

***Guía para el Examen General  
de Egreso de la Licenciatura  
en Ingeniería de Software***

***EGEL-ISOFT  
CON  
PRÁCTICAS DE EXAMEN  
Y  
ANTOLOGÍA***

# ÍNDICE

PRESENTACIÓN

PROPÓSITO Y ALCANCE DEL EGEL-ISOFT

DESTINATARIOS DEL EGEL-ISOFT

¿CÓMO SE CONSTRUYE EL EGEL-ISOFT?

CARACTERÍSTICAS DEL EGEL-ISOFT

¿QUÉ EVALÚA EL EGEL-ISOFT?

¿Qué tipo de preguntas se incluyen en el examen?

1. Preguntas o reactivos de cuestionamiento directo

2. Ordenamiento

3. Elección de elementos

4. Relación de columnas

PRIMERA PRÁCTICA DE EXAMEN

RESPUESTAS DE LA PRIMERA PRÁCTICA DE EXAMEN

SEGUNDA PRÁCTICA DE EXAMEN

RESPUESTAS DE LA SEGUNDA PRÁCTICA DE EXAMEN

TERCERA PRÁCTICA DE EXAMEN

RESPUESTAS DE LA TERCERA PRÁCTICA DE EXAMEN

CUARTA PRÁCTICA DE EXAMEN

RESPUESTAS DE LA CUARTA PRÁCTICA DE EXAMEN

ANTOLOGÍA

Lectura 1. Principios que guían la práctica

Lectura 2. Comprensión de los requerimientos

Lectura 3. Modelado de los requerimientos: flujos

Lectura 4. Modelado de los requerimientos: escenarios

Lectura 5. Diseño e implementación 1

Lectura 6. Diseño e implementación 2

Lectura 7. Gestión de proyectos

Lectura 8. Gestión de la calidad

Lectura 9. Arquitectura de redes de información

Lectura 10. Modelos de gestión de red

Lectura 11. Gestión y planificación de redes

Lectura 12. Gestión de Redes

Lectura 13. SISTEMAS OPERATIVOS



## **PRESENTACIÓN**

Esta Guía está dirigida a quienes sustentarán el Examen General para el Egreso de la Licenciatura en Ingeniería de Software (EGEL-ISOFT). Su propósito es ofrecer información que permita a los sustentantes familiarizarse con las principales características del examen, los contenidos que se evalúan, el tipo de preguntas (reactivos) que encontrarán en el examen, así como con algunas sugerencias de estudio y de preparación para presentar el examen.

## **PROPÓSITO Y ALCANCE DEL EGEL-ISOFT**

El propósito del EGEL-ISOFT es identificar si los egresados de la licenciatura en Ingeniería de Software cuentan con los conocimientos y habilidades necesarios para iniciarse eficazmente en el ejercicio de la profesión. La información que ofrece permite al sustentante:

- Conocer el resultado de su formación en relación con un estándar de alcance nacional mediante la aplicación de un examen confiable y válido, probado con egresados de instituciones de educación superior (IES) de todo el país.
- Conocer el resultado de la evaluación en cada área del examen, por lo que puede ubicar aquéllas donde tiene un buen desempeño, así como aquéllas en las que presenta debilidades.
- Beneficiarse curricularmente al contar con un elemento adicional para integrarse al mercado laboral.

## DESTINATARIOS DEL EGEL-ISOFT

Está dirigido a los egresados de la licenciatura en Ingeniería de Software, que hayan cubierto el 100% de los créditos, estén o no titulados, y en su caso a estudiantes que cursan el último semestre de la carrera, siempre y cuando la institución formadora así lo solicite.

El EGEL-ISOFT se redactó en idioma español, por lo que está dirigido a individuos que puedan realizar esta evaluación bajo dicha condición lingüística. Los sustentantes con necesidades físicas especiales serán atendidos en función de su requerimiento especial.

## ¿CÓMO SE CONSTRUYE EL EGEL-ISOFT?

Con el propósito de asegurar pertinencia y validez en los instrumentos de evaluación, el Ceneval se apoya en Consejos Técnicos integrados por expertos en las áreas que conforman la profesión, los cuales pueden representar a diferentes instituciones educativas, colegios o asociaciones de profesionistas, instancias empleadoras del sector público, privado y de carácter independiente. Estos Consejos Técnicos funcionan de acuerdo con un reglamento y se renuevan periódicamente.

El contenido del EGEL-ISOFT es el resultado de un complejo proceso metodológico, técnico y de construcción de consensos en el Consejo Técnico y en sus Comités Académicos de apoyo en torno a:

- i) La definición de principales funciones o ámbitos de acción del profesional
- ii) La identificación de las diversas actividades que se relacionan con cada ámbito
- iii) La selección de las tareas indispensables para el desarrollo de cada actividad
- iv) Los conocimientos y habilidades requeridos para la realización de esas tareas profesionales
- v) La inclusión de estos conocimientos y habilidades en los planes y programas de estudio vigentes de la licenciatura en Ingeniería de Software

Todo esto tiene como referente fundamental la opinión de centenares de profesionistas activos en el campo de la Ingeniería de Software, formados con planes de estudios diversos y en diferentes instituciones, quienes (en una encuesta nacional) aportaron su punto de vista respecto a:

- i) Las tareas profesionales que se realizan con mayor frecuencia
- ii) El nivel de importancia que estas tareas tienen en el ejercicio de su profesión
- iii) El estudio o no, durante la licenciatura, de los conocimientos y habilidades que son necesarios para la realización de estas tareas



## CARACTERÍSTICAS DEL EGEL-ISOFT

Es un instrumento de evaluación que puede describirse como un examen con los siguientes atributos:

Atributo	Definición
<b>Especializado para la carrera profesional de Ingeniería de Software</b>	Evalúa conocimientos y habilidades específicos de la formación profesional del licenciado en Ingeniería de Software que son críticos para iniciarse en el ejercicio de la profesión. No incluye conocimientos y habilidades profesionales genéricos o transversales.
<b>De alcance nacional</b>	Considera los aspectos esenciales en la licenciatura en Ingeniería de Software para iniciarse en el ejercicio de la profesión en el país. No está referido a un currículo en particular. Se diseñan y preparan para que tengan validez en todo el país.
<b>Estandarizado</b>	Cuenta con reglas fijas de diseño, elaboración, aplicación y calificación.
<b>Criterial</b>	Los resultados de cada sustentante se comparan contra un estándar de desempeño nacional pre establecido por el Consejo Técnico del examen.
<b>Objetivo</b>	Tiene criterios de calificación únicos y precisos, lo cual permite su automatización.
<b>De máximo esfuerzo</b>	Permite establecer el nivel de rendimiento del sustentante, sobre la base de que este hace su mejor esfuerzo al responder los reactivos de la prueba.
<b>De alto impacto</b>	Con base en sus resultados los sustentantes pueden titularse y las IES obtienen un indicador de rendimiento académico.
<b>De opción múltiple</b>	Cada pregunta se acompaña de cuatro opciones de respuesta, entre las cuales sólo una es la correcta.
<b>Contenidos centrados en problemas</b>	Permite determinar si los sustentantes son capaces de utilizar lo aprendido durante su Licenciatura en la resolución de problemas y situaciones a las que típicamente se enfrenta un egresado al inicio del ejercicio profesional.
<b>Sensible a la instrucción</b>	Evalúa resultados de aprendizaje de programas de formación profesional de la licenciatura en Ingeniería de Software, los cuales son una consecuencia de la experiencia educativa institucionalmente organizada.
<b>Contenidos validados socialmente</b>	Contenidos validados por comités de expertos y centenares de profesionistas en ejercicio en el país.

## ¿QUÉ EVALÚA EL EGEL-ISOFT?

El examen está organizado en áreas, subáreas y temas. Las áreas corresponden a ámbitos profesionales en los que actualmente se organiza la labor del ingeniero en software. Las subáreas comprenden las principales actividades profesionales de cada uno de los ámbitos profesionales referidos. Por último, los temas identifican los conocimientos y habilidades necesarios para realizar tareas específicas relacionadas con cada actividad profesional.

### Estructura general del EGEL-ISOFT por áreas y subáreas

Área/ Subárea	% en el examen	Número de reactivos	Distribución de reactivos por sesión	
			1 <sup>a</sup>	2 <sup>a</sup>
<b>A. Análisis de sistemas de información</b>	<b>13.26</b>	<b>24</b>	<b>24</b>	
1. Diagnóstico del problema y valoración de la factibilidad para el desarrollo de sistemas de información	7.18	13	13	
2. Modelado de los requerimientos de un sistema de información	6.08	11	11	
<b>B. Desarrollo e implantación de aplicaciones computacionales</b>	<b>40.88</b>	<b>74</b>	<b>74</b>	
1. Diseño de la solución del problema de tecnología de información	9.39	17	17	
2. Desarrollo de sistemas	22.65	41	41	
3. Implantación de sistemas	3.87	7	7	
4. Aplicación de modelos matemáticos	4.97	9	9	
<b>C. Gestión de proyectos de tecnologías de información</b>	<b>14.36</b>	<b>26</b>		<b>26</b>
1. Administración de proyectos de tecnologías de información	5.52	10		10
2. Control de calidad de proyectos de tecnologías de información	8.84	16		16
<b>D. Implementación de redes, bases de datos, sistemas operativos y lenguaje de desarrollo</b>	<b>31.49</b>	<b>57</b>		<b>57</b>
1. Gestión de redes de datos	8.84	16		16
2. Gestión de bases de datos	12.15	22		22
3. Gestión de sistemas operativos o lenguajes de desarrollo	10.50	19		19
<b>Total</b>	<b>100.00</b>	<b>181</b>	<b>98</b>	<b>83</b>
Estructura aprobada por el Consejo Técnico del EGEL-ISOFT el 27 de junio de 2012.				
*NOTA: Adicionalmente se incluye un 20% de reactivos piloto que no califican.				



## ¿Qué tipo de preguntas se incluyen en el examen?

En el examen se utilizan reactivos o preguntas de opción múltiple que contienen fundamentalmente los siguientes dos elementos:

- **La base** es una pregunta, afirmación, enunciado o gráfico acompañado de una instrucción que plantea un problema explícitamente.
- **Las opciones de respuesta** son enunciados, palabras, cifras o combinaciones de números y letras que guardan relación con la base del reactivo, donde *sólo una* opción es la correcta. Para todas las preguntas del examen **siempre** se presentarán cuatro opciones de respuesta.

