño describe una estructura de diseño que resuelve un problema particular del diseño dentro de un contexto específico y entre "fuerzas" que afectan la manera en la que se aplica y en la que se utiliza dicho patrón.

El objetivo de cada patrón de diseño es proporcionar una descripción que permita a un diseñador determinar 1) si el patrón es aplicable al trabajo en cuestión, 2) si puede volverse a usar (con lo que se ahorra tiempo de diseño) y 3) si sirve como guía para desarrollar un patrón distinto en funciones o estructura. En el capítulo 12 se estudian los patrones de diseño.

8.3.4 División de problemas

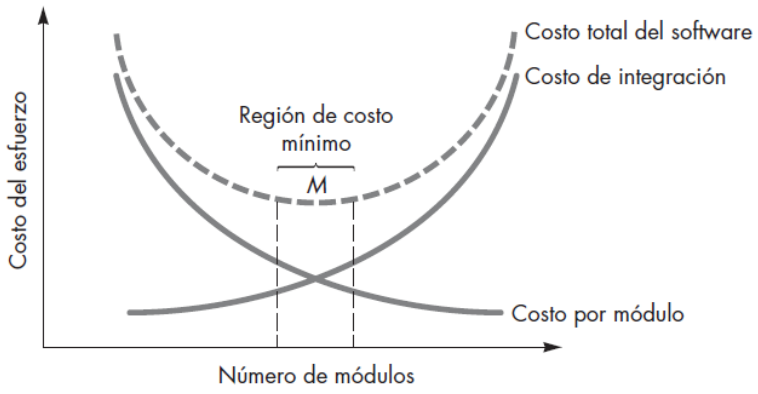
La *división de problemas* es un concepto de diseño que sugiere que cualquier problema complejo puede manejarse con más facilidad si se subdivide en elementos susceptibles de resolverse u optimizarse de manera independiente. Un *problema* es una característica o comportamiento que se especifica en el modelo de los requerimientos para el software. Al separar un problema en sus piezas más pequeñas y por ello más manejables, se requiere menos esfuerzo y tiempo para resolverlo.

Si para dos problemas, p_1 y p_2, la complejidad que se percibe para p_1 es mayor que la percibida para p_2, entonces se concluye que el esfuerzo requerido para resolver p_1 es mayor que el necesario para resolver p_2. Como caso general, este resultado es intuitivamente obvio. Lleva más tiempo resolver un problema difícil.

También se concluye que cuando se combinan dos problemas, con frecuencia la complejidad percibida es mayor que la suma de la complejidad tomada por separado. Esto lleva a la estrategia de divide y vencerás, pues es más fácil resolver un problema complejo si se divide en elementos manejables. Esto tiene implicaciones importantes en relación con la modularidad del software.

8.3.5 Modularidad

La modularidad es la manifestación más común de la división de problemas. El software se divide en componentes con nombres distintos y abordables por separado, en ocasiones llamados *módulos*, que se integran para satisfacer los requerimientos del problema.



Se ha dicho que "la modularidad es el único atributo del software que permite que un programa sea manejable en lo intelectual" [Mye78]. El software monolítico (un programa grande compuesto de un solo módulo) no es fácil de entender para un ingeniero de software. El número de trayectorias de control, alcance de referencia, número de variables y complejidad general haría que comprenderlo fuera casi imposible. En función de las circunstancias, el diseño debe descomponerse en muchos módulos con la esperanza de que sea más fácil entenderlos y, en consecuencia, reducir el costo requerido para elaborar el software.

Según el punto de vista de la división de problemas, sería posible concluir que si el software se dividiera en forma indefinida, el esfuerzo requerido para desarrollarlo ¡sería despreciable por pequeño! Desafortunadamente, hay otras fuerzas que entran en juego y que hacen que esta conclusión sea (tristemente) inválida. De acuerdo con la figura 8.2, el esfuerzo (costo) de desarrollar un módulo individual de software disminuye conforme aumenta el número total de módulos. Dado el mismo conjunto de requerimientos, tener más módulos significa tamaños individuales más pequeños. Sin embargo, a medida que se incrementa el número de módulos, el esfuerzo (costo) asociado con su integración también aumenta. Estas características llevan a una curva de costo total como la que se muestra en la figura. Existe un número, M, de módulos que arrojarían el mínimo costo de desarrollo, pero no se dispone de las herramientas necesarias para predecir M con exactitud.

Las curvas que aparecen en la figura 8.2 constituyen una guía útil al considerar la modularidad. Deben hacerse módulos, pero con cuidado para permanecer en la cercanía de M. Debe evitarse hacer pocos o muchos módulos. Pero, ¿cómo saber cuál es la cercanía de M? ¿Cuán modular debe hacerse el software? Las respuestas a estas preguntas requieren la comprensión de otros conceptos de diseño que se analizan más adelante en este capítulo.

Debe hacerse un diseño (y el programa resultante) con módulos, de manera que el desarrollo pueda planearse con más facilidad, que sea posible definir y desarrollar los incrementos del software, que los cambios se realicen con más facilidad, que las pruebas y la depuración se efectúen con mayor eficiencia y que el mantenimiento a largo plazo se lleve a cabo sin efectos colaterales de importancia.

8.3.6 Ocultamiento de información

El concepto de modularidad lleva a una pregunta fundamental: "¿Cómo descomponer una solución de software para obtener el mejor conjunto de módulos?" El principio del ocultamiento de información sugiere que los módulos se "caractericen por decisiones de diseño que se oculten (cada una) de las demás". En otras palabras, deben especificarse y diseñarse módulos, de forma que la información (algoritmos y datos) contenida en un módulo sea inaccesible para los que no necesiten de ella.

8.3.7 Independencia funcional

El concepto de independencia funcional es resultado directo de la separación de problemas y de los conceptos de abstracción y ocultamiento de información. En escritos cruciales sobre el diseño de software, Wirth [Wir71] y Parnas [Par72] mencionan técnicas de mejora que promueven la independencia modular. Los trabajos posteriores de Stevens, Myers y Constantine [Ste74] dan solidez al concepto.

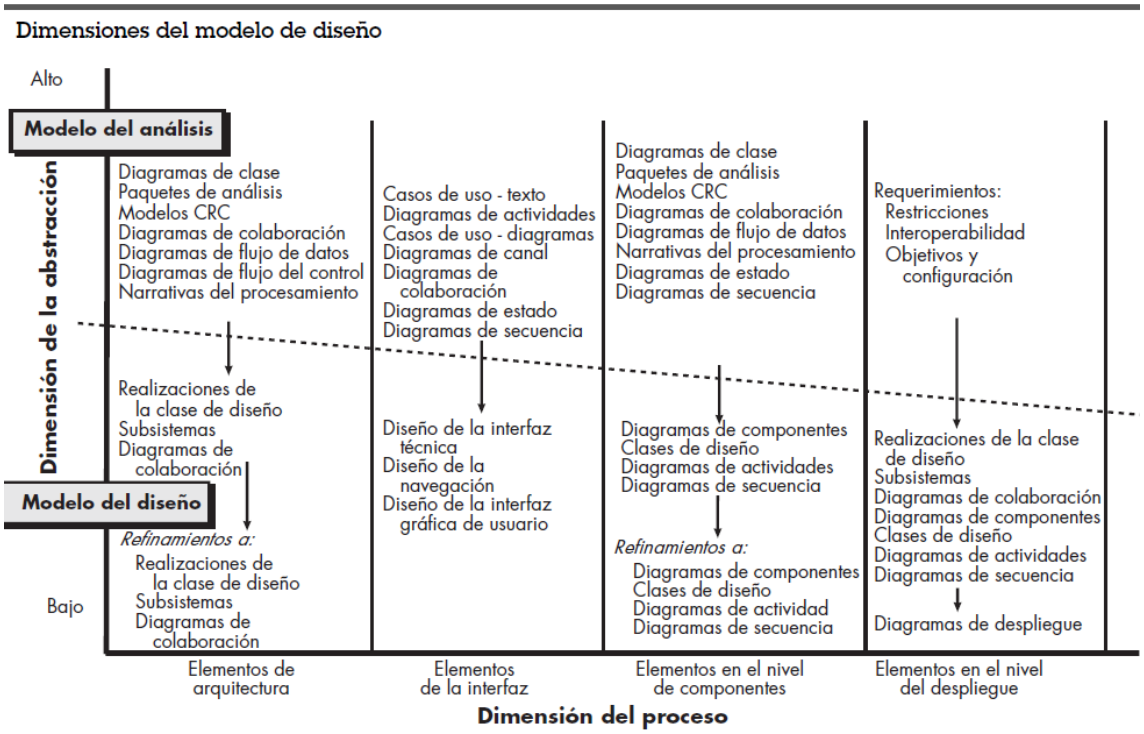
La independencia funcional se logra desarrollando módulos con funciones "miopes" que tengan "aversión" a la interacción excesiva con otros módulos. Dicho de otro modo, debe diseñarse software de manera que cada módulo resuelva un subconjunto específico de requerimientos y tenga una interfaz sencilla cuando se vea desde otras partes de la estructura del programa. Es lógico preguntar por qué es importante la independencia.

El software con modularidad eficaz, es decir, con módulos independientes, es más fácil de desarrollar porque su función se subdivide y las interfaces se simplifican (cuando el desarrollo es efectuado por un equipo hay que considerar las ramificaciones). Los módulos independientes son más fáciles de mantener (y probar) debido a que los efectos secundarios causados por el diseño o por la modificación del código son limitados, se reduce la propagación del error y es posible obtener módulos reutilizables. En resumen, la independencia funcional es una clave para el buen diseño y éste es la clave de la calidad del software.

La independencia se evalúa con el uso de dos criterios cualitativos: la cohesión y el acoplamiento. La *cohesión* es un indicador de la fortaleza relativa funcional de un módulo. El *acoplamiento* lo es de la independencia relativa entre módulos.