Durante el examen usted encontrará diferentes formas de preguntar. En algunos casos se le hace una pregunta directa, en otros se le pide completar una información, algunos le solicitan elegir un orden determinado, otros requieren de usted la elección de elementos de una lista dada y otros más le piden relacionar columnas. Comprender estos formatos le permitirá llegar mejor preparado al examen. Con el fin de apoyarlo para facilitar su comprensión, a continuación se presentan algunos ejemplos.

### 1. Preguntas o reactivos de cuestionamiento directo

En este tipo de reactivos el sustentante tiene que seleccionar una de las cuatro opciones de respuestas a partir del criterio o acción que se solicite en el enunciado, afirmativo o interrogativo, que se presenta en la base del reactivo.

*Ejemplo correspondiente al área de Análisis de sistemas:*

El área de desarrollo de sistemas de una empresa requiere implementar un sistema de información en todas sus sucursales. Se están evaluando las siguientes alternativas para resolver ese requerimiento:

1. El costo del desarrollo externo es en promedio de \$1,300,000.00 y cubre el 100% de los requerimientos.
2. El desarrollo interno para cubrir el 100% de los requerimientos implica 6 meses de trabajo y el sistema resultante puede ser vendido
3. Adquirir un software comercial cuyo costo es de \$700,000.00 y cubre el 80% de los requerimientos
4. Continuar con el uso de los sistemas de información ocasiona costos de operación y mantenimiento de \$1,000,000

¿Cuál de las siguientes metodologías se aplica para evaluar la factibilidad de las propuestas?

- A) Benchmarking
- B) Costo – beneficio
- C) Análisis operativo
- D) Análisis técnico

### *Argumentación de las opciones de respuesta*

La opción correcta es la B: porque la información proporcionada está asociada al costo de las propuestas.

Las otras opciones son incorrectas porque la información proporcionada no refleja el desempeño de las propuestas, no da elementos para establecer la factibilidad operativa del sistema y no es de tipo técnica.

### *Ejemplo correspondiente al área de Desarrollo e implantación de aplicaciones computacionales:*

Una incubadora de negocios está organizando un proyecto para producir un videojuego de caracteres que se desarrollará en varias fases. El cliente especifica los requerimientos en etapas posteriores a cada demostración del producto. Las primeras versiones tienen propósitos académicos y se espera que las últimas sean productos comerciales. ¿Qué modelo del proceso se utiliza para desarrollar este proyecto?

- A) Lineal
- B) En espiral
- C) Incremental
- D) De prototipos

### *Argumentación de las opciones de respuesta*

La opción **correcta** es la A: porque Benchmarking implica comparar las prácticas reales o planificadas del proyecto con aquellas de otros proyectos, a fin de generar ideas para mejorar o para establecer una norma por medio de la cual medir el desempeño de un proyecto.

Las otras opciones son incorrectas porque análisis Costo/Beneficio toma como base los beneficios y los costos que se obtienen de llevar a cabo los proyectos actuales, a menor costo menor beneficio, el diagrama de causa-y-efecto detecta los posibles factores con los problemas potenciales que existen en un proyecto de software y el diseño de experimentos es un método estadístico que ayuda a identificar cuáles son los factores que influyen en variables específicas.



## 2. Ordenamiento

Este tipo de reactivos demandan el ordenamiento o jerarquización de un listado de elementos de acuerdo con un criterio determinado. La tarea del sustentante consiste en seleccionar la opción en la que aparezcan los elementos en el orden solicitado.

*Ejemplo correspondiente al área de Análisis de sistemas:* Ordene secuencialmente los pasos necesarios para preparar una entrevista para la obtención de los requerimientos de una aplicación computacional.

1. Decidir el tipo de preguntas y la estructura
2. Conocer los antecedentes de la organización
3. Decidir a quién entrevistar
4. Establecer los objetivos de la entrevista

- A) 2, 3, 1, 4
- B) 2, 4, 3, 1
- C) 3, 4, 2, 1
- D) 3, 2, 1, 4

### *Argumentación de las opciones de respuesta*

La opción **correcta** es la **B**: porque la secuencia de pasos adecuada para preparar una entrevista es; conocer antecedentes de la organización, establecer objetivos, decidir a quién entrevistar y decidir el tipo de preguntas y su estructura. El producto de cada paso es un insumo necesario para el siguiente.

Las otras opciones son incorrectas porque no cumplen con el orden adecuado.

*Ejemplo correspondiente al área de Desarrollo e implantación de aplicaciones computacionales:*

Ordene los pasos que se requieren para elaborar el diagrama relacional a partir de un diagrama entidad relación de un modelo de datos.

1. Elaborar por cada una de las relaciones con cardinalidad muchos a muchos una relación asociativa
2. Elaborar por cada una de las entidades del diagrama ER una relación en el diagrama relacional
3. Reducción de las relaciones muchos a uno con el paso de llaves
4. Fusionar las entidades con relaciones de cardinalidad uno a uno

- A) 2, 1, 3, 4
- B) 2, 3, 4, 1
- C) 4, 3, 1, 2
- D) 4, 2, 1, 3

### *Argumentación de las opciones de respuesta*

La opción **correcta** es la **D**: porque son los pasos que se requieren para elaborar el diagrama relacional a partir de un diagrama entidad relación de un modelo de datos.

Las otras opciones son incorrectas porque no cumplen con el orden adecuado.

**Esta guía de estudio, fue vendida por [www.guiaaceneval.mx](http://www.guiaaceneval.mx) Todos los Derechos Reservados.**

**Ejemplo correspondiente al área de Liderazgo de proyectos de tecnologías de información:**

El departamento de control de calidad de una empresa de consultoría, implementa un plan de aseguramiento de calidad como un mecanismo de control. Ordene las actividades de dicho plan.

1. Desarrollar la descripción del proceso de software
  2. Preparar el plan de SQA
  3. Registrar cualquier falta de ajuste para informar al gestor ejecutivo
  4. Garantizar que estén documentadas las desviaciones
  5. Auditarse productos de trabajo de software para verificar que se ajusten con los requerimientos
- A) 1, 2, 4, 3, 5  
B) 1, 3, 2, 5, 4  
C) 2, 1, 5, 4, 3  
D) 2, 5, 1, 3, 4

*Argumentación de las opciones de respuesta*

La opción **correcta** es la **C**: porque es la secuencia idónea preparar un plan, desarrollar la descripción, auditarse los productos, garantizar que estén documentadas las desviaciones y, finalmente, registrar cualquier falta de ajuste para informar al gestor ejecutivo.

Las otras opciones son incorrectas porque no cumplen con el orden adecuado.

### 3. Elección de elementos

A partir de un criterio, se seleccionan elementos que forman parte de un conjunto incluido en la base. En las opciones de respuesta se presentan subconjuntos.

**Ejemplo correspondiente al área de Implantación de infraestructura tecnológica:** De los siguientes protocolos, ¿cuáles corresponden a la capa de red en el modelo OSI?

1. HDLC
2. IP
3. TCP
4. RIP

- A) 1, 2  
B) 1, 3  
C) 2, 4  
D) 3, 4

*Argumentación de las opciones de respuesta*

La opción **correcta** es la **C**: porque ambos protocolos pertenecen a la capa de red.

Las otras opciones son incorrectas porque HDLC es un protocolo de la capa de enlace de datos y TCP es un protocolo de la capa de transporte.



*Ejemplo correspondiente al área de Desarrollo e implantación de aplicaciones computacionales:*

El departamento de tecnologías de la información de una empresa está a punto de iniciar el desarrollo de una aplicación, considerando los siguientes lenguajes de programación. Seleccione los que sean orientados a objetos.

- 1. LISP
- 2. JAVA
- 3. FORTRAN
- 4. DELPHI
- 5. PHP

- A) 1, 2, 3
- B) 1, 3, 4
- C) 2, 4, 5
- D) 3, 4, 5

*Argumentación de las opciones de respuesta*

La opción **correcta** es la **C**: porque los tres son lenguajes orientados a objetos.

Las otras opciones son incorrectas porque LISP y Fortran son lenguajes de alto nivel, pero no orientados a objetos.

## 4. Relación de columnas

En este tipo de reactivos hay dos columnas, cada una con contenidos distintos, que el sustentante tiene que relacionar de acuerdo con el criterio especificado en la base del reactivo:

*Ejemplo correspondiente al área de **Implantación de infraestructura tecnológica**:*

Un equipo de desarrollo tiene alojado el sistema de control de versiones en una carpeta compartida en un servidor de la empresa. Para su operación, esta carpeta tiene asignados diferentes permisos para diferentes usuarios, de acuerdo con el uso que hace cada uno. Así, un desarrollador baja versiones nuevas y sube actualizaciones; el personal de soporte técnico utiliza controladores y actualizaciones del sistema; mientras que el líder de proyecto controla y gestiona el sistema de control de versiones, además de participar como desarrollador. Relacione los perfiles de usuario con sus respectivos permisos de acceso.

Perfil de usuario	Permisos de acceso
1. Desarrollador	a) Creación, eliminación
2. Líder de proyecto	b) Lectura
3. Personal de soporte técnico	c) Lectura, creación, eliminación d) Lectura, escritura e) Lectura, escritura, creación f) Lectura, escritura, creación, eliminación
A) 1a, 2c, 3e	
B) 1b, 2d, 3e	
C) 1d, 2b, 3a	
D) 1e, 2f, 3b	

*Argumentación de las opciones de respuesta*

La opción **correcta** es la **D**: porque todos los perfiles tienen asignados los permisos apropiados a su función.

Las otras opciones son incorrectas porque el desarrollador debe poder leer y escribir, además de que no es conveniente que pueda eliminar y el líder de proyecto debe tener todos los permisos.



## **PRIMERA PRÁCTICA DE EXAMEN**

1. ¿Cuáles son las características del software?

- a. El software está desarrollado o diseñado; No se fabrica en el sentido clásico.
- b. El software no se "desgasta".
- c. El software puede ser personalizado o personalizac.
- d. Todo lo mencionado anteriormente

2. Los compiladores, los editores de software vienen bajo qué tipo de software?

- a. Software del sistema
- b. Software de la aplicacion
- c. Software científico
- d. Ninguna de las anteriores

3. El software se define como \_\_\_\_\_.

- a. Instrucciones
- b. Estructuras de datos
- c. Documentos
- d. Todas las anteriores

4. ¿Cuáles son las señales de que un proyecto de software está en problemas?

- a. El alcance del producto está mal definic.
- b. Los plazos son poco realistas.
- c. Los cambios se gestionan mal.
- d. Todas las anteriores.

5. Usted está trabajando como gerente de proyecto. Tu empresa quiere desarrollar un proyecto.

Usted también está involucrado en el equipo de planificación. ¿Cuál será tu primer paso en la planificación de proyectos?

- a. Establecer los objetivos y alcance del producto.
- b. Determine las restricciones del proyecto.
- c. Selecciona el equipo.
- d. Ninguna de las anteriores.

6. ¿Qué elemento de codificación se omite generalmente al final de la línea?

- a. Convenciones de nombres
- b. Identificando
- c. Espacio en blanco
- d. Los operadores

7. Las reglas para escribir 'if-then-else', 'case-switch', 'while-until' y 'for' flujo de control se denominan \_\_\_\_\_.

- a. Comentarios
- b. Funciones
- c. Longitud de línea y envoltura
- d. Estructura de control

8. Haga coincidir la Lista 1 con la Lista 2 y elija la opción correcta.

- 1. Elicitación de requisitos
- 2. Diseño
- 3. Implementación
- 4. Mantenimiento

- a. Módulo de desarrollo e integración
- b. Análisis.
- c. Estructura y comportamiento
- d. La optimización del rendimiento.

- a. 1-c, 2-a, 3-d, 4-b
- b. 1-c, 2-a, 3-b, 4-d
- c. 1-a, 2-c, 3-d, 4-b
- d. 1-b, 2-c, 3-a, 4-d

9. Un proyecto puede ser caracterizado como \_\_\_\_\_.

- a. Cada proyecto puede no tener un objetivo único y distinto.
- b. El proyecto es una actividad de rutina o operaciones diarias.
- c. El proyecto no viene con una hora de inicio y finalización.
- d. Ninguna de las anteriores.

10. Elija la opción correcta en términos de problemas relacionados con la responsabilidad profesional.

- a. Confidencialidad
- b. Derechos de propiedad intelectual
- c. Ambos a y b
- d. Gestionando las relaciones con los clientes



11. "Los ingenieros de software no deben usar sus habilidades técnicas para hacer un mal uso de las computadoras de otras personas". Aquí el término uso incorrecto se refiere a:

- a. Acceso no autorizado a material informático
- b. Modificación no autorizada de material informático.
- c. Diseminación de virus u otro malwad.
- d. Todos los mencionados

12. Explique qué se entiende por PRODUCTO con referencia a uno de los ocho principios según el Código de ética de ACM / IEEE.

- a. El producto debe ser fácil de usar.
- b. Los ingenieros de software se asegurarán de que sus productos y las modificaciones relacionadas cumplan con los estándares profesionales más altos posibles.
- c. Los ingenieros de software se asegurarán de que sus productos y las modificaciones relacionadas satisfagan al cliente.
- d. Significa que el producto diseñado / creado debe estar fácilmente disponible.

13. Identifique un dilema ético de las situaciones mencionadas a continuación:

- a. Su empleador lanza un sistema crítico para la seguridad sin terminar la prueba del sistema.
- b. Negarse a emprender un proyecto.
- c. Acuerdo en principio con las políticas de la alta dirección.
- d. Ninguna de las anteriores

14. Identifique la afirmación correcta: "Los ingenieros de software

- a. actuar de una manera que sea en el mejor interés de su experiencia y favor.
- b. actuar consistentemente con el interés público.
- c. Asegurar que sus productos solo cumplan con el SRS.
- d. Ninguna de las anteriores

15. Seleccione la declaración incorrecta:

Los ingenieros de software deben\_\_\_\_\_.

- a. No a sabiendas acepte trabajos que estén fuera de su competencia.
- b. No uses tus habilidades técnicas para hacer mal uso de las computadoras de otras personas.
- c. Ser dependientes de sus colegas.
- d. Mantener la integridad y la independencia en su juicio profesional.

16. La eficiencia en un producto de software no incluye \_\_\_\_\_

- a. sensibilidad
- b. licenciamiento
- c. utilización de memoria
- d. Tiempo de procesamiento

17. ¿Cuál de estos no se encuentra entre los ocho principios seguidos por el Código de ética y práctica profesional de ingeniería de software?

- a. PÚBLICO
- b. PROFESIÓN
- c. PRODUCTO
- d. AMBIENTE

18. ¿Qué es un software?

- a. El software es un conjunto de programas.
- b. El software es documentación y configuración de datos.
- c. Tanto a como B
- d. Ninguno de los mencionados

19. ¿Cuál de estos no tiene en cuenta la falla del software?

- a. Aumento de la demanda
- b. Baja expectativa
- c. Aumento de la oferta
- d. Menos confiable y costoso

20. ¿Cuáles son los atributos de un buen software?

- a. Mantenimiento del software
- b. Funcionalidad de software
- c. Desarrollo de software
- d. a y B



21. ¿Cuál de estas actividades de ingeniería de software no forma parte de los procesos de software?

- a. Dependencia de software
- b. Desarrollo de software
- c. Validación de software
- d. Especificación de software

22. ¿Cuál de estos es incorrecto?

- a. La ingeniería de software pertenece a la informática.
- b. La ingeniería de software es una parte de la forma más general de la ingeniería de sistemas.
- c. La informática pertenece a la ingeniería de software.
- d. La ingeniería de software se ocupa de los aspectos prácticos de desarrollar y entregar software útil.

23. ¿Cuál de estos es verdad?

- a. Los productos genéricos y los productos personalizados son tipos de productos de software.
- b. Los productos genéricos se producen por organización y se venden al mercado abierto.
- c. Los productos personalizados son encargados por clientes particulares.
- d. Todas las anteriores.

24. ¿Cuál de estos no afecta a los diferentes tipos de software en su conjunto?

- a. Heterogeneidad
- b. Flexibilidad
- c. Negocios y cambio social.
- d. Seguridad

25. ¿Las nociones fundamentales de la ingeniería de software no tienen en cuenta?

- a. Procesos de software
- b. Seguridad del software
- c. Reutilización de software
- d. Validación de Software

26. ¿Cuál de estos no es cierto?

- a. La web ha llevado a la disponibilidad de servicios de software y la posibilidad de desarrollar sistemas basados en servicios altamente distribuidos.
- b. Los sistemas basados en web han conducido a la degradación de los lenguajes de programación.
- c. Web trae concepto de software como servicio.
- d. El sistema basado en la web debe ser desarrollado y entregado de manera incremental.

27. Identifique un lenguaje de cuarta generación (4GL) a partir de lo que se indica a continuación.

- a. FORTRAN
- b. COBOL
- c. Shell de unix
- d. C ++

28. Organice las siguientes actividades para hacer un producto de software utilizando 4GT.

- i. Estrategia de diseño
  - ii. Transformación en producto.
  - iii. Implementación
  - iv. Recolección de requisitos
- a. 1, 4, 3, 2
  - b. 4, 3, 1, 2
  - c. 4, 1, 3, 2
  - d. 1, 3, 4, 2

29. 4GL es un ejemplo de procesamiento \_\_\_\_\_.

- a. Caja blanca
- b. Caja negra
- c. Funcional
- d. Ambos b & c

30. El modelo 4GT es un paquete de \_\_\_\_\_.

- a. Herramientas de caja
- b. Herramientas de software
- c. Programas de software
- d. Documentación



31. Modelo de proceso del software RAD significa \_\_\_\_\_.

- a. Desarrollo rápido de aplicaciones.
- b. Desarrollo relativo de aplicaciones.
- c. Diseño de aplicaciones rápidas.
- d. Desarrollo de aplicaciones recientes.

32. ¿Cuál es el modelo más simple de paradigma de desarrollo de software?

- a. Modelo espiral
- b. Modelo de Big Bang
- c. Modelo V
- d. Modelo de cascada

33. El modelo arquitectónico se deriva de cuál de estas fuentes?

- A) Información sobre el dominio de la aplicación para el software que se construirá;
- B) Elementos del modelo de requisitos específicos, como diagramas de flujo de datos o clases de análisis, sus relaciones y colaboraciones para el problema en cuestión;
- C) La disponibilidad de estilos y patrones arquitectónicos.

- a. Tanto A y B
- b. Ambos B y C
- c. Ambos A y C
- d. Todo lo mencionado anteriormente

34. ¿Qué modelo también se conoce como modelo de verificación y validación?

- a. Modelo de cascada
- b. Modelo de Big Bang
- c. Modelo V
- d. Modelo espiral

35. ¿Qué modelo no es adecuado para grandes proyectos de software pero es bueno para aprender y experimentar?

- a. Modelo de Big Bang
- b. Modelo espiral
- c. Modelo iterativo
- d. Modelo de cascada

36. ¿A qué modelo también se le llama ciclo de vida clásico o modelo de cascada?

- a. Desarrollo iterativo
- b. Desarrollo secuencial lineal
- c. Modelo RAD
- d. Desarrollo incremental

37. ¿Cuál es la tasa de actividad global efectiva promedio en un sistema de tipo E en evolución que es invariante durante la vida útil del producto?

- a. Autorregulación
- b. Reduciendo la calidad
- c. Sistemas de retroalimentación
- d. Estabilidad organizacional

38. Build & Fix Model es adecuado para los ejercicios de programación de \_\_\_\_\_ LOC (línea de código).

- a. 100-200
- b. 200-400
- c. 400-1000
- d. por encima de 1000

39. ¿Cuál de los siguientes modelos no es adecuado para adaptarse a cualquier cambio?

- a. Construir y arreglar el modelo
- b. Modelo de prototipos
- c. Modelo RAD
- d. Modelo de cascada

40. ¿Cuál no es uno de los tipos de prototipo de Modelo de Prototipos?

- a. Prototipo horizontal
- b. Prototipo vertical
- c. Prototipo diagonal
- d. Prototipo de dominio



41. ¿Cuál de los siguientes no es una fase del modelo de prototipos?

- a. Diseño rapido
- b. Codificación
- c. Refinamiento de prototipos
- d. Producto de ingeniero

42. ¿Cuál de las siguientes afirmaciones con respecto al modelo de construcción y reparación es incorrecta?