La cohesión es una extensión natural del concepto de ocultamiento de información descrito en la sección 8.3.6. Un módulo cohesivo ejecuta una sola tarea, por lo que requiere interactuar poco con otros componentes en otras partes del programa. En pocas palabras, un módulo cohesivo debe (idealmente) hacer sólo una cosa. Aunque siempre debe tratarse de lograr mucha cohesión (por ejemplo, una sola tarea), con frecuencia es necesario y aconsejable hacer que un componente de software realice funciones múltiples. Sin embargo, para lograr un buen diseño hay que evitar los componentes "esquizofrénicos" (módulos que llevan a cabo funciones no relacionadas).

El acoplamiento es una indicación de la interconexión entre módulos en una estructura de software, y depende de la complejidad de la interfaz entre módulos, del grado en el que se entra o se hace referencia a un módulo y de qué datos pasan a través de la interfaz. En el diseño de software, debe buscarse el mínimo acoplamiento posible. La conectividad simple entre módulos da como resultado un software que es más fácil de entender y menos propenso al "efecto de oleaje" [Ste74], ocasionado cuando ocurren errores en un sitio y se propagan por todo el sistema.



8.4.1 Elementos del diseño de datos

Igual que otras actividades de la ingeniería de software, el diseño de datos (en ocasiones denominado *arquitectura de datos*) crea un modelo de datos o información que se representa en un nivel de abstracción elevado (el punto de vista del usuario de los datos). Este modelo de los datos se refina después en forma progresiva hacia representaciones más específicas de la implementación que puedan ser procesadas por el sistema basado en computadora. En muchas aplicaciones de software, la arquitectura de los datos tendrá una influencia profunda en la arquitectura del software que debe procesarlo.

La estructura de los datos siempre ha sido parte importante del diseño de software. En el nivel de componentes del programa, del diseño de las estructuras de datos y de los algoritmos requeridos para manipularlos, es esencial la creación de aplicaciones de alta calidad. En el nivel de la aplicación, la traducción de un modelo de datos (obtenido como parte de la ingeniería de los requerimientos) a una base de datos es crucial para lograr los objetivos de negocios de un sistema. En el nivel de negocios, el conjunto de información almacenada en bases de datos incompatibles y reorganizados en un "data warehouse" permite la minería de datos o descubrimiento de conocimiento que tiene un efecto en el éxito del negocio en sí. En cada caso, el diseño de los datos juega un papel importante. El diseño de datos se estudia con más detalle en el capítulo 9.

8.4.2 Elementos del diseño arquitectónico

El *diseño de la arquitectura* del software es el equivalente del plano de una casa. Éste ilustra la distribución general de las habitaciones, su tamaño, forma y relaciones entre ellas, así como las puertas y ventanas que permiten el movimiento entre los cuartos. El plano da una visión general de la casa. Los elementos del diseño de la arquitectura dan la visión general del software.

El modelo arquitectónico [Sha96] proviene de tres fuentes: 1) información sobre el dominio de la aplicación del software que se va a elaborar, 2) los elementos específicos del modelo de requerimientos, tales como diagramas de flujo de datos o clases de análisis, sus relaciones y colaboraciones para el problema en cuestión y 3) la disponibilidad de estilos arquitectónicos (capítulo 9) y sus patrones (capítulo 12).

Por lo general, el elemento de diseño arquitectónico se ilustra como un conjunto de sistemas interconectados, con frecuencia obtenidos de paquetes de análisis dentro del modelo de requerimientos. Cada subsistema puede tener su propia arquitectura (por ejemplo, la interfaz gráfica de usuario puede estar estructurada de acuerdo con un estilo de arquitectura preexistente para interfaces de usuario). En el capítulo 9 se presentan técnicas para obtener elementos específicos del modelo arquitectónico.

8.4.3 Elementos de diseño de la interfaz

El diseño de la interfaz para el software es análogo al conjunto de trazos (y especificaciones) detalladas para las puertas, ventanas e instalaciones de una casa. Tales dibujos ilustran el tamaño y forma de puertas y ventanas, la manera en la que operan, la forma en la que llegan las instalaciones de servicios (agua, electricidad, gas, teléfono, etc.) a la vivienda y se distribuyen entre las habitaciones indicadas en el plano. Indican dónde está el timbre de la puerta, si se usará un intercomunicador para anunciar la presencia de un visitante y cómo se va a instalar el sistema de seguridad. En esencia, los planos (y especificaciones) detallados para las puertas, ventanas e instalaciones externas nos dicen cómo fluyen las cosas y la información hacia dentro y fuera de la casa y dentro de los cuartos que forman parte del plano. Los elementos de diseño de la interfaz del software permiten que la información fluya hacia dentro y afuera del sistema, y cómo están comunicados los componentes que son parte de la arquitectura.

Hay tres elementos importantes del diseño de la interfaz: 1) la interfaz de usuario (IU), 2) las interfaces externas que tienen que ver con otros sistemas, dispositivos, redes y otros productores o consumidores de información y 3) interfaces internas que involucran a los distintos componentes del diseño. Estos elementos del diseño de la interfaz permiten que el software se comunique externamente y permita la comunicación y colaboración internas entre los componentes que constituyen la arquitectura del software.

El diseño de la IU (denominada cada vez con más frecuencia *diseño de la usabilidad*) es una acción principal de la ingeniería de software y se estudia con detalle en el capítulo 11. El diseño de la usabilidad incorpora elementos estéticos (como distribución, color, gráficos, mecanismos de interacción, etc.), elementos ergonómicos (por ejemplo, distribución y colocación de la información, metáforas, navegación por la IU, etc.) y elementos técnicos (como patrones de la IU y patrones reutilizables). En general, la IU es un subsistema único dentro de la arquitectura general de la aplicación.

9.1.1 ¿Qué es la arquitectura?

Cuando se piensa en la arquitectura de una construcción, llegan a la mente muchos atributos distintos. En el nivel más sencillo, se considera la forma general de la estructura física. Pero, en realidad, la arquitectura es mucho más que eso. Es la manera en la que los distintos componentes del edificio se integran para formar un todo cohesivo. Es la forma en la que la construcción se adapta a su ambiente y se integra a los demás edificios en la vecindad. Es el grado en el que el edificio cumple con su propósito y en el que satisface las necesidades del propietario. Es la sensación estética de la estructura —el efecto visual de la edificación— y el modo en el que se combinan texturas, colores y materiales para crear la fachada en el exterior y el "ambiente de vida" en el interior. Es los pequeños detalles: diseño de las lámparas, tipo de piso, color de las cortinas... la lista es casi interminable. Y, finalmente, es arte.

Pero la arquitectura también es algo más. Son los "miles de decisiones, tanto grandes como pequeñas" [Tyt05]. Algunas de éstas se toman en una etapa temprana del diseño y tienen un efecto profundo en todas las demás acciones. Otras se dejan para más adelante, con lo que se eliminan las restricciones prematuras que llevarían a una mala implementación del estilo arquitectónico.

Pero, ¿qué es la arquitectura del software? Bass, Clements y Kazman [Bas03] definen este término tan elusivo de la manera siguiente:

> La arquitectura del software de un programa o sistema de cómputo es la estructura o estructuras del sistema, lo que comprende a los componentes del software, sus propiedades externas visibles y las relaciones entre ellos.

La arquitectura no es el software operativo. Es una representación que permite 1) analizar la efectividad del diseño para cumplir los requerimientos establecidos, 2) considerar alternativas arquitectónicas en una etapa en la que hacer cambios al diseño todavía es relativamente fácil y 3) reducir los riesgos asociados con la construcción del software.

Esta definición pone el énfasis en el papel de los "componentes del software" en cualquier representación arquitectónica. En el contexto del diseño de la arquitectura, un componente del software puede ser algo tan simple como un módulo de programa o una clase orientada a objeto, pero también puede ampliarse para que incluya bases de datos y "middleware" que permitan la configuración de una red de clientes y servidores. Las propiedades de los componentes son aquellas características necesarias para entender cómo interactúan unos componentes con otros. En el nivel arquitectónico, no se especifican las propiedades internas (por ejemplo, detalles de un algoritmo). Las relaciones entre los componentes pueden ser tan simples como una invocación de procedimiento de un módulo a otro o tan complejos como un protocolo de acceso a una base de datos.

En su texto evolutivo *Handbook of Software Architecture* [Boo08], Grady Booch sugiere los siguientes géneros arquitectónicos para sistemas basados en software:

- **Inteligencia artificial:** Sistemas que simulan o incrementan la cognición humana, su locomoción u otros procesos orgánicos.
- **Comerciales y no lucrativos:** Sistemas que son fundamentales para la operación de una empresa de negocios.
- **Comunicaciones:** Sistemas que proveen la infraestructura para transferir y manejar datos, para conectar usuarios de éstos o para presentar datos en la frontera de una infraestructura.
- **Contenido de autor:** Sistemas que se emplean para crear o manipular artefactos de texto o multimedios.
- **Dispositivos:** Sistemas que interactúan con el mundo físico a fin de brindar algún servicio puntual a un individuo.
- **Entretenimiento y deportes:** Sistemas que administran eventos públicos o que proveen una experiencia grupal de entretenimiento.
- **Financieros:** Sistemas que proporcionan la infraestructura para transferir y manejar dinero y otros títulos.
- **Juegos:** Sistemas que dan una experiencia de entretenimiento a individuos o grupos.

- **Gobierno:** Sistemas que dan apoyo a la conducción y operaciones de una institución política local, estatal, federal, global o de otro tipo.
- **Industrial:** Sistemas que simulan o controlan procesos físicos.
- **Legal:** Sistemas que dan apoyo a la industria jurídica.
- **Médicos:** Sistemas que diagnostican, curan o contribuyen a la investigación médica.
- **Militares:** Sistemas de consulta, comunicaciones, comando, control e inteligencia (o C4I), así como de armas ofensivas y defensivas.
- **Sistemas operativos:** Sistemas que están inmediatamente instalados en el hardware para dar servicios de software básico.
- **Plataformas:** Sistemas que se encuentran en los sistemas operativos para brindar servicios avanzados.
- **Científicos:** Sistemas que se emplean para hacer investigación científica y aplicada.
- **Herramientas:** Sistemas que se utilizan para desarrollar otros sistemas.
- **Transporte:** Sistemas que controlan vehículos acuáticos, terrestres, aéreos o espaciales.
- **Utilidades:** Sistemas que interactúan con otro software para brindar algún servicio específico.