- a. No hay espacio para el diseño estructurac.
- b. El código pronto se vuelve no apto para ser arreglado e inmutable
- c. El mantenimiento es prácticamente imposible.
- d. Se adapta bien a grandes proyectos.

43. Modelo RAD tiene

- a. 2 fases
- b. 3 fases
- c. 5 fases
- d. 6 fases

44. ¿Cuál es el mayor inconveniente de usar el modelo RAD?

- a. Se requieren desarrolladores / diseñadores altamente especializados y capacitados.
- b. Aumenta la reutilización de los componentes.
- c. Alienta la retroalimentación de cliente / cliente.
- d. Tanto a y c.

45. SDLC significa

- a. Ciclo de vida del desarrollo de programas
- b. Ciclo de vida de desarrollo de sistemas
- c. Ciclo de vida del diseño de software
- d. Diseño del sistema ciclo de vida

46. ¿Qué modelo se puede seleccionar si el usuario está involucrado en todas las fases de SDLC?

- a. Modelo de cascada
- b. Modelo de prototipos
- c. Modelo RAD
- d. ambos b & c

47. ¿Cuál de los siguientes no es un modelo de proceso evolutivo?

- a. Modelo Espiral WINWIN
- b. Modelo incremental
- c. Modelo de desarrollo concurrente
- d. Todos son modelos de software evolutivos

48. El modelo incremental es el resultado de una combinación de elementos, ¿de qué dos modelos?

- a. Construir y arreglar modelo y modelo de cascada
- b. Modelo lineal y modelo RAD
- c. Modelo lineal y modelo de prototipos.
- d. Modelo de cascada y modelo de RAD

49. ¿Cuál es la principal ventaja de usar el modelo incremental?

- a. El cliente puede responder a cada incremento
- b. Más fácil de probar y depurar
- c. Se utiliza cuando existe la necesidad de llevar un producto al mercado antes de tiempo.
- d. Ambos b & c

50. El modelo espiral fue propuesto originalmente por

- a. IBM
- b. Barry Boehm
- c. Periodista
- d. Royce



51. El modelo en espiral tiene dos dimensiones: \_\_\_\_\_ y \_\_\_\_\_.

- a. diagonal, angular
- b. radial, perpendicular
- c. radial, angular
- d. diagonal, perpendicular

52. ¿En qué se diferencia WINWIN Spiral Model de Spiral Model?

- a. Define las tareas necesarias para definir recursos, líneas de tiempo y otra información relacionada con el proyecto.
- b. Define un conjunto de actividades de negociación al comienzo de cada paso alrededor de la espiral.
- c. Define las tareas requeridas para evaluar los riesgos tanto técnicos como de gestión.
- d. Define las tareas necesarias para construir, probar, instalar y proporcionar soporte al usuario.

53. Identificar la desventaja del modelo espiral.

- a. No funciona bien para proyectos más pequeños.
- b. Gran cantidad de análisis de riesgo
- c. Fuerte aprobación y control de documentación.
- d. La funcionalidad adicional se puede agregar en una fecha posterior

54. ¿En qué se diferencia el modelo incremental del modelo espiral?

- a. El progreso se puede medir para el modelo incremental.
- b. Los requisitos cambiantes se pueden acomodar en el modelo incremental.
- c. Los usuarios pueden ver el sistema temprano en el modelo incremental.
- d. no hay diferencia entre estos dos

55. Si tuviera que crear aplicaciones cliente / servidor, ¿qué modelo elegiría?

- a. Modelo Espiral WINWIN
- b. Modelo espiral
- c. Modelo concurrente
- d. Modelo incremental

56. La selección de un modelo se basa en

- a. Requerimientos
- b. Equipo de desarrollo
- c. Usuarios
- d. Todos los mencionados

57. ¿Qué dos modelos no permiten definir requisitos al principio del ciclo?

- a. Cascada y RAD
- b. Prototipos y Espiral
- c. Prototipado y RAD
- d. Cascada y espiral

58. ¿Cuál del siguiente modelo de ciclo de vida se puede elegir si el equipo de desarrollo tiene menos experiencia en proyectos similares?

- a. Espiral
- b. Cascada
- c. RAD
- d. Modelo de mejora iterativa

59. Si fuera un desarrollador líder de una compañía de software y se le solicita que envíe un proyecto / producto dentro de un marco de tiempo estipulado sin barreras de costo, ¿qué modelo seleccionaría?

- a. Cascada
- b. Espiral
- c. RAD
- d. Incremental

60. ¿Qué dos de los siguientes modelos no podrán dar el resultado deseado si la participación del usuario no está involucrada?

- a. Cascada y espiral
- b. RAD y espiral
- c. RAD y cascada
- d. RAD y prototipado



61. ¿Cuál de los siguientes no está definido en un buen documento de especificación de requisitos de software (SRS)?

- a. Requerimiento funcional
- b. Requisito no funcional
- c. Objetivos de implementación
- d. Algoritmo para la implementación de software

62. ¿Cuál de las siguientes es la comprensión de las limitaciones de los productos de software, los problemas relacionados con el sistema de aprendizaje o los cambios que deben realizarse en los sistemas existentes de antemano, identificando y abordando el impacto del proyecto en la organización y el personal, etc.?

- a. Diseño de software
- b. Estudio de factibilidad
- c. Recolección de requisitos
- d. Análisis del sistema

63. ¿Qué proyecto se emprende como consecuencia de una solicitud específica de un cliente?

- a. Proyectos de desarrollo de conceptos
- b. Proyectos de mejora de aplicaciones
- c. Nuevos proyectos de desarrollo de aplicaciones.
- d. Proyectos de mantenimiento de aplicaciones

64. El proceso de ingeniería de requisitos incluye cuál de estos pasos?

- a. Estudio de factibilidad
- b. Recolección de requisitos
- c. Especificación y validación de requisitos de software
- d. Todo lo mencionado anteriormente

65. La especificación de requisitos de software (SRS) también se conoce como especificación de \_\_\_\_\_.

- a. Prueba de caja blanca
- b. Test de aceptación
- c. Pruebas integradas
- d. Prueba de caja negra

66. ¿En qué proceso de obtención los desarrolladores discuten con el cliente y los usuarios finales y conocen sus expectativas del software?

- a. Recolección de requisitos
- b. Requisitos de organización
- c. Negociación y discusión
- d. Documentación

67. Si los requisitos son fácilmente comprensibles y definidos, ¿qué modelo es el más adecuado?

- a. Modelo espiral
- b. Modelo de cascada
- c. Modelo de prototipado
- d. Ninguna de las anteriores

68. ¿Qué documento crea el analista del sistema después de que se recopilan los requisitos de varios interesados?

- a. Especificación de requisitos de software
- b. Validación de requisitos de software
- c. Estudio de factibilidad
- d. Recolección de requisitos

69. ¿Cuál está enfocado hacia el objetivo de la organización?

- a. Estudio de factibilidad
- b. Recolección de requisitos
- c. Especificación de requisitos de software
- d. Validación de requisitos de software

70. ¿Qué documentación funciona como herramienta clave para el diseñador de software, el desarrollador y su equipo de prueba para llevar a cabo sus tareas respectivas?

- a. Documentación de requisitos
- b. Documentación del usuario
- c. Documentación de diseño de software
- d. Documentación técnica



71. ¿Cuál es el significado de la obtención de requisitos en ingeniería de software?

- a. Recopilación de requisitos.
- b. Comprensión del requisito.
- c. Obteniendo los requerimientos del cliente.
- d. Todas las anteriores.

72. ¿Cuál de las siguientes es / es técnica de estimación de proyecto?

- a. Técnica de estimación empírica.
- b. Técnica de estimación heurística.
- c. Técnica de estimación analítica.
- d. Todas las anteriores.

73. ¿Qué herramientas son útiles en todas las etapas de SDLC, para la recopilación de requisitos para pruebas y documentación?

- a. Herramientas mayúsculas
- b. Herramientas de caja baja
- c. Herramientas de caja integradas
- d. Ninguna de las anteriores

74. ¿Cuáles son los tipos de requisitos de desarrollo de software?

- a. Disponibilidad
- b. Confiabilidad
- c. Usabilidad
- d. Todos los mencionados

75. Seleccione el requisito específico del desarrollador?

- a. Potabilidad
- b. Mantenibilidad
- c. Disponibilidad
- d. Tanto a como B

76. FAST significa

- a. Técnica funcional de especificación de aplicaciones
- b. Técnica de especificación de aplicación rápida
- c. Técnica de especificación de aplicación facilitada
- d. Ninguno de los mencionados

77. QFD significa

- a. diseño de funciones de calidad
- b. desarrollo de la función de calidad
- c. Despliegue de la función de calidad
- d. ninguno de los mencionados

78. Los requisitos del sistema del usuario son las partes de qué documento?

- a. SDD
- b. SRS
- c. DDD
- d. DFD

79. ¿Cuál es uno de los actores más importantes de los siguientes?

- a. Personal de nivel de entrada
- b. Accionista de nivel medio
- c. Gerentes
- d. Usuarios del softwad.

80. ¿Cuál de los siguientes es un requisito funcional?

- a. Mantenibilidad
- b. Portabilidad
- c. Robustez
- d. Ninguno de los mencionados



81. ¿Cuál de los siguientes es un requisito que se ajusta al módulo de un desarrollador?

- a. Disponibilidad
- b. Probabilidad
- c. Usabilidad
- d. Flexibilidad

82. "Consideré un sistema donde, un sensor de calor detecta una intrusión y alerta a la compañía de seguridad". ¿Qué tipo de requisito está proporcionando el sistema?

- a. Funcional
- b. No funcional
- c. Requisito conocido
- d. Ninguna de las anteriores

83. ¿Cuál de las siguientes afirmaciones explica la portabilidad en los requisitos no funcionales?

- a. Es un grado en que el software que se ejecuta en una plataforma se puede convertir fácilmente para ejecutarse en otra plataforma.
- b. Se puede mejorar mediante el uso de idiomas, sistemas operativos y herramientas universalmente disponibles y estandarizadas.
- c. La capacidad del sistema para comportarse de forma coherente de una manera aceptable para el usuario cuando se opera dentro del entorno para el que se diseñó el sistema.
- d. Tanto a como B

84. Elija la declaración incorrecta con respecto al requisito no funcional (NFR).

- a. Enfoque orientado al producto: enfoque en la calidad del sistema (o software)
- b. Enfoque orientado al proceso: enfoque en cómo se pueden usar los NFR en el proceso de diseño
- c. Enfoque cuantitativo: encuentre escalas medibles para los atributos de funcionalidad
- d. Enfoque cualitativo - Estudie varias relaciones entre los objetivos de calidad

85. ¿Cuántos esquemas de clasificación se han desarrollado para los NFR (requisito no funcional)?

- a. Dos
- b. Tres
- c. Cuatro
- d. Cinco

86. Según los componentes de FURPS +, ¿cuál de los siguientes no pertenece a S?

- a. Probabilidad
- b. Eficiencia de velocidad
- c. Utilidad
- d. Instalabilidad

87. ¿Cuáles son las cuatro dimensiones de la fiabilidad?

- a. Usabilidad, Fiabilidad, Seguridad, Flexibilidad.
- b. Disponibilidad, fiabilidad, mantenibilidad, seguridad.
- c. Disponibilidad, fiabilidad, seguridad, seguridad.
- d. Seguridad, Seguridad, Probabilidad, Usabilidad.

88. ¿Cuál es el primer paso de la obtención de requisitos?

- a. Identificando a los interesados
- b. Listado de requisitos
- c. Recopilación de requisitos
- d. Ninguna de las anteriores

89. Comenzando de menos a más importante, elija el orden de las partes interesadas.

- i. Gerentes
  - ii. Personal de nivel de entrada
  - iii. Usuarios
  - iv. Accionista de nivel medio
- a. i, ii, iv, iii
  - b. i, ii, iii, iv
  - c. ii, iv, i, iii
  - d. Ninguna de las anteriores

90. Organice las tareas involucradas en la obtención de requisitos de una manera apropiada.

- i. Consolidación
  - ii. Priorización
  - iii. Recopilación de requisitos
  - iv. Evaluación
- a. iii, i, ii, iv
  - b. iii, iv, ii, i
  - c. iii, ii, iv, i
  - d. ii, iii, iv, i



91. ¿Qué diseño identifica el software como un sistema con muchos componentes que interactúan entre sí?

- a. Diseño arquitectónico
- b. Diseño de alto nivel
- c. Diseño detallado
- d. Ambos B y C

92. La herramienta CASE significa \_\_\_\_\_.

- a. Ingeniería de software asistida por computadora
- b. Ingeniería de software asistida por componentes
- c. Ingeniería constructiva de software asistido
- d. Ingeniería de Software de Análisis Computacional

93. FAST significa \_\_\_\_\_.

- a. Técnica de software de aplicación facilitada.
- b. Técnica de software de aplicación funcional.
- c. Facilitado técnica de especificación de aplicaciones.
- d. Ninguna de las anteriores.

94. Abrevia el término CASO.

- a. Ingeniería de software autorizada por computadora
- b. Ingeniería de software asistida por computadora
- c. Ingeniería de Software Autorizada Común
- d. Ingeniería de software con asistencia común

95. ¿Qué se describe por medio de DFD como se estudió anteriormente y se representa en forma algebraica?

- a. Flujo de datos
- b. Almacenamiento de datos
- c. Estructuras de datos
- d. Elementos de datos

96. Una entidad en el Modelo ER es un ser del mundo real, que tiene algunas propiedades llamadas\_\_\_\_\_.

- a. Atributos
- b. Relación
- c. Dominio
- d. Ninguna de las anteriores

97. El número máximo de objetos que pueden participar en una relación se denomina\_\_\_\_\_.

- a. Cardinalidad
- b. Atributos
- c. Operaciones
- d. Transformadores

98. Si una aplicación permite ejecutar varias instancias de sí misma, aparecen en la pantalla, ya que las ventanas separadas se denominan \_\_\_\_\_.

- a. Ventana
- b. Pestañas
- c. Menú
- d. Cursor

99. Un marco de proceso genérico para la ingeniería de software abarca cinco actividades. ¿Qué son esas actividades?

- a. Comunicación, gestión de riesgos, medición, producción, despliegue.
- b. Comunicación, planificación, modelado, construcción, despliegue.
- c. Análisis, diseño, programación, depuración, mantenimiento.
- d. Ninguna de las anteriores

100. Boehm sugiere un enfoque que aborde los objetivos del proyecto, los hitos y los cronogramas, las responsabilidades, los enfoques de gestión y técnicos y los recursos necesarios. Este principio se llama \_\_\_\_\_.

- a. Principio W3HH
- b. Principio de la OMS
- c. Principio W5HH
- d. Ninguna de las anteriores



## RESPUESTAS DE LA PRIMERA PRÁCTICA DE EXAMEN

1. (d) .Todos los mencionados anteriormente.
2. (a) software del sistema
3. (d). Todas las anteriores
4. (d) .Todo lo anterior.
5. (a). Establezca los objetivos y el alcance del producto.
6. (c).
7. (d) .Control Estructura
8. (d) .1-b, 2-c, 3-a, 4-d
9. (d) .Ninguno de los anteriores.
10. (c). Ambas a y b
11. (d) .Todos los mencionados
12. (b). Los ingenieros de software se asegurarán de que sus productos y las modificaciones relacionadas cumplan con los estándares profesionales más altos posibles.
13. (a). Su empleador lanza un sistema crítico para la seguridad sin terminar la prueba del sistema.
14. (b) .act Consisten con el interés público.
15. (c). Ser dependientes de sus colegas
16. (b) licencia
17. (d) .ENVIRONMENT
18. (c). Ambos ayb
19. (c). Aumento de la oferta.
20. (d) .ayb
21. (a). Dependencia del softwad.
22. (c) .La informática pertenece a la ingeniería de softwad.
23. (d) .Todo lo anterior.
24. (b). Flexibilidad.
25. (d). Validación del softwad.
26. (b). Los sistemas basados en la Web han llevado a la degradación de los lenguajes de programación.
27. (c) .Unix shell
28. (c) .4, 1, 3, 2
29. (d). Ambos b & c
30. (b). Herramientas de softwad.
31. (a). Desarrollo rápido de aplicaciones.
32. (d).
33. (d) .Todos los mencionados anteriormente.
34. (c) .V-modelo
35. (a) .Big Bang modelo
36. (b). Desarrollo secuencial lineal
37. (d). Estabilidad organizacional.
38. (a) .100-200
39. (d).
40. (c). Prototipo diagonal.
41. (b) .Codificación
42. (d). Se amplía bien a proyectos grandes.
43. (c) .5 fases
44. (d). Ambos a y c.
45. (a). Ciclo de vida de desarrollo de software

Esta guía de estudio, fue vendida por [www.guiaaceneval.mx](http://www.guiaaceneval.mx) Todos los Derechos Reservados.

46. (c) .RAD Modelo
47. (d). Todos son modelos de software evolutivos.
48. (c) .Linear Model & Prototyping Model
49. (d). Ambos b & c
50. (b) .Barry Boehm
51. (c) .radial, angular
52. (b). Define un conjunto de actividades de negociación al comienzo de cada paso alrededor de la espiral.
53. (a). No funciona bien para proyectos más pequeños
54. (a). El progreso se puede medir para un modelo incremental.
55. (c). Modelo de Corriente
56. (d) .Todos los mencionados
57. (b) .Prototipado y espiral.
58. (a) .Espiral
59. (c) .RAD
60. (d) .RAD & Prototyping
61. (d) .Algoritmo para la implementación de softwad.
62. (d). Análisis del sistema
63. (c) .Nuevos proyectos de desarrollo de aplicaciones.
64. (d) .Todos los mencionados anteriormente.
65. (d). Prueba de caja negra
66. (a) Recopilación de requisitos.
67. (b).
68. (a). Especificación de requisitos de software
69. (a) Estudio de viabilidad.
70. (a). Documentación de requisitos.
71. (d) .Todo lo anterior.
72. (d) .Todo lo anterior.
73. (c). Herramientas de caja integradas.
74. (d) .Todos los mencionados
75. (d). Ambos ayb
76. (c). Técnica de especificación de aplicación facilitada
77. (c) despliegue de la función de calidad
78. (b) .SRS
79. (d) .Usuarios del softwad.
80. (d) .Ninguno de los mencionados
81. (b) .Testabilidad
82. (a) .Funcional
83. (d). Ambos ayb
84. (c). Enfoque cuantitativo: encuentre escalas medibles para los atributos de funcionalidad
85. (d). Cinco
86. (b). Eficiencia de la velocidad.
87. (c) .Disponibilidad, confiabilidad, seguridad, seguridad.
88. (a) Identificación de las partes interesadas
89. (c) .ii, iv, i, iii
90. (b) .iii, iv, ii, i
91. (a). Diseño arquitectónico.
92. (a). Ingeniería de software asistida por computadora
93. (c). Técnica de especificación de aplicación facilitada.
94. (b) .Ingeniería de software asistida por computadora



95. (a). Flujo de datos.
96. (a) .Atributos
97. (a) .Cardinalidad
98. (b) .Tabs
99. (b). Comunicación, planificación, modelado, construcción, despliegue
100. (c) Principio .W5HH

## SEGUNDA PRÁCTICA DE EXAMEN

1. Las actividades y las acciones realizadas sobre los datos están representadas por círculos o rectángulos de bordes redondos que se llaman \_\_\_\_\_.
  - a. Entidades
  - b. Proceso
  - c. Almacenamiento de datos
  - d. Flujo de datos
  
2. Los lenguajes OOD proporcionan un mecanismo donde los métodos que realizan tareas similares pero varían en los argumentos, y que se pueden asignar al mismo nombre se llama \_\_\_\_\_.
  - a. Las clases
  - b. Objeto
  - c. Polimorfismo
  - d. Encapsulacion
  
3. ¿Qué sistema basado en computadora puede tener un efecto profundo en el diseño que se elija y también se aplicará el enfoque de implementación?
  - a. Elementos basados en escenarios
  - b. Elementos basados en la clase
  - c. Elementos de comportamiento
  - d. Elementos orientados al flujo.
  