Desde el punto de vista del diseño arquitectónico, cada género representa un desafío único. Por ejemplo, considere la arquitectura del software de un sistema de juego. Esta clase de sistemas, en ocasiones llamados *aplicaciones interactivas de inmersión*, requieren el cómputo de algoritmos intensivos, gráficas avanzadas en computadora, fuentes de datos continuas en multimedios, interactividad en tiempo real a través de dispositivos de entrada convencionales y no convencionales, y otras preocupaciones especializadas.

10.1 ¿Qué es un componente?

talles.

Un *componente* es un bloque de construcción de software de cómputo. Con más formalidad, la *Especificación OMG del Lenguaje de Modelado Unificado* [OMG03a] define un componente como "una parte modular, desplegable y sustituible de un sistema, que incluye la implantación y expone un conjunto de interfaces".

Como se dijo en el capítulo 9, los componentes forman la arquitectura del software y, en consecuencia, juegan un papel en el logro de los objetivos y de los requerimientos del sistema que se va a construir. Como los componentes se encuentran en la arquitectura del software, deben comunicarse y colaborar con otros componentes y con entidades (otros sistemas, dispositivos, personas, etc.) que existen fuera de las fronteras del software.

El verdadero significado del término *componente* difiere en función del punto de vista del ingeniero de software que lo use. En las secciones que siguen, se estudian tres visiones importantes de lo que es un componente y cómo se emplea en el desarrollo de la modelación del diseño.

10.1.1 Una visión orientada a objetos

En el contexto de la ingeniería de software orientada a objetos, un componente contiene un conjunto de clases que colaboran.[1] Cada clase dentro de un componente se elabora por completo para que incluya todos los atributos y operaciones relevantes para su implantación. Como parte de la elaboración del diseño, también deben definirse todas las interfaces que permiten que las clases se comuniquen y colaboren con otras clases de diseño. Para lograr esto, se comienza con el modelo de requerimientos y se elaboran clases de análisis (para los componentes que se relacionan con el dominio del problema) y clases de infraestructura (para los componentes que dan servicios de apoyo para el dominio del problema).

10.1.2 La visión tradicional

En el contexto de la ingeniería de software tradicional, un componente es un elemento funcional de un programa que incorpora la lógica del procesamiento, las estructuras de datos internas que

se requieren para implantar la lógica del procesamiento y una interfaz que permite la invocación del componente y el paso de los datos. Dentro de la arquitectura del software se encuentra un componente tradicional, también llamado *módulo*, que tiene tres funciones importantes: 1) como *componente de control* que coordina la invocación de todos los demás componentes del dominio del problema, 2) como *componente del dominio del problema* que implanta una función completa o parcial que requiere el cliente y 3) como *componente de infraestructura* que es responsable de las funciones que dan apoyo al procesamiento requerido en el dominio del problema.

10.1.3 Visión relacionada con el proceso

La visión orientada a objetos y la tradicional del diseño en el nivel de componentes, presentadas en las secciones 10.1.1 y 10.1.2, suponen que el componente se diseña desde la nada. Es decir, que se crea un nuevo componente con base en las especificaciones obtenidas del modelo de requerimientos. Por supuesto, existe otro enfoque.

En las últimas dos décadas, la comunidad de la ingeniería de software ha puesto el énfasis en la necesidad de elaborar sistemas que utilicen componentes de software o patrones de diseño ya existentes. En esencia, a medida que avanza el trabajo de diseño se dispone de un catálogo de diseño probado o de componentes en el nivel de código. Conforme se desarrolla la arquitectura del software, se escogen del catálogo componentes o patrones de diseño y se usan para construir la arquitectura. Como estos componentes fueron construidos teniendo en mente lo reutilizable, se dispone totalmente de la descripción de su interfaz, de las funciones que realizan y de la comunicación y colaboración que requieren. En la sección 10.6 se estudian algunos aspectos importantes de la ingeniería de software basada en componentes.

10.2.3 Cohesión

En el capítulo 8 se describió la cohesión como la "unidad de objetivo" de un componente. En el contexto del diseño en el nivel de componentes para los sistemas orientados a objetos, la *cohesión* implica que un componente o clase sólo contiene atributos y operaciones que se relacionan de cerca uno con el otro y con la clase o componente en sí. Lethbridge y Laganiére [Let01] definen varios tipos diferentes de cohesión (se listan en función del nivel de cohesión):[4]

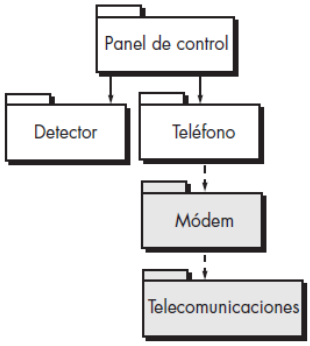
Funcional. Lo tienen sobre todo las operaciones; este nivel de cohesión ocurre cuando un componente realiza un cálculo y luego devuelve el resultado.

De capa. Lo tienen los paquetes, componentes y clases; este tipo de cohesión ocurre cuando una capa más alta accede a los servicios de otra más baja, pero ésta no tiene acceso a las superiores. Por ejemplo, considere el requerimiento de la función de seguridad de *CasaSegura* para hacer una llamada telefónica si se detecta una alarma. Podría definirse un conjunto de paquetes en capas, como se aprecia en la figura 10.5. Los paquetes sombreados contienen componentes de infraestructura. Es posible realizar el acceso del paquete del panel de control hacia abajo.

De comunicación. Todas las operaciones que acceden a los mismos datos se definen dentro de una clase. En general, tales clases se centran únicamente en los datos en cuestión, acceden a ellos y los guardan.

Las clases y componentes que tienen cohesión funcional, de capa y comunicación son relativamente fáciles de implantar, probar y mantener. Siempre que sea posible, deben alcanzarse estos niveles de cohesión. Sin embargo, es importante notar que en ocasiones hay aspectos pragmáticos del diseño y de la implantación que obligan a optar por niveles de cohesión más bajos.

FIGURA 10.5
Cohesión de capa



4 En general, entre más alto sea el nivel de cohesión, el componente es más fácil de implantar, probar y mantener.

10.2.4 Acoplamiento

En el estudio anterior del análisis y el diseño, se dijo que la comunicación y la colaboración eran elementos esenciales de cualquier sistema orientado a objetos. Sin embargo, esta característica tan importante (y necesaria) tiene un lado oscuro. A medida que aumentan la comunicación y colaboración (es decir, conforme se eleva la "conectividad" entre las clases), la complejidad del sistema también se incrementa. Y si la complejidad aumenta, también crece la dificultad de implantar, probar y dar mantenimiento al software.

El *acoplamiento* es la medición cualitativa del grado en el que las clases se conectan una con otra. Conforme las clases (y componentes) se hacen más interdependientes, el acoplamiento crece. Un objetivo importante del diseño en el nivel de componente es mantener el acoplamiento tan bajo como sea posible.

El acoplamiento de las clases se manifiesta de varias maneras. Lethbridge y Laganiére [Let01] definen las siguientes categorías de acoplamiento:

Acoplamiento de contenido. Tiene lugar cuando un componente "modifica subrepticiamente datos internos en otro componente" [Let01]. Esto viola el ocultamiento de la información, concepto básico del diseño.

Acoplamiento común. Sucede cuando cierto número de componentes hacen uso de una variable global. Aunque a veces esto es necesario (por ejemplo, para establecer valores de-

finidos que se utilizan en toda la aplicación), el acoplamiento común lleva a la propagación incontrolada del error y a efectos colaterales imprevistos cuando se hacen los cambios.

Acoplamiento del control. Tiene lugar si la operación *A( )* invoca a la operación *B( )* y pasa una bandera de control a *B*. La bandera "dirige" entonces el flujo de la lógica dentro de *B*. El problema con esta forma de acoplamiento es que un cambio no relacionado en *B* puede dar como resultado la necesidad de cambiar el significado de la bandera de control que pasa *A*. Si esto se pasa por alto ocurrirá un error.

Acoplamiento de molde. Se presenta cuando se declara a **ClaseB** como un tipo para un argumento de una operación de **ClaseA**. Como **ClaseB** ahora forma parte de la definición de **ClaseA**, la modificación del sistema se vuelve más compleja.

Acoplamiento de datos. Ocurre si las operaciones pasan cadenas largas de argumentos de datos. El "ancho de banda" de la comunicación entre clases y componentes crece y la complejidad de la interfaz se incrementa. Se hace más difícil hacer pruebas y dar mantenimiento.

Acoplamiento de rutina de llamada. Tiene lugar cuando una operación invoca a otra. Este nivel de acoplamiento es común y con frecuencia muy necesario. Sin embargo, aumenta la conectividad del sistema.

Acoplamiento de tipo de uso. Ocurre si el componente **A** usa un tipo de datos definidos en el componente **B** (esto ocurre siempre que "una clase declara una variable de instancia o una variable local como si tuviera otra clase para su tipo" [Let01]). Si cambia la definición de tipo, también debe cambiar todo componente que la utilice.

Acoplamiento de inclusión o importación. Pasa cuando el componente **A** importa o incluye un paquete o el contenido del componente **B**.

Acoplamiento externo. Sucede si un componente se comunica o colabora con componentes de infraestructura (por ejemplo, funciones del sistema operativo, capacidad de la base de datos, funciones de telecomunicación, etc.). Aunque este tipo de acoplamiento es necesario, debe limitarse a un número pequeño de componentes o clases dentro de un sistema.

El software debe tener comunicación interna y externa. Por tanto, el acoplamiento es un hecho de la vida. Sin embargo, el diseñador debe trabajar para reducirlo siempre que sea posible, y entender las ramificaciones que tiene el acoplamiento abundante cuando no puede evitarse.