4. El nombre del diagrama que representa el flujo de actividades descrito por los casos de uso y, al mismo tiempo, los captores están involucrados en UML.
  - a. Diagrama de estado
  - b. Diagrama de carril de natación
  - c. Diagrama de actividad
  - d. Diagrama de componentes
  
5. De lo siguiente, seleccione la opción correcta que se usa para mostrar la opción disponible para la selección.
  - a. Caja
  - b. Caja de texto
  - c. Botón
  - d. Boton de radio



6. ¿Cuál de los niveles lleva a cabo el objetivo, el objetivo, las tareas de trabajo, los productos de trabajo y otras actividades del proceso de software?
- a. Realizado
  - b. Incompleto
  - c. Optimizado
  - d. Gestionado cuantitativamente
7. En OOD, los atributos (variables de datos) y los métodos (operación en los datos) que se agrupan se llaman \_\_\_\_\_.
- a. Las clases
  - b. Objetos
  - c. Encapsulacion
  - d. Herencia
8. ¿Qué es lo que a los diseñadores les gustaría tener una lista de todos los requisitos funcionales y no funcionales de la GUI que pueden tomarse del usuario y su solución de software existente?
- a. Análisis de usuario
  - b. Análisis de tareas
  - c. Recopilación de requisitos de GUI
  - d. Diseño e implementación de GUI
9. ¿Qué clases implementan las abstracciones de negocios de nivel inferior que se requieren para administrar completamente las clases de dominio de negocios?
- a. Clases de interfaz de usuario
  - b. Clases de dominio de negocios
  - c. Clases de proceso
  - d. Clases de sistema
10. ¿Qué tipo de DFD se concentra en el proceso del sistema y el flujo de datos en el sistema?
- a. DFD lógico
  - b. DFD fisico
  - c. Tanto A y B
  - d. Ninguna de las anteriores

11. Cada atributo se define por su correspondiente conjunto de valores que se llama \_\_\_\_\_.

- a. Entidad
- b. Dominio
- c. Relación
- d. Ninguna de las anteriores

12. El Lenguaje de modelado unificado (UML) se ha convertido en un estándar efectivo para el modelado de software. ¿Cuántas notaciones diferentes tiene?

- a. Tres
- b. Cuatro
- c. Seis
- d. Nueve

13. ¿Qué modelo en el modelado del sistema representa el comportamiento dinámico del sistema?

- a. Modelo de contexto
- b. Modelo de comportamiento
- c. Modelo de datos
- d. Modelo de objeto

14. ¿Qué modelo en el modelado de sistemas representa la naturaleza estática del sistema?

- a. Modelo de comportamiento
- b. Modelo de contexto
- c. Modelo de datos
- d. Modelo estructural

15. ¿Qué perspectiva en el modelado de sistemas muestra el sistema o la arquitectura de datos?

- a. Perspectiva estructural
- b. Perspectiva de comportamiento
- c. Perspectiva externa
- d. Ninguna de las anteriores



16. El UML admite el modelado basado en eventos utilizando diagramas \_\_\_\_\_.

- a. Despliegue
- b. Colaboración
- c. Estado
- d. Ninguna de las anteriores

17. ¿Cuál de los siguientes diagramas no es compatible con UML considerando el modelado basado en datos?

- a. Actividad
- b. Diagrama de flujo de datos (DFD)
- c. Carta del estado
- d. Componente

18. \_\_\_\_\_ nos permite inferir que diferentes miembros de clases tienen algunas características comunes.

- a. Realización
- b. Agregación
- c. Generalización
- d. dependencia

19. \_\_\_\_\_ y \_\_\_\_\_ diagramas de UML representan modelos de interacción.

- a. Caso de uso, secuencia
- b. Clase, objeto
- c. Actividad, gráfico del estado
- d. Ninguna de las anteriores

20. ¿Qué nivel de Diagrama de relaciones entre entidades (ERD) modela todas las entidades y relaciones?

- a. Nivel 1
- b. Nivel 2
- c. Nivel 3
- d. Nivel 4

21. La desventaja de la modularización es

- a. Los componentes más pequeños son más fáciles de mantener
- b. Programa puede dividirse en función de aspectos funcionales.
- c. El nivel deseado de abstracción se puede traer en el programa
- d. Ninguna de las anteriores

22. La aplicación que genera un diálogo para obtener la confirmación del usuario y para eliminar un archivo es un ejemplo para \_\_\_\_\_.

- a. Botón de radio
- b. Caja de texto
- c. Caja
- d. Cuadro de dialogo

23. ¿Cuál de las siguientes opciones no se puede aplicar con el software de acuerdo con las capas de ingeniería de software?

- a. Proceso
- b. Métodos
- c. Fabricación
- d. Ninguna de las anteriores

24. ¿Qué software se usa para controlar productos y sistemas para los mercados de consumo e industrial?

- a. Software del sistema
- b. Software de inteligencia artificial
- c. Software embebido
- d. Ingeniería y software científico.

25. ¿Qué componente de software permite que el programa manipule adecuadamente la información?

- a. Instrucciones
- b. Estructuras de datos
- c. Documentos
- d. Todas las anteriores



26. A partir de lo siguiente, ¿qué software se ha caracterizado por los algoritmos de 'Crujido de números'?

- a. Software del sistema
- b. Software de inteligencia artificial
- c. Software embebido
- d. Ingeniería y software científico.

27. ¿Qué es una ventana secundaria que contiene un mensaje para el usuario y solicita que se realice alguna acción?

- a. Cuadro de dialogo
- b. Caja de texto
- c. Caja
- d. Boton de radio

28. ¿Qué nivel de subsistema se usa de una aplicación?

- a. Nivel de aplicación
- b. Nivel de componente
- c. Nivel de modulos
- d. Ninguna de las anteriores

29. ¿Qué diseño define la estructura lógica de cada módulo y sus interfaces que se utiliza para comunicarse con otros módulos?

- a. Diseños de alto nivel
- b. Diseños arquitectonicos
- c. Diseño detallado
- d. Todo lo mencionado anteriormente

30. ¿Qué diseño se ocupa de la parte de implementación en la que muestra un sistema y sus subsistemas en los dos diseños anteriores?

- a. Diseño arquitectonico
- b. Diseño de alto nivel
- c. Diseño detallado
- d. Tanto A y B

31. El proceso y la mejora del software son evaluados por \_\_\_\_.
- a. ISO 9000
  - b. ISO 9001
  - c. ESPECIA (ISO / IEC15504)
  - d. Tanto b/c
32. ¿Cuál de los siguientes no es una sección en el estándar para los planes SQA recomendados por IEEE?
- a. Presupuesto
  - b. Hora
  - c. Gente
  - d. Ninguna de las anteriores
33. ¿Cuál es la exactitud, integridad y consistencia del modelo de requisitos tendrá una fuerte influencia en la calidad de todos los productos de trabajo que siguen?
- a. Calidad de los requisitos
  - b. Calidad de diseño
  - c. Calidad del código
  - d. Efectividad del control de calidad.
34. IEEE proporciona un estándar como IEEE 830-1993. ¿Para qué actividad se recomienda esta norma estándar?
- a. Especificación de requisitos de software
  - b. Diseño de software
  - c. Pruebas
  - d. Tanto a como B
35. El Six Sigma para ingeniería de software ¿Qué proporciona el proceso existente y su salida para determinar el rendimiento de calidad actual?
- a. Definir
  - b. Analizar
  - c. Medida
  - d. Ninguna de las anteriores



36. ¿Cuál de los siguientes no es un software de calidad?

- a. Productividad
- b. Portabilidad
- c. Oportunidad
- d. Visibilidad

37. Cuál de las siguientes no es una Actividad Paraguas que complemente las cinco actividades del marco de proceso y ayude al equipo a gestionar y controlar el progreso, la calidad, el cambio y el riesgo.

- a. Gestión de reutilización.
- b. Medición
- c. Gestión de riesgos
- d. Opiniones de los usuarios

38. Elija un software interno de calidad a continuación:

- a. escalabilidad
- b. usabilidad
- c. reutilización
- d. confiabilidad

39. El modelo CMM en Ingeniería de Software es una técnica de \_\_\_\_\_.

- a. Desarrollar el software
- b. Mejorar el proceso del softwad.
- c. Mejorar el proceso de prueba.
- d. Todas las anteriores

40. Si no tiene idea de cómo mejorar el proceso para el software de calidad, ¿qué modelo se utiliza?

- a. Un modelo continuo
- b. Un modelo por etapas
- c. Tanto A y B
- d. Ninguna de las anteriores

41. ¿Qué tipo de prueba de software se usa generalmente en Mantenimiento de Software?

- a. Pruebas de regresión
- b. Pruebas del sistema
- c. Pruebas de integración
- d. Examen de la unidad

42. La prueba de regresión es una actividad muy costosa.

- a. Ciento
- b. Falso
- c. No puedo decir
- d. Ninguno de los mencionados

43. Las técnicas de reevaluación selectiva pueden ser más económicas que la técnica de "reprobar todo". ¿Cuántas técnicas de reevaluación selectiva existen?

- a. dos
- b. Tres
- c. cuatro
- d. cinco

44. ¿Qué técnica de reevaluación selectiva selecciona cada caso de prueba que hace que un programa modificado produzca una salida diferente a su versión original?

- a. Cobertura
- b. Minimización
- c. Seguro
- d. Maximización

45. \_\_\_\_\_ mide la capacidad de una técnica de selección de prueba de regresión para manejar aplicaciones realistas.

- a. Eficiencia
- b. Precisión
- c. Generalidad
- d. Inclusión



46. ¿Qué técnica de selección de prueba de regresión expone fallas causadas por modificaciones?

- a. Eficiencia
- b. Precisión
- c. Generalidad
- d. Inclusión

47. El proceso de generación de documentos de análisis y diseño se conoce como

- a. Ingeniería de software
- b. Reingeniería de software
- c. Ingeniería inversa
- d. Reingeniería

48. ¿Qué es un parche de software?

- a. Arreglo requerido o crítico
- b. Arreglo de emergencia
- c. Corrección diaria o rutinaria
- d. Ninguno de los mencionados

49. ¿Cuál de los siguientes no es un modelo de mantenimiento?

- a. Modelo de cascada
- b. Modelo orientado a la reutilización
- c. Modelo de mejora iterativa
- d. Modelo de solución rápida

50. ¿Qué significa ACT en el modelo In Boehm para el mantenimiento del software?

- a. Pista de cambio real
- b. Pista de cambio anual
- c. Tráfico de cambio anual
- d. Tráfico de cambio real

51. La certificación de terceros para estándares de software se basa en

- a. UI 1998, Segunda Edición
- b. UT 1998, Segunda Edición
- c. UI 1992, Segunda Edición
- d. UI 1996, Segunda Edición

52. ¿Cuáles son los objetivos para obtener la acreditación de laboratorio?