UNA MIRADA RÁPIDA

¿Qué es? El diseño de la interfaz de usuario crea un medio eficaz de comunicación entre los seres humanos y la computadora. Siguiendo un conjunto de principios de diseño de la interfaz, el diseño identifica los objetos y acciones de ésta y luego crea una plantilla de pantalla que constituye la base del prototipo de la interfaz de usuario.

¿Quién lo hace? Un ingeniero de software diseña la interfaz de usuario con la aplicación de un proceso iterativo que sigue principios de diseño predefinidos.

¿Por qué es importante? Si el software es difícil de usar, fuerza al usuario a cometer errores, o si frustra sus esfuerzos para alcanzar las metas, entonces no le gustará, sin que importe el poder computacional que tenga, el contenido que entregue o las funciones que ofrezca. La interfaz tiene que estar bien hecha porque moldea la percepción que el usuario tiene del software.

¿Cuáles son los pasos? El diseño de la interfaz de usuario comienza con la identificación de los requerimientos del usuario, la tarea y el ambiente. Una vez identificadas las tareas del usuario, se crean y analizan los escenarios para éste y se define un conjunto de objetos y acciones de la interfaz. Esto forma la base para crear una plantilla de la pantalla que ilustra el diseño gráfico y la colocación de los iconos, la definición de textos descriptivos, la especificación y títulos de las ventanas, y la especificación de aspectos mayores y menores del menú. Con el empleo de herramientas, se hace el prototipo, se implementa en definitiva el modelo del diseño y se evalúa la calidad del resultado.

¿Cuál es el producto final? Se crean los escenarios del usuario y se generan los formatos de la pantalla. Se desarrolla un prototipo de la interfaz y se modifica de manera iterativa.

¿Cómo me aseguro de que lo hice bien? Los usuarios "prueban" un prototipo de la interfaz y la retroalimentación de esta prueba se utiliza para la siguiente modificación iterativa del prototipo.

LAS REGLAS DORADAS

En su libro sobre el diseño de la interfaz, Theo Mandel [Man97] acuñó tres *reglas doradas*:

1. Dejar el control al usuario.
2. Reducir la carga de memoria del usuario.
3. Hacer que la interfaz sea consistente.

En realidad, estas reglas doradas constituyen la base de un conjunto de principios de diseño de la interfaz de usuario que guían este aspecto tan importante del diseño del software.

11.1.1 Dejar el control al usuario

Definir modos de interacción de manera que no se obligue al usuario a realizar acciones innecesarias o no deseadas. Un modo de interacción es el estado actual de la interfaz. Por ejemplo, si en el menú de un procesador de textos se selecciona *revisar ortografía*, el software pasa al modo de revisión de la ortografía. No hay razón para obligar al usuario a permanecer en este modo si acaso desea hacer una pequeña edición del texto. El usuario debe poder entrar y salir del modo con poco o ningún esfuerzo.

Dar una interacción flexible. Debido a que diferentes usuarios tienen distintas preferencias para la interacción, debe darse la posibilidad de elegir. Por ejemplo, el software debe permitir que el usuario interactúe por medio de comandos introducidos con el teclado, el ratón, una pluma digitalizadora, una pantalla sensible al tacto o un mecanismo de reconocimiento de voz. Pero no todas las acciones son accesibles a través de cualquier mecanismo de interacción. Por ejemplo, piénsese en la dificultad de usar comandos del teclado (o entradas con la voz) para hacer un dibujo complicado.

Permitir que la interacción del usuario sea interrumpible y también reversible. El usuario debe poder suspender la secuencia de su trabajo (aun cuando consista en una secuencia de acciones) para hacer otra cosa (sin perder el trabajo realizado hasta ese momento). También debe poder "deshacer" cualquier acción.

Facilitar la interacción a medida que aumenta la habilidad y permitir que aquélla se personalice. Es frecuente que los usuarios realicen la misma secuencia de interacciones en forma repetida. Es benéfico diseñar un mecanismo de "macros" que permita que un usuario avanzado personalice la interfaz para facilitar la interacción.

Ocultar los tecnicismos internos al usuario ocasional. La interfaz de usuario debe introducirlo al mundo virtual de la aplicación. El usuario no debe tener que preocuparse del sistema operativo, de las funciones de administración de archivos ni de ninguna otra tecnología de computación secreta. En esencia, la interfaz no debe requerir que el usuario interactúe en un nivel "interno" de la máquina (nunca debería tener que escribir comandos del sistema operativo desde una aplicación de software).

Diseñar la interacción directa con objetos que aparezcan en la pantalla. El usuario tiene la sensación de control cuando puede manipular los objetos que se necesitan a fin de ejecutar un trabajo en la misma forma en la que lo haría si el objeto fuera algo físico. Por ejemplo, una interfaz de aplicación que le permita "estirar" un objeto (modificar su tamaño) es una implementación de manipulación directa.

11.1.2 Reducir la necesidad de que el usuario memorice

Entre más cosas tenga que recordar el usuario, más fácil será que cometa errores al interactuar con el sistema. Es por esto que una interfaz de usuario bien diseñada no sobrecarga la memoria del usuario. Siempre que sea posible, el sistema debe "recordar" la información pertinente y ayudar al usuario con un escenario de interacción que lo ayude a recordar. Mandel [Man97] define los siguientes principios de diseño que permiten que una interfaz reduzca la necesidad de que el usuario memorice:

Reducir la demanda de memoria de corto plazo. Cuando los usuarios se involucran en tareas complejas, la demanda de memoria de corto plazo es significativa. La interfaz debe diseñarse para disminuir la necesidad de recordar acciones, entradas y resultados del pasado. Esto se logra dando claves visuales que permitan al usuario reconocer acciones anteriores, en lugar de que tenga que recordarlas.

Hacer que lo preestablecido sea significativo. Lo que al principio se dé por preestablecido debe tener sentido para el usuario promedio, pero éste debería poder especificar sus preferencias individuales. Sin embargo, debe disponerse de la opción de "reiniciar" para restablecer los valores originales.

Definir atajos que sean intuitivos. Cuando se utilice nemotecnia para ejecutar una función del sistema (como la secuencia Ctrl-B para invocar la función de buscar), debe estar ligada con la acción, de modo que sea fácil de recordar (por ejemplo, con la primera letra de la tarea que se va a realizar).

La distribución visual de la interfaz debe basarse en una metáfora del mundo real. Por ejemplo, un sistema de pagos debe usar una metáfora de chequera y talonario que guíe al usuario a través del proceso de pago. Esto permite que el usuario se base en claves visuales que comprende bien, en vez de tener que memorizar una secuencia críptica de interacciones.

Revelar información de manera progresiva. La interfaz debe estar organizada de manera jerárquica. Es decir, la información acerca de una tarea, objeto o comportamiento debe presentarse primero en un nivel de generalización elevado. Después de que con el ratón el usuario manifieste interés, deben darse más detalles. Un ejemplo, común para muchas aplicaciones de procesamiento de textos, es la función de subrayar. La función en sí es una de varias en el menú *estilo del texto*. No obstante, no se enlista cada una de las herramientas para subrayar. El usuario debe hacer *clic* en la opción de subrayar; después se presentan todas las opciones para esta función (una raya, doble raya, línea punteada, etcétera).

11.1.3 Hacer consistente la interfaz

La interfaz debe presentar y obtener información en forma consistente. Esto implica: 1) que toda la información se organice de acuerdo con reglas de diseño que se respeten en todas las pantallas desplegadas, 2) que los mecanismos de entrada se limiten a un conjunto pequeño usado en forma consistente en toda la aplicación, y 3) que los mecanismos para pasar de una tarea a otra se definan e implementen de modo consistente. Mandel [Man97] define varios principios de diseño que ayudan a que la interfaz tenga consistencia:

Permita que el usuario coloque la tarea en curso en un contexto significativo. Muchas interfaces implementan capas complejas de interacciones con decenas de imágenes en la pantalla. Es importante dar indicadores (títulos en las ventanas, iconos gráficos, código de colores consistente, etc.) que permitan al usuario conocer el contexto del trabajo en curso. Además, debe poder determinar de dónde viene y qué alternativas hay para hacer la transición a una nueva tarea.

Mantener la consistencia en toda la familia de aplicaciones. Todas las aplicaciones (o productos) que hay en un grupo deben implementar las mismas reglas de diseño a fin de que se mantenga la consistencia en toda la interacción.

Si los modelos interactivos anteriores han creado expectativas en el usuario, no haga cambios a menos de que haya una razón ineludible para ello. Una vez que una secuencia interactiva en particular se ha convertido en un estándar (como el uso de alt-G para guardar un archivo), el usuario la espera en toda aplicación que emplea. Un cambio (como utilizar alt-G para invocar la función de modificar la escala) generará confusión.

Los principios de diseño de la interfaz analizados en esta sección y en las anteriores dan una guía básica. En las que siguen, el lector aprenderá acerca del proceso de diseño de la interfaz en sí.

11.2.1 Análisis y modelos del diseño de la interfaz

Cuando se analiza y diseña la interfaz de usuario, entran en juego cuatro diferentes modelos. Un ingeniero (o el encargado del software) establece un *modelo de usuario*, el ingeniero de software crea un *modelo del diseño*, el usuario final desarrolla una imagen mental que frecuentemente se nombra *modelo mental* o *percepción del sistema*, y los implementadores del sistema crean un *modelo de implementación*. Desafortunadamente, cada uno de estos modelos difiere en forma significativa. El papel del diseñador de la interfaz es conciliar estas diferencias y obtener una representación consistente de la interfaz.

El *modelo mental* del usuario (percepción del sistema) es la imagen del sistema que los usuarios finales llevan en la cabeza. Por ejemplo, si se pidiera a un usuario de un procesador de texto en particular que describiera su operación, lo que guiaría su respuesta sería la percepción que tuviera del sistema. La exactitud de la descripción dependerá del perfil del usuario (por ejemplo, en el mejor de los casos, los principiantes darán una respuesta esquemática) y de la familiaridad general con el software en el dominio de la aplicación. Un usuario que entienda bien los procesadores de texto, pero que haya trabajado con el procesador específico una sola vez, tal vez esté más preparado para hacer una descripción más completa de su funcionamiento que el principiante que haya pasado semanas tratando de entender el sistema.

El *modelo de implementación* combina la manifestación externa del sistema basado en computadora (la vista y sensación de la interfaz) con toda la información de apoyo (libros, manuales, videos, archivos de ayuda, etc.) que describe la sintaxis y semántica de la interfaz. Cuando el modelo de la implementación y el modelo mental del usuario coinciden, quienes utilizan el software por lo general se sienten cómodos con éste y lo usan de manera eficaz. Para lograr esta "fusión" de los modelos, el modelo del diseño debe haberse desarrollado de manera que incluya la información contenida en el modelo del usuario, y el modelo de la implementación debe reflejar de manera exacta la información sintáctica y semántica de la interfaz.

Los modelos descritos en esta sección son "abstracciones de lo que el usuario hace o piensa que hace o de lo que alguien piensa que debe hacerse cuando se usa un sistema interactivo" [Mon84]. En esencia, estos modelos permiten que el diseñador de la interfaz satisfaga un ele-

11.2.2 El proceso

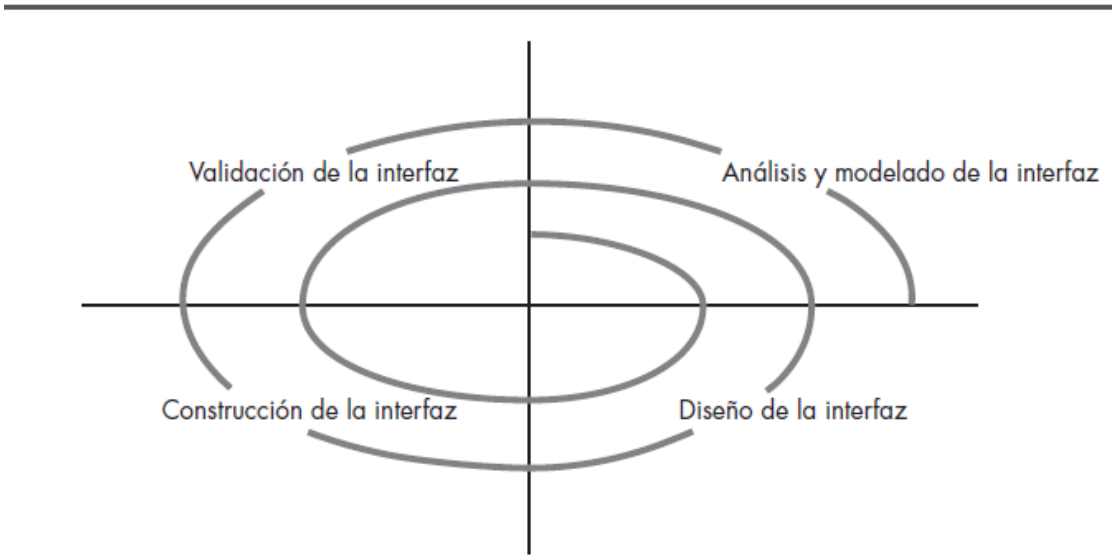
El proceso de análisis y diseño de interfaces de usuario es iterativo y se representa con un modelo espiral similar al que se estudió en el capítulo 2. En relación con la figura 11.1, el proceso de análisis y diseño de la interfaz de usuario comienza en el interior de la espiral e incluye cuatro actividades estructurales distintas [Man97]: 1) análisis y modelado de la interfaz, 2) diseño de ésta, 3) construcción y 4) validación. La espiral que se presenta en la figura 11.1 implica que cada una de dichas tareas tendrá lugar más de una vez y que cada recorrido del contorno de la espiral representa una elaboración mayor de los requerimientos y del diseño resultante. En la mayoría de los casos, la actividad de modelado involucra la hechura de prototipos, única forma práctica de validar lo que se haya diseñado.

El *análisis de la interfaz* se centra en el perfil de los usuarios que interactuarán con el sistema. Se registra el nivel de habilidad, la comprensión del negocio y la receptividad general hacia el nuevo sistema; también se definen diferentes categorías de usuarios. Se recaban los requerimientos de cada una de éstas. En esencia, se trabaja para entender la percepción del sistema (véase la sección 11.2.1) para cada clase de usuarios.

Una vez definidos los requerimientos generales, se lleva a cabo un detallado *análisis de la tarea*. Asimismo, se identifican, describen y elaboran aquellas tareas que el usuario realice para alcanzar las metas del sistema (por medio de varios recorridos de la espiral). En la sección 11.3, se estudia con más detalle el análisis de la tarea. Por último, el análisis del ambiente del usuario se centra en las características físicas del lugar de trabajo. Entre las preguntas por responder se encuentran las siguientes:

- ¿Dónde se encontrará físicamente la interfaz?
- ¿El usuario estará sentado, de pie o haciendo otras tareas no relacionadas con la interfaz?
- ¿El hardware de la interfaz cumple las restricciones de espacio, iluminación o ruido?
- ¿Hay consideraciones especiales de factores humanos generadas por los factores ambientales?

La información recabada como parte del análisis se utiliza para crear un modelo de análisis de la interfaz. Con este modelo como base comienza la acción de diseñar.



La meta del *diseño de la interfaz* es definir un conjunto de objetos y acciones de ésta (y sus representaciones en la pantalla) que permitan al usuario efectuar todas las tareas definidas en forma tal que cumpla cada meta de la usabilidad definida para el sistema. El diseño de la interfaz se estudia con más detalle en la sección 11.4.

La *construcción de la interfaz* comienza por lo general con la creación de un prototipo que permite evaluar los escenarios de uso. A medida que avanza el proceso de diseño, se emplea un grupo de herramientas de la interfaz de usuario (véase la sección 11.5) para terminar de construirla.

La *validación de la interfaz* se centra en: 1) la capacidad de la interfaz para implementar correctamente todas las tareas del usuario, incluir todas las variaciones de éstas y alcanzar todos los requerimientos generales del usuario; 2) el grado en el que la interfaz es fácil de usar y de aprender y 3) la aceptación que tiene por parte del usuario como herramienta útil en su trabajo.

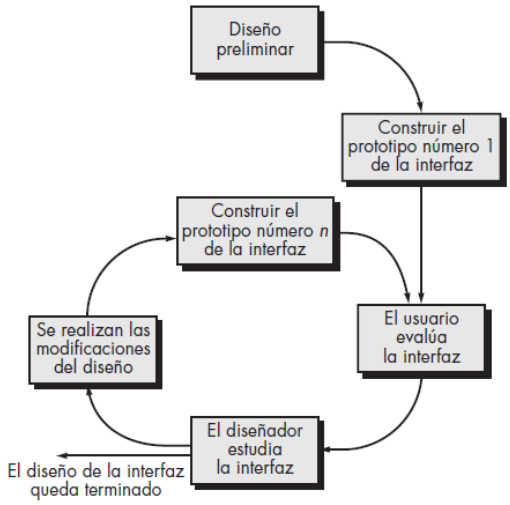
Como ya se dijo, las actividades descritas en esta sección ocurren de manera iterativa. Por esto, no es necesario intentar especificar cada uno de los detalles (del modelo de análisis o de diseño) en la primera etapa. En los pasos posteriores del proceso, se elaboran los detalles de la tarea, la información de diseño y las características de operación de la interfaz.

11.6 Evaluación del diseño

Una vez que se crea un prototipo operativo de la interfaz de usuario, debe evaluarse con objeto de determinar si satisfacen las necesidades de éste. La evaluación abarca un espectro de formalidad que va desde una "prueba de manejo" informal, en la que el usuario da retroalimentación instantánea a un estudio diseñado formalmente que utilice métodos estadísticos para evaluar cuestionarios que respondería una población de usuarios finales.

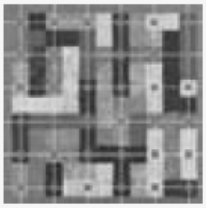
El ciclo de evaluación de la interfaz de usuario toma la forma que se aprecia en la figura 11.5. Una vez terminado el modelo del diseño, se crea un prototipo de primer nivel. Éste es evaluado

Figura 11.5
Ciclo de evaluación del diseño de la interfaz



por el usuario,[11] quien hace comentarios directos acerca de la eficacia de la interfaz. Además, se emplean técnicas formales de evaluación (tales como cuestionarios, hojas de calificación, etc.), de las que se extrae información (por ejemplo: a 80 por ciento de todos los usuarios no le gusta el mecanismo para guardar archivos de datos). Las modificaciones del diseño se hacen con base en las aportaciones de los usuarios, y así se crea el siguiente nivel de prototipo. El ciclo de evaluación continúa hasta que ya no es necesario modificar más el diseño de la interfaz.

Diseño de las SALIDAS



La salida es información que se entrega a los usuarios por medio del sistema de información a través de intranets, extranets o la World Wide Web. Algunos datos requieren de mucho procesamiento para poder convertirse en una salida adecuada; otros se almacenan y, cuando se recuperan, se consideran salida sin que necesiten mucho procesamiento (a veces no requieren procesamiento en absoluto). La salida puede tomar muchas formas: la tradicional copia en papel de los informes impresos y la copia transitoria como las pantallas, microformas y la salida de video y audio. Los usuarios se basan en la salida para realizar sus tareas y con frecuencia juzgan el mérito del sistema únicamente con base en ella. Para crear la salida más útil posible, el analista de sistemas trabaja de cerca con el usuario a través de un proceso interactivo hasta que se considere que el resultado es satisfactorio.

OBJETIVOS DE DISEÑO DE LA SALIDA

Como es esencial una salida útil para asegurar el uso y la aceptación del sistema de información, hay seis objetivos que el analista de sistemas trata de alcanzar al diseñar la salida:

1. Diseñar la salida para servir al propósito previsto.
2. Diseñar la salida para ajustarla al usuario.
3. Entregar la cantidad apropiada de salida.
4. Asegurarse que la salida esté donde se necesite.
5. Proveer la salida en forma oportuna.
6. Elegir el método de salida correcto.