- a. Aumentar la disponibilidad de servicios de pruebas a través de laboratorios de terceros.
- b. Aumentar la disponibilidad del mercado de pruebas para fomentar el desarrollo de la industria de pruebas de software.
- c. Reducir los costos al aumentar la oferta de servicios de pruebas.
- d. Todos los mencionados

53. El Programa Nacional de Acreditación de Laboratorios Voluntarios aprueba la acreditación en

- a. Estándares ambientales
- b. Computadoras y electrónica
- c. Pruebas de producto
- d. Todos los mencionados

54. CSTE significa

- a. Tecnología de software certificada
- b. Probador de software certificado
- c. Aprendiz de software certificado
- d. Ninguno de los mencionados

55. CSQA significa

- a. Analista de Calidad de Software Certificado
- b. Calidad de software certificada aprobada
- c. Calidad de software certificada acreditada
- d. Ninguno de los mencionados



56. "Robustez" responde cuál de las siguientes descripciones?

- a. Las herramientas CASE se utilizarán para apoyar las actividades del proceso.
- b. Los errores de proceso se evitan o atrapan antes de que den como resultado errores del producto
- c. El proceso definido es aceptable y utilizable por los ingenieros responsables de producir el software
- d. El proceso continúa a pesar de problemas inesperados.

57. La mejora de procesos es el conjunto de actividades, métodos y transformaciones que los desarrolladores utilizan para desarrollar y mantener sistemas de información.

- a. Cierto
- b. Falso
- c. No puedo decir
- d. Ninguno de los mencionados

58. ¿La "comprensibilidad" responde cuál de las siguientes descripciones?

- a. La medida en que el proceso se define explícitamente
- b. Los errores de proceso se evitan o atrapan antes de que den como resultado errores del producto
- c. El proceso definido es aceptable y utilizable por los ingenieros responsables de producir el producto de software
- d. El proceso continúa a pesar de problemas inesperados.

59. ¿Cuántas etapas hay en proceso de mejora?

- a. Tres
- b. cuatro
- c. cinco
- d. seis

60. ¿En qué etapa de mejora de procesos se identifican cuellos de botella y debilidades?

- a. Medición del proceso
- b. Análisis de proceso
- c. Cambio de proceso
- d. Ninguno de los mencionados

61. COCOMO significa \_\_\_\_\_.

- a. Consumo de café
- b. Consejo de Construcción
- c. Control de Control
- d. Composición

62. ¿Cuál puede estimarse en términos de KLOC (línea de código de Kilo) o al calcular el número de puntos de función en el software?

- a. Estimación del tiempo
- b. Estimación del esfuerzo
- c. Estimación de costos
- d. Estimación del tamaño del software

63. ¿Qué métricas se derivan al normalizar las medidas de calidad y / o productividad considerando el tamaño del software que se ha producido?

- a. Tamaño orientado
- b. Orientado a funciones
- c. Orientado a objetos
- d. Uso-caso-orientado

64. ¿Cuál es la medida más común para la corrección?

- a. Defectos por KLOC
- b. Errores por KLOC
- c. \$ por KLOC
- d. Páginas de documentación por KLOC.

65. Abrevia el término SMI.

- a. Índice de madurez del software
- b. Instrucción del modelo de software
- c. Instrucción de madurez de software
- d. Índice de modelo de software



66. En la técnica de estimación empírica, ¿qué modelo ha desarrollado Barry W. Boehm?

- a. Modelo putnam
- b. COCOMO
- c. Ambos a y b
- d. Ninguna de las anteriores

67. ¿La línea de código (LOC) del producto viene bajo qué tipo de medidas?

- a. Medidas indirectas
- b. Medidas directas
- c. Codificación
- d. Ninguna de las anteriores

68. ¿Cuál es el tamaño del software de estimación debe ser conocido?

- a. Estimación del tiempo
- b. Estimación del esfuerzo
- c. Estimación de costos
- d. Estimación del tamaño del software

69. ¿Cuántos números de niveles de madurez en CMM están disponibles?

- a. 3
- b. 4
- c. 5
- d. 6

70. Cuál de los siguientes es la tarea de los indicadores del proyecto:

- a. Ayuda en la evaluación del estado del proyecto en curso.
- b. rastrear el riesgo potencial
- c. tanto a como B
- d. ninguno de los mencionados

71. La cantidad de tiempo que el software está disponible para su uso se conoce como

- a. Confiabilidad
- b. Usabilidad
- c. Eficiencia
- d. Funcionalidad

72. El tamaño y la complejidad son parte de

- a. Métricas del producto
- b. Métricas del proceso
- c. Métricas del proyecto
- d. Ninguna de las anteriores

73. El costo y el horario son parte de

- a. Métricas del producto
- b. Métricas del proceso
- c. Métricas del proyecto
- d. Ninguna de las anteriores

74. El número de errores encontrados por persona por hora gastada es un ejemplo de

- a. medición
- b. medida
- c. métrico
- d. Ninguna de las anteriores

75. ¿Cuál de los siguientes no está clasificado en la Operación del producto de los factores de calidad del software de McCall?

- a. Flexibilidad
- b. Confiabilidad
- c. Usabilidad
- d. Integridad



76. La relación de arco a nodo se da como  $r = a / n$ . ¿Qué representa 'a' en la relación?

- a. Número máximo de nodos en cualquier nivel
- b. El camino más largo desde la raíz hasta una hoja.
- c. número de módulos
- d. líneas de control

77. ¿Cuál de las siguientes no está categorizada bajo Métricas de diseño a nivel de componente?

- a. Métricas de Complejidad
- b. Métricas de Cohesión
- c. Métricas de morfología
- d. Métricas de acoplamiento

78. El porcentaje de módulos inspeccionados es parte de

- a. Métricas del producto
- b. Métricas del proceso
- c. Métricas del proyecto
- d. Ninguna de las anteriores

79. Identifique la opción correcta con referencia a Métricas de calidad del softwad.

- a. Integridad = [Sigma (1 - amenaza)] \* (1 - seguridad)
- b. Integridad = [1 - Sigma (amenaza)] \* (1 - seguridad)
- c. Integridad = [1 - amenaza \* Sigma (1 - seguridad)]
- d. Integridad = Sigma [1 - amenaza \* (1 - seguridad)]

80. De los siguientes métodos, ¿qué tamaño del producto de software se puede calcular?

- a. Contando las líneas de código entregadas.
- b. Contando los puntos de función entregados
- c. Tanto a como b anterior
- d. Ninguna de las anteriores

81. El modelo de proceso está representado por

- a. CMMI
- b. SEI
- c. CMI
- d. SE

82. Área de proceso en la que están presentes la innovación organizativa y el análisis y la resolución informales de despliegue.

- a. mejora continua del proceso
- b. gestión cuantitativa
- c. proceso de estandarización
- d. gestión básica de proyectos

83. Nivel de capacidad en el que se han alcanzado todos los criterios del nivel 3 y el área de proceso se controla y mejora mediante la medición y la evaluación de la calidad en

- a. Nivel 1: Realizado
- b. Nivel 2: Gestionado
- c. Nivel 3: Definido
- d. Nivel 4: Gestionado Cuantitativamente

84. El área de proceso que define y se enfoca en el proceso organizativo y la capacitación se enfoca en

- a. mejora continua del proceso
- b. proceso de estandarización
- c. gestión básica de proyectos
- d. gestión cuantitativa

85. El modelo CCMI que describe un proceso en dos dimensiones es

- a. modelo continuo
- b. modelo discreto
- c. modelo en escena
- d. modelo estable



86. El área de proceso que define y se centra en el proceso organizativo y la capacitación está presente en el nivel de

- a. optimizando
- b. definido
- c. realizado
- d. manejado

87. CMMI define la planificación del proyecto para la categoría

- a. gestión de proyectos
- b. tecnologías de la información
- c. gestión de acuerdos con proveedores
- d. gestión de riesgos

88. El modelo de proceso CMMI puede ser representado por

- a. solo una manera
- b. dos caminos
- c. tres maneras
- d. cuatro formas

89. CMMI significa

- a. Capacidad de madurez Integración de fusión
- b. Integración del modelo de madurez consumida
- c. Capacidad de integración del modelo de madurez
- d. Incremento del modelo de madurez de la capacidad

90. En el ciclo de planear-hacer-verificar-actuar, el establecimiento de objetivos de proceso, actividades y tareas necesarias para lograr un software de alta calidad se logra de

- a. plan
- b. hacer
- c. comprobar
- d. acto

91. Las métricas de cohesión y las métricas de acoplamiento son métricas en qué nivel de diseño?

- a. Diseño de interfaz de usuario
- b. Diseño basado en patrones
- c. Diseño arquitectónico
- d. Diseño a nivel de componente

92. Cuando los elementos del módulo se agrupan y se ejecutan secuencialmente para realizar una tarea, se llama \_\_\_\_\_.

- a. Cohesión procesal
- b. Cohesión lógica
- c. Cohesión temporal
- d. Cohesión incidental

93. ¿Qué acoplamiento también se conoce como "acoplamiento global"?

- a. Acoplamiento de contenido
- b. Acoplamiento de sello
- c. Acoplamiento de datos
- d. Acoplamiento común

94. ¿Cuál es la secuencia detallada de pasos que describe la interacción entre el usuario y la aplicación?

- a. Guiones de escenarios
- b. Clases de apoyo
- c. Clases claves
- d. Subsistemas

95. ¿Qué clases representan almacenes de datos (por ejemplo, una base de datos) que persistirán más allá de la ejecución del software?

- a. Clases de proceso
- b. Clases de sistema
- c. Clases persistentes
- d. Clases de interfaz de usuario



96. Abrevia el término CMMI.

- a. Capacidad de integración del modelo de madurez
- b. Integración de madurez modelo de capacidad
- c. Instrucciones del modelo de madurez de la capacidad
- d. Modelo de capacidad Instrucciones de madurez

97. La agrupación de todos los elementos relacionados funcionalmente se conoce como \_\_\_\_\_.

- a. Cohesión
- b. Acoplamiento
- c. Tanto A y B
- d. Ninguna de las anteriores

98. La cohesión es una indicación cualitativa del grado en que un módulo

- a. Se puede escribir de forma más compacta.
- b. se centra en una sola cosa.
- c. es capaz de completar su función de manera oportuna.
- d. Está conectado a otros módulos y al mundo exterior.