Diseñar la salida para servir al propósito previsto

Toda salida debe tener un propósito. Durante la fase de determinación de los requerimientos de información del análisis, el analista de sistemas averigua qué propósitos de los usuarios y la organización existen. Después se diseña la salida con base en esos propósitos.

Usted tendrá numerosas oportunidades para proveer la salida simplemente porque la aplicación así se lo permite. Sin embargo, recuerde la regla de propósito: si la salida no es funcional, no se debe crear ya que acarrea costos de tiempo y materiales.

Diseñar la salida para ajustarla al usuario

En un sistema de información extenso que atiende a muchos usuarios con muchos fines, y a menudo es difícil personalizar la salida. Con base en las entrevistas, observaciones, consideraciones de costo y tal vez los prototipos, es posible diseñar una salida para atender lo que muchos usuarios (si no es que todos) necesitan y prefieren.

Hablando en general, es más práctico crear una salida específica o personalizada para el usuario cuando se diseña para un sistema de soporte de decisiones u otras aplicaciones con alto grado de interactividad, como las que utilizan a la Web como plataforma. Sin embargo, sí es posible diseñar la salida para ajustarse a las tareas y la función de un usuario en la organización, lo cual nos lleva al siguiente objetivo.

Entregar la cantidad apropiada de salida

Parte de la tarea de diseñar la salida es decidir qué cantidad de salida es la correcta para los usuarios. Una heurística útil es que el sistema debe proveer lo que necesita cada persona para realizar su trabajo. Pero esta respuesta está aún lejos de ser una solución total, ya que tal vez sea apropiado mostrar un subconjunto de esa información al principio y después ofrecer al usuario la manera de acceder a la información adicional con facilidad.

El problema de la sobrecarga de información se ha convertido en una condición muy frecuente, por lo que constituye una preocupación para el diseñador de sistemas. No sirve de nada proporcionar información excesiva sólo para alardear sobre las capacidades del sistema. Siempre hay que tener en cuenta a los encargados de tomar las decisiones. Muy raras veces necesitarán grandes cantidades de salida, en especial si hay una forma sencilla de acceder a más información a través de un hipervínculo o una herramienta para explorar estructuras jerárquicas.

Asegurarse que la salida esté donde se necesite

Con frecuencia la salida se produce en una ubicación y después se distribuye al usuario. El aumento en el uso de la salida que se muestra a través de una pantalla en línea y a la que se puede acceder personalmente ha reducido en parte el problema de la distribución, pero la distribución apropiada sigue siendo un objetivo importante para el analista de sistemas. Para ser útil, la salida se debe presentar al usuario correcto. Sin importar qué tan bien diseñados estén los informes, si no los ven los encargados apropiados de tomar las decisiones, no tendrán valor.

Proveer la salida en forma oportuna

Una de las quejas más comunes de los usuarios es que no reciben la información a tiempo para tomar las decisiones necesarias. Aunque la sincronización no lo es todo, sí desempeña un importante papel en cuanto a qué tan útil será la salida. Muchos informes se requieren a diario, algunos sólo cada mes, otros anualmente y otros más sólo por excepción. Al usar una salida basada en Web bien promocionada, también podemos solucionar algunos problemas de sincronización de la distribución de salida. La sincronización precisa de la salida puede ser imprescindible para las operaciones de negocios.

Elegir el método de salida correcto

La elección del método de salida correcto para cada usuario es otro de los objetivos del diseño correcto de salida. Gran parte de la salida aparece ahora en pantallas donde los usuarios tienen la opción de imprimirla. El analista necesita reconocer los conflictos involucrados en el proceso de elegir un método de salida. Los costos difieren; para el usuario también hay diferencias en cuanto a accesibilidad, flexibilidad, durabilidad, distribución, posibilidades de almacenamiento y recuperación, capacidad de transportación e impacto general de los datos. La elección de los métodos de salida no es trivial y tampoco es una cuestión resuelta.

RELACIONAR EL CONTENIDO DE LA SALIDA CON EL MÉTODO DE SALIDA

Debemos considerar que el contenido de la salida de los sistemas de información está interrelacionado con el método de salida. Cada vez que diseñamos la salida, necesitamos pensar acerca de cómo la función influye sobre la forma y cómo influirá el propósito previsto en el método de salida que seleccionemos.

Debemos pensar sobre la salida en sentido general, de manera que cualquier información que produzca el sistema computarizado y que sea útil para las personas se pueda considerar como salida. Es posible conceptualizar la salida como externa (que sale de la empresa); por ejemplo, la información que aparece en el sitio Web para el público, o como interna (que permanece dentro del negocio); por ejemplo, el material disponible en una intranet.

Método de salida	Ventajas	Desventajas
Impresora	• Asequible para la mayoría de las organizaciones • Flexible en cuanto a tipos de salida, ubicación y capacidades • Maneja grandes volúmenes de salida • Altamente confiable, con poco tiempo de inactividad	• Aún se requiere de intervención por parte del operador. • Problemas de compatibilidad con el software de computadora • Puede requerir provisiones especiales y costosas • Puede ser lenta, dependiendo del modelo • Poco amigable para el ambiente
Pantalla	• Interactiva • Transmisión en línea y tiempo real • Silenciosa • Aprovecha las capacidades de la computadora para desplazarse por las bases de datos y los archivos • Buena para los mensajes efímeros con acceso frecuente	• Puede requerir cables y espacio para su instalación • De todas formas se puede requerir documentación impresa
Salida de audio y podcasts	• Buena para usuarios individuales • Buena para mensajes transientes • Buena en donde el trabajador necesita las manos libres • Buena si hay que distribuir la salida en áreas muy amplias	• Se requieren audífonos cuando la salida no debe interferir con otras tareas • Tiene una aplicación limitada
DVC, CD-ROM y CD-RW	• Tiene gran capacidad • Permite salida multimedia	• Se requiere una computadora y una pantalla para leer los datos
Salida electrónica (email, sitios Web, blogs y fuentes RSS)	• Reduce el papel • Se puede actualizar con mucha facilidad • Se puede "transmitir" • Se puede hacer interactiva	• No contribuye al formato (email) • Es difícil transmitir el contexto de los mensajes (email) • Los sitios Web requieren de un mantenimiento diligente

FIGURA 11.2

Una comparación de los métodos de salida.

Cómo diseñar la salida impresa

La fuente de información que debemos incluir en los informes es el diccionario de datos, la compilación de lo que vimos en el capítulo 8. Recuerde que el diccionario de datos incluye los nombres de los elementos de datos, así como la longitud de campo requerida para cada entrada.

Los informes se clasifican en tres categorías: detallados, por excepciones y sintetizados. Los informes detallados imprimen una línea por cada registro en el archivo maestro. Se utilizan para enviar correspondencia a los clientes, enviar las calificaciones de los estudiantes, imprimir catálogos, etcétera. Las pantallas de consulta han sustituido a muchos informes.

Los informes por excepciones imprimen una línea para todos los registros que coincidan con un conjunto de condiciones, como cuáles serán las decoraciones de días festivos que tendrán un descuento un día después del día festivo, o cuáles estudiantes estarán en la lista de honor. Por lo general se utilizan para ayudar a los gerentes de operaciones y al personal de oficina a operar un negocio. Los informes sintetizados imprimen una línea para un grupo de registros y se utilizan para tomar decisiones, como cuáles son los artículos que no se venden y cuáles se venden mucho.

Lineamientos para el diseño de informes impresos

La figura 11.6 es un informe de salida previsto para los gerentes divisionales de un mayorista de alimentos que abastece a varias tiendas de abarrotes de franquicia. Nos enfocaremos en diversos aspectos del informe a medida que examinemos herramientas, convenciones y atributos funcionales y de estilo de los informes de salida impresos.

CONVENCIONES DE DISEÑO DE LOS INFORMES Las convenciones a seguir al momento de diseñar un formulario incluyen el tipo de datos (alfabético, especial o numérico) que aparecerá en cada posición, mostrar el tamaño del formulario que se está preparando y mostrar la forma de indicar que los datos continúan en formularios dispuestos en forma consecutiva. La mayoría del software de diseño de formularios que utilizan los analistas en la actualidad posee convenciones estandarizadas para diseñar formularios en pantalla. Además se incluyen interfaces familiares tipo "arrastrar y soltar" que nos permiten seleccionar atributos con el ratón, como un bloque de direcciones, para

después soltarlo en la pantalla donde deseamos colocarlo en el formulario. Es muy común utilizar WYSIWYG (lo que se ve es lo que se obtiene), por lo que el diseño de formularios es un ejercicio muy visual.

La *información constante* es información que permanece igual cada vez que se imprime el informe. El título del informe y todos los encabezados de las columnas se escriben como información constante. La *información variable* es información que puede variar cada vez que se imprime el informe. En nuestro ejemplo cambian las cifras de ventas en miles de dólares; por ende, se indican como información variable.

CALIDAD DE PAPEL, TIPO Y TAMAÑO La salida se puede imprimir en innumerables tipos de papel. Por lo general, la restricción primordial es el costo. Un ejemplo es el uso de papel de seguridad para los cheques y los sobres de los cheques, así como para los documentos que deben portar sellos oficiales inalterables u hologramas, como los pasaportes.

Los formularios pre-impresos pueden transmitir con facilidad una imagen corporativa distintiva mediante el uso de los colores, logotipos y otros elementos de diseño relacionados con la empresa. El uso de formas, colores y distribuciones innovadoras también es una forma efectiva de llamar la atención de los usuarios para el informe que contiene el formulario impreso.

CONSIDERACIONES DE DISEÑO Al diseñar el informe impreso, el analista de sistemas trabaja con los usuarios para incorporar las consideraciones funcionales y estilísticas o estéticas, de manera que el informe provea al usuario la información necesaria en un formato legible y agradable. Como la función y la forma se refuerzan una a otra, no hay que hacer énfasis en una a expensas de la otra.