99. El acoplamiento es una indicación cualitativa del grado en que un módulo

- a. Se puede escribir de forma más compacta.
- b. se centra en una sola cosa.
- c. es capaz de completar su función de manera oportuna.
- d. Está conectado a otros módulos y al mundo exterior.

100. Los paquetes de Java y las subrutinas de FORTRAN son ejemplos de \_\_\_\_\_

- a. Funciones
- b. Módulos
- c. Las clases
- d. Sub procedimientos

# RESPUESTAS DE LA SEGUNDA PRÁCTICA DE EXAMEN

1. (b) .Proceso
2. (c) .Polimorfismo
3. (c). Elementos de comportamiento.
4. (b) Diagrama de carril Swim
5. (d) .Radio-Button
6. (a). Realizada
7. (c) .Encapsulación.
8. (c) .GUI requisito de recopilación
9. (c). Clases de proceso.
10. (a) DFD lógica.
11. (b) .Dominio
12. (d) .Nine
13. (b). Modelo de comportamiento.
14. (d). Modelo estructural.
15. (a). Perspectiva estructural.
16. (c) .Estado
17. (b) Diagrama de flujo de datos (DFD)
18. (c) .Generalización.
19. (a) .Use Caso, Secuencia
20. (b) .Nivel 2
21. (d) .Ninguno de los anteriores
22. (d). Cuadro de diálogo
23. (c) .Manufactura
24. (c). Software embebido.
25. (b). Estructuras de datos.
26. (d) .Ingeniería y software científico.
27. (a). Cuadro de diálogo
28. (b). Nivel de componente
29. (c) .Diseño detallado.
30. (c) .Diseño detallado.
31. (d). Ambos b y c
32. (a). Presupuesto
33. (a). Requerimiento de calidad.
34. (a). Especificación de requisitos de software
35. (c) .Medida
36. (b). Portabilidad
37. (d) .Usuario Comentarios
38. (c) .reusabilidad
39. (b). Mejorar el proceso de software
40. (b) .Un modelo por etapas.
41. (a). Pruebas de regresión
42. (a).
43. (b) .tres
44. (c).
45. (c). Generalidad.
46. (d) .Inclusividad.
47. (c) .Ingeniería inversa.



**SEP**

SECRETARÍA DE  
EDUCACIÓN PÚBLICA



CENTRO NACIONAL  
DE EVALUACIÓN PARA  
LA EDUCACIÓN SUPERIOR, A.C.



**EDITORIAL:**  
**guiaceneval.mx**



48. (b). Arreglo de emergencia
49. (a).
50. (c). Cambio anual de tráfico.
51. (a) .UI 1998, Segunda Edición
52. (d) .Todos los mencionados
53. (d) .Todos los mencionados
54. (b). Probador de software certificado
55. (a). Analista de calidad de software certificado
56. (d). El proceso continúa a pesar de problemas inesperados
57. (b). Falso
58. (a). La medida en que el proceso se define explícitamente
59. (a) .tres
60. (b) .Análisis del proceso.
61. (b).
62. (d). Estimación del tamaño del softwad.
63. (a). Orientado al tamaño.
64. (a) .Dfectos por KLOC
65. (a). Índice de madurez del software
66. (b) .COCOMO
67. (b) .Medidas directas.
68. (b). Estimación del esfuerzo.
69. (c) .5
70. (c). Tanto a y b
71. (a) .Fiabilidad
72. (a). Métricas del producto
73. (c). Métricas del proyecto
74. (c) .metric
75. (a). Flexibilidad.
76. (d) líneas de control.
77. (c) .Métricas de la morfología.
78. (b). Métricas del proceso
79. (d) .Integridad = Sigma [1 - amenaza * (1 - seguridad)]
80. (c). Tanto a como b arriba
81. (a). CMMI
82. (a). Mejora continua del proceso.
83. (d) .Level4: Gestionado cuantitativamente
84. (b) .Procesamiento de procesos.
85. (a) modelo continuo.
86. (b). Definida
87. (a) Gestión de proyectos.
88. (b). Dos maneras.
89. (c) .Capability Maturity Model Integration
90. (a) .plan
91. (d) Diseño a nivel de componente.
92. (a). Cohesión procesal.
93. (d). Acoplamiento común
94. (a) Scripts de escenario
95. (c). Clases persistentes.
96. (a) .Capability Maturity Model Integration
97. (a). Cohesión

98. (b) .se enfoca en una sola cosa.

99. (d) .está conectado a otros módulos y al mundo exterior.

100. (b) .Módulos



## TERCERA PRÁCTICA DE EXAMEN

1. Cuando los elementos del módulo se agrupan porque la salida de un elemento sirve como entrada a otro y así sucesivamente, se llama \_\_\_\_\_.
  - a. Cohesión funcional
  - b. Cohesión secuencial
  - c. Cohesión comunicacional
  - d. Cohesión procesal
  
2. A partir de lo siguiente, ¿qué método se adoptará en el proceso de reutilización?
  - a. Ya sea manteniendo los mismos requisitos y ajustando los componentes.
  - b. Manteniendo los mismos componentes y modificando los requisitos.
  - c. Tanto a como B
  - d. Ninguna de las anteriores
  
3. ¿Qué herramientas se utilizan para representar los componentes del sistema, los datos y el flujo de control entre varios componentes de software y una estructura del sistema en forma gráfica?
  - a. Herramientas de modelado de procesos
  - b. Herramientas de gestión de proyectos
  - c. Herramientas de diagrama
  - d. Herramientas de documentación
  
4. El diccionario de datos contiene descripciones de cada software.
  - a. elemento de control
  - b. objeto de datos
  - c. diagrama
  - d. tanto a como B
  
5. ¿Cuál de estos no es un elemento de un modelo de análisis orientado a objetos?
  - a. Elementos de comportamiento
  - b. Elementos basados en la clase
  - c. Elementos de datos
  - d. Elementos basados en escenarios

6. El polimorfismo reduce el esfuerzo requerido para extender un sistema de objetos en

- a. Acoplamiento de objetos más apretados.
- b. habilitando varias operaciones diferentes para compartir el mismo nombre.
- c. Haciendo los objetos más dependientes unos de otros.
- d. Eliminando las barreras impuestas por la encapsulación.

7. Cuando el flujo general en un segmento de un diagrama de flujo de datos es en gran parte secuencial y sigue caminos de línea recta, \_\_\_\_\_ está presente.

- a. acoplamiento bajo
- b. buena modularidad
- c. flujo de transacciones
- d. flujo de transformación

8. Cuando un solo elemento que activa otro flujo de datos a lo largo de una de las muchas rutas de un diagrama de flujo de datos, \_\_\_\_\_ caracteriza el flujo de información.

- a. acoplamiento alto
- b. pobre modularidad
- c. flujo de transacciones
- d. flujo de transformación

9. En el mapeo de transacciones, la factorización de primer nivel da como resultado el

- a. creación de un CFD
- b. Derivación de la jerarquía de control.
- c. Distribución de módulos de trabajo.
- d. refinamiento de la vista del módulo

10. Una aplicación exitosa de la transformación o la asignación de transacciones para crear un diseño arquitectónico se complementa con

- a. diagramas de relaciones de entidades
- b. Descripciones de la interfaz del módulo
- c. Procesamiento de narrativas para cada módulo.
- d. tanto y c



11. ¿Qué modelo de amplificación de defectos se utiliza para ilustrar la generación y detección de errores durante los pasos preliminares de un proceso de ingeniería de software?

- a. Diseño
- b. Diseño detallado
- c. Codificación
- d. Todo lo mencionado anteriormente

12. ¿Qué método se usa para evaluar la expresión que pasa la función como un argumento?

- a. Evaluación estricta
- b. Recursion
- c. Cálculo
- d. Funciones puras

13. La organización puede tener inspección interna, la participación directa de los usuarios y el lanzamiento de la versión beta son algunos de ellos y también incluye facilidad de uso, compatibilidad, aceptación del usuario, etc.

- a. Análisis de tareas
- b. Recopilación de requisitos de GUI
- c. Diseño e implementación de GUI
- d. Pruebas

14. ¿Qué prueba es la re-ejecución de algún subconjunto de pruebas que ya se han realizado para garantizar los cambios que no se propagan?

- a. Examen de la unidad
- b. Pruebas de regresión
- c. Pruebas de integración
- d. Pruebas basadas en hilos

15. ¿Cuál de estos pertenece a las pruebas de integración en el contexto OO?

- a. Examen de la unidad
- b. Pruebas de regresión
- c. Prueba de sándwich
- d. Pruebas basadas en hilos

16. Durante las pruebas de seguridad, el probador desempeña el papel de la persona que desea\_\_\_\_\_.

- a. Penetra el sistema
- b. Penetra el oyente
- c. Tanto a como B
- d. Ninguna de las anteriores

17. ¿Qué caja especifica el comportamiento de un sistema o una parte de un sistema?

- a. Caja de estado
- b. Caja clara
- c. Caja negra
- d. Ninguna de las anteriores

18. Cuanto más tiempo existe una falla en el software

- a. cuanto más tediosa se vuelve su eliminación
- b. Cuanto más costoso es detectar y corregir.
- c. Es menos probable que se corrija adecuadamente.
- d. Todos los mencionados

19. ¿Qué condición define las circunstancias para que una operación en particular sea válida?

- a. Postcondición
- b. Condición previa
- c. Invariante
- d. Ninguna de las anteriores

20. ¿Cuál no es una actividad de SQA?

- a. Prueba de caja negra
- b. Prueba de caja blanca
- c. Pruebas de integración
- d. Examen de la unidad



21. La prueba unitaria se realiza por

- a. Usuarios
- b. Desarrolladores
- c. Clientes
- d. Ninguno de los mencionados

22. Las pruebas de comportamiento son

- a. Prueba de caja blanca
- b. Prueba de caja negra
- c. Prueba de caja gris
- d. Ninguno de los mencionados

23. ¿Cuál de las siguientes es una prueba de caja negra?

- a. Prueba de ruta básica
- b. Análisis del valor límite
- c. Análisis de ruta de código
- d. Ninguno de los mencionados

24. ¿Cuál de los siguientes no se usa para medir el tamaño del software?

- a. KLOC
- b. Puntos de función
- c. Tamaño