Atributos funcionales Los atributos funcionales de un informe impreso incluyen: 1) el encabezado o título del informe, 2) el número de página, 3) la fecha de preparación, 4) los encabezados de columnas, 5) el agrupamiento de los elementos de datos relacionados y 6) el uso de interrupciones de control. Cada uno de estos atributos sirve un propósito distintivo para el usuario.

Hay varias consideraciones estilísticas o estéticas que el analista de sistemas debe observar al diseñar un informe impreso. Si la salida impresa es poco atractiva y difícil de leer, no se utilizará en forma efectiva o tal vez no se utilice para nada. El resultado es la desinformación de los encargados de tomar decisiones y un desperdicio de los recursos de la organización.

CÓMO DISEÑAR SALIDA PARA PANTALLAS

En el capítulo 12 veremos cómo diseñar pantallas para entrada humana o de computadora; aquí se aplican los mismos lineamientos para diseñar la salida, aunque el contenido varía. Cabe mencionar que la salida para las pantallas difiere de la salida impresa en varios puntos. Es efímera (es decir, una pantalla no es permanente como las impresiones), puede estar orientada de manera más específica al usuario, está disponible en un horario más flexible, *no* es portátil de la misma forma y algunas veces se puede modificar a través de la interacción directa.

Además, hay que instruir a los usuarios en cuanto a las teclas que deben oprimir, los vínculos en los que deben hacer clic o cómo desplazarse cuando quieren continuar leyendo pantallas adicionales, cuando desean saber cómo terminar la pantalla y cuando quieren saber cómo interactuar con la pantalla (si es posible). El acceso de los usuarios a las pantallas se puede controlar mediante una contraseña, mientras que la distribución de la salida impresa se controla en base a otros medios.

Lineamientos para el diseño de pantallas

Cuatro lineamientos facilitan el diseño de pantallas:

1. Mantenga la pantalla simplificada.
2. Mantenga la presentación consistente.
3. Facilite el movimiento del usuario entre la salida que se muestra en la pantalla.
4. Cree una pantalla atractiva y agradable.

Al igual que con la salida impresa, las buenas pantallas no se crean en forma aislada. Los analistas de sistemas necesitan la retroalimentación de los usuarios para diseñar pantallas que valgan la pena. Una vez aprobada por los usuarios después de varios prototipos y refinaciones, se puede finalizar la distribución de la pantalla.

En la figura 11.7 se muestra la salida producida desde la pantalla de diseño. Cabe mencionar que no está atestada de información, e incluso así proporciona un resumen básico del estado de envío. La pantalla orienta a los usuarios en cuanto a lo que están viendo mediante el uso de un encabezado. Las instrucciones en la parte in-

FIGURA 11.7

La pantalla de salida de New Zoo está ordenada y orienta bien a los usuarios.

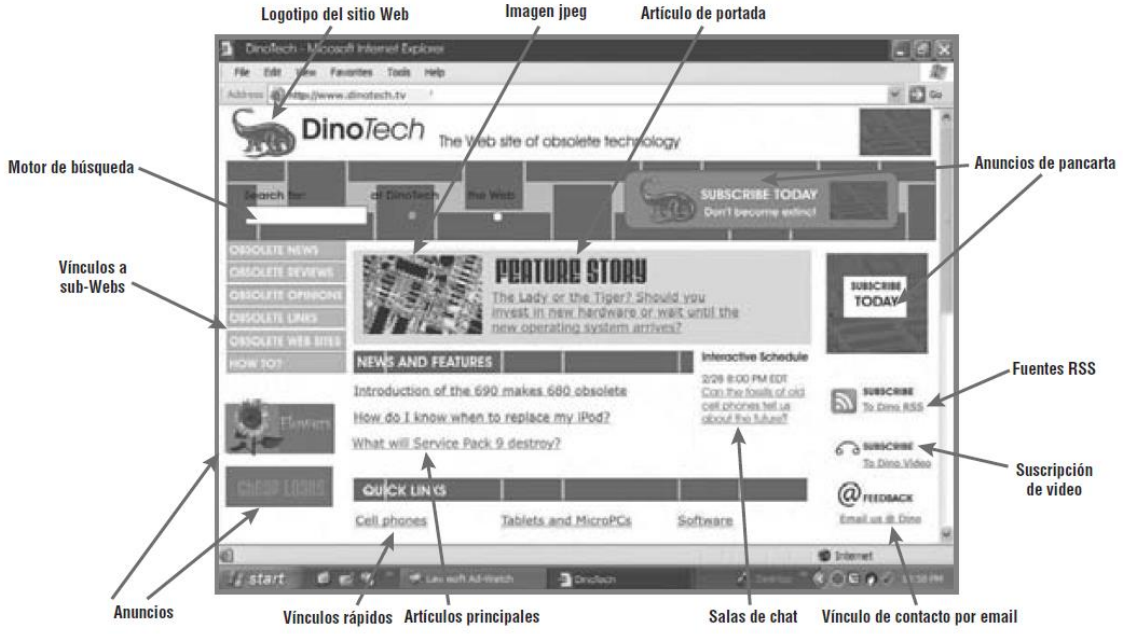
Estado de pedidos de New Zoo

Vendedor	Pedido #	Fecha del pedido	Estado del pedido
Animals Unlimited	933401	09/05/2009	Se envió el 09/29
	934567	09/11/2009	Se envió el 09/21
	934613	09/13/2009	Se envió el 09/21
	934691	09/14/2009	Se envió el 09/21
Bear Bizarre	933603	09/02/2009	Envío parcial
	933668	09/08/2009	Programado para el 10/03
	934552	09/18/2009	Programado para el 10/03
	934683	09/18/2009	Se envió el 09/28
Cuddles Co.	933414	09/12/2009	Se envió el 09/18
	933422	09/14/2009	Se envió el 09/21
	934339	09/16/2009	Se envió el 09/26
	934387	09/18/2009	Se envió el 09/21
	934476	09/25/2009	Pedido pendiente
Stuffed Stuff	934341	09/14/2009	Se envió el 09/26
	934591	09/18/2009	Envío parcial
	934633	09/26/2009	Pedido pendiente
	934664	09/29/2009	Envío parcial

Oprima cualquier tecla para ver el resto de la lista; ESC para finalizar; ? para ayuda
Para obtener más detalle, coloque el cursor sobre el número de pedido y oprima Intro.

FIGURA 11.14

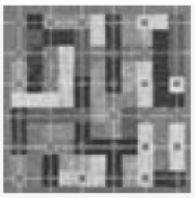
El sitio Web de DinoTech utiliza al máximo los vínculos, fuentes RSS, suscripciones de video y anuncios de pancarta.



Término Web	Significado
Ajax	Un método que utiliza JavaScript y XML para modificar páginas Web en forma dinámica, sin mostrar una nueva página, mediante la obtención de pequeñas cantidades de datos del servidor.
CSS	Hojas de estilo en cascada, un conjunto de estilos que controlan el formato de una página Web. Los estilos CSS se pueden almacenar en un archivo y usar para dar formato a varias páginas Web, o se pueden definir dentro de una página Web.
DHTML	HTML dinámico, una forma de combinar JavaScript y tal vez las hojas de estilo en cascada para hacer que la página Web cambie con base en las acciones del usuario.
FAQ	Preguntas frecuentes. Con frecuencia los sitios Web tienen una página dedicada a estas preguntas frecuentes, para que la fuerza de ventas de la empresa o el equipo de soporte técnico no se vean inundados con las mismas preguntas una y otra vez; además los usuarios pueden tener acceso a las respuestas las 24 horas del día.
FTP	Protocolo de transferencia de archivos, en la actualidad es la forma más común de mover archivos entre distintos sistemas computarizados.
GIF	Formato de intercambio de gráficos, un formato popular de imágenes comprimidas que se adapta muy bien al material gráfico.
Java	Un lenguaje orientado a objetos que permite ejecutar aplicaciones dinámicas en Internet. Los que no son programadores pueden usar paquetes de software tales como Visual Café para Java de Symantec.
JPEG	Grupo unido de expertos en fotografía, el acrónimo para un popular formato de imágenes comprimidas que se adapta muy bien a las fotografías; el diseñador puede ajustar la calidad de estas fotografías.
HTML	Lenguaje de marcado de hipertexto, el lenguaje detrás de la apariencia de los documentos en Web. En realidad es un conjunto de convenciones que marcan las porciones de un documento para indicarle al navegador qué formato distintivo debe aparecer en cada porción de una página.
http://	Protocolo de transferencia de hipertexto, se utiliza para mover páginas Web entre distintas computadoras; por ejemplo, desde un sitio Web en una computadora que esté en otro país, hasta su computadora personal.
PHP	Un lenguaje de programación de código fuente abierto, que a menudo se utiliza con MySQL, un sistema de administración de bases de datos.
complementos o plug-ins	Software adicional (a menudo desarrollado por un tercero) que se puede utilizar con otro programa; por ejemplo, Real Player de RealNetworks o Macromedia Flash se utilizan como complementos en los navegadores Web para reproducir audio o video de flujo continuo y ver animaciones basadas en vectores.
URL	Localizador uniforme de recursos, la dirección de un documento o programa en Internet. Las extensiones conocidas son .com para comercios, .edu para instituciones educativas, .gov o .gob para instituciones gubernamentales, .org para organizaciones, etcétera.
VB .NET	Visual Basic .NET, un entorno de programación de Microsoft.
Webmaster	La persona responsable de mantener el sitio Web.
WMP	Windows media photo, una alternativa a JPEG desarrollada por Microsoft.

FIGURA 11.11
Términos del vocabulario Web.

DISEÑO DE LA ENTRADA



Los usuarios merecen una salida de calidad. En buena medida, la calidad de la entrada del sistema determina la calidad de su salida. Es vital que los formularios de entrada, pantallas y documentos Web interactivos se diseñen teniendo en cuenta esta relación crítica.

Los formularios de entrada, las pantallas y los formularios interactivos para llenar a través de la Web bien diseñados deben cumplir con los objetivos de efectividad, precisión, facilidad de uso, consistencia, simpleza y atracción. Podemos alcanzar todos estos objetivos debemos apegarnos a los principios básicos de diseño, conocer lo que se requiere como entrada para el sistema y comprender la forma en que los usuarios responden a los distintos elementos de los formularios y pantallas.

La efectividad significa que los formularios de entrada, las pantallas de entrada y los formularios a llenar en Web sirvan a propósitos específicos para los usuarios del sistema de información, mientras que la precisión se refiere a la certeza que el diseño proporcione para el llenado adecuado. La facilidad de uso significa que los formularios y pantallas sean simples, de manera tal que el usuario requiera el mínimo tiempo posible para descifrar su estructura. La consistencia implica que todos los formularios de entrada, sean pantallas de entrada o formularios a llenar en Web, agrupen los datos de manera similar de una aplicación a otra, mientras que simpleza se refiere a mantener esos mismos diseños ordenados de una manera que el usuario enfoque su atención en ellos. En cuanto a la atracción, significa que los usuarios disfruten al usar los formularios de entrada debido a su diseño agradable.

DISEÑO DE BUENOS FORMULARIOS

El analista de sistemas debe ser capaz de diseñar un formulario completo y útil; los formularios innecesarios desperdician los recursos de una organización.

Los formularios son instrumentos importantes para dirigir el curso de trabajo. Son documentos impresos que las personas deben llenar de manera estandarizada. Los formularios solicitan y capturan la información requerida por los miembros de la organización que, a menudo, se introducirá en la computadora; por medio de este proceso, sirven como documentos fuente para los usuarios o constituyen el origen de los datos que los humanos deben introducir en aplicaciones de comercio electrónico.

Para diseñar formularios útiles debemos tener en cuenta los siguientes cuatro lineamientos de diseño de formularios:

1. Que los formularios sean fáciles de llenar.
2. Que cumplan con el propósito para el que se diseñaron.
3. Que su diseño contribuya a que se completen con precisión.
4. Que sean atractivos.

En las siguientes secciones veremos cada uno de estos cuatro lineamientos por separado.

Hacer que los formularios sean fáciles de llenar

Para reducir los errores, agilizar el proceso de completarlos y facilitar la introducción de los datos, es esencial que los formularios sean fáciles de llenar. El costo de los formularios es mínimo si se compara con el del tiempo que invierten los empleados en llenarlos y después en introducir los datos en el sistema de información. Con frecuencia es posible eliminar el segundo paso mediante el uso del envío electrónico, en el que los mismos usuarios teclean los datos, desde los sitios Web establecidos para transacciones de comercio electrónico o solicitud de información.

FLUJO DEL FORMULARIO Un formulario con un flujo apropiado reduce el tiempo y esfuerzo que invierten los empleados en llenarlo. Los formularios deben fluir de izquierda a derecha y de arriba abajo. Un flujo ilógico requiere tiempo adicional y es frustrante. Un formulario en el que las personas tienen que pasar directamente a la parte inferior y después saltar hacia la parte superior para terminarlo exhibe un mal flujo.

SIETE SECCIONES DE UN FORMULARIO Hay un segundo método que facilita a las personas llenar los formularios en forma correcta; éste se enfoca en agrupar la información en forma lógica. Las siete secciones principales de un formulario son:

1. Encabezado.
2. Identificación y acceso.
3. Instrucciones.
4. Cuerpo.
5. Firma y verificación.
6. Totales.
7. Comentarios.

Lo ideal es que estas secciones aparezcan agrupadas como en el recibo de gastos de empleado de Bakerloo Brothers, de la figura 12.1. Observe que las siete secciones abarcan la información básica requerida en la mayoría de los formularios. El tercio superior del formulario se dedica a tres secciones: encabezado, sección de identificación y acceso, y sección de instrucciones.

LEYENDAS El uso de leyendas claras es otra técnica para facilitar el trabajo de llenar un formulario. Las leyendas indican a la persona que completa el formulario lo que debe poner en una línea, espacio o cuadro en blanco. En la figura 12.2 se muestran varias opciones para las leyendas. Se muestran dos tipos de leyendas de línea, dos tipos de leyendas de verificación y ejemplos de una leyenda enmarcada y una leyenda de tabla.

La ventaja de poner la leyenda debajo de la línea es que hay más espacio en la línea para los datos; la desventaja es que algunas veces no está claro cuál línea está asociada con la leyenda, si la de arriba o la de abajo.

Las leyendas de línea pueden estar a la izquierda de los espacios en blanco, a la misma altura, o se pueden imprimir debajo de la línea en la que se van a introducir los datos.

Otra forma de usar leyendas es proveer un cuadro para los datos en vez de una línea. Las leyendas se pueden colocar en el interior, encima o debajo del cuadro. Los cuadros en los formularios ayudan a las personas a introducir los datos en el lugar correcto y facilitan a la persona que recibe el formulario el proceso de leerlo. La leyenda debe usar un tamaño de letra pequeño, de manera que no domine el área de entrada. Es posible incluir pequeñas marcas de verificación en el cuadro si los datos se van a introducir en un sistema computarizado. Si no hay suficiente espacio en un registro para los datos, la persona que llena el formulario (en vez del operador que introducirá los datos) tiene la libertad de determinar cómo se deben abreviar los datos. Las leyendas también pueden incluir pequeñas notas aclaratorias para ayudar al usuario a introducir la información correctamente, como Fecha (MM/DD/AAAA) o Nombre completo (Apellido paterno, apellido materno, Nombre(s)).

No importan los estilos de leyendas de línea que se elijan, es imprescindible emplearlos de manera consistente (por ejemplo, sería confuso tratar de llenar un formulario diseñado con leyendas tanto arriba como debajo de las líneas).

Las leyendas de verificación son una mejor opción cuando es necesario restringir las opciones de respuesta. Observe la lista de métodos de viaje que se muestran para el ejemplo de verificación vertical en la figura anterior. Si se va a rembolsar los gastos del empleado en el viaje de negocios sólo para los métodos de viaje de la lista, un sistema de verificación es más conveniente que una línea en blanco. Este método tiene la ventaja adicional de recordar a la persona que verifica los datos que debe buscar un talón de boleto de avión o cualquier otro recibo relacionado.

Cumplir con el propósito previsto

Los formularios se crean para servir a uno o más propósitos en los procesos de registrar, procesar, almacenar y recuperar la información para las empresas. Algunas veces es conveniente proveer distinta información a los

diferentes departamentos o usuarios y compartir al mismo tiempo cierta información básica. Aquí es donde son útiles los formularios especializados.

El término *formulario especializado* también se puede referir sólo a la manera en que la papelería prepara los formularios. Algunos ejemplos de formularios especiales de papelería son los formularios de tantos múltiples que se utilizan para crear triplicados instantáneos de los datos, los formularios de alimentación continua que pueden pasar por la impresora sin necesidad de intervención humana y los formularios perforados que dejan un talón como registro cuando se separan.

Asegurar que se llenen en forma precisa

Por lo general, las tasas de errores asociadas con la recolección de los datos disminuyen de manera considerable cuando los formularios se diseñan de tal forma que sea obvia la manera en que deben completarse con precisión. El diseño es importante para asegurar que las personas hagan lo correcto con el formulario siempre que lo utilicen. Cuando los empleados de servicio, como los que leen medidores o toman inventario, usan dispositivos portátiles para escanear o teclear los datos en el sitio apropiado, se evita el paso adicional de tener que transcribir los datos para introducirlos en la computadora. Los dispositivos portátiles utilizan la transmisión inalámbrica o se conectan a los sistemas computarizados más grandes para poder descargar los datos que el trabajador de servicio almacenó en ellos. No es necesario transcribir lo que ocurrió en el campo.

Mantener los formularios atractivos

Aunque dotar a los formularios de atractivo se deja al último, esto no significa que sea menos importante: se hace de esta forma debido a que para producir formularios atractivos hay que aplicar las técnicas descritas en las secciones anteriores. Los formularios estéticos atraen a las personas y éstas se sienten animadas para completarlos.

Los formularios deben lucir ordenados. Para ser atractivos, deben solicitar la información en el orden esperado: la convención dicta que se debe pedir nombre, dirección, ciudad, estado y código postal (y país, de ser necesario). La distribución y el flujo apropiados contribuyen al atractivo de un formulario.

Utilizar distintos tipos de letra en el mismo formulario puede ser útil para que a los usuarios se les haga más atractivo llenarlo. Separar las categorías y subcategorías con líneas gruesas y delgadas también puede fomentar el interés. Los tipos de letras y los grosores de las líneas son elementos de diseño útiles para capturar la atención y hacer que las personas se sientan con la seguridad de que llenan el formulario en forma correcta.

Hay paquetes de diseño de formularios disponibles para PC. La figura 12.3 muestra cómo se puede crear formularios mediante el uso de software que permite al analista automatizar con rapidez los procesos de negocios para los que ya existen formularios en papel. El analista puede usar un conjunto de herramientas para configurar campos, casillas de verificación, líneas, cuadros y muchas otras características. También es posible digitalizar formularios en papel para después publicarlos en la Web.

Cómo controlar los formularios de las empresas

Controlar los formularios de las empresas es una tarea importante; a menudo cuentan con un especialista en formularios que se encarga de llevar el control, pero algunas veces este trabajo recae en el analista de sistemas, quien se encarga de establecer e implementar el control de los formularios.

Las tareas básicas para controlar formularios incluyen asegurar que cada formulario en uso cumpla con su propósito específico para ayudar a los trabajadores a realizar sus tareas y que el propósito especificado sea integral para el funcionamiento de la organización, de manera que se evite la duplicación de la información recopilada y de los formularios correspondientes; diseñar formularios efectivos, decidir sobre cómo reproducirlos en la forma más económica posible y establecer procedimientos que los pongan a disposición de los empleados (cuando sea necesario) al menor costo posible. A menudo, para esto se requiere que los formularios estén disponibles en Web para poder imprimirlos. Hay que incluir en cada formulario un número único y una fecha de revisión (mes/año), independientemente de que se llene y envíe en forma manual o electrónica. Esto ayuda a que los usuarios sean organizados y eficientes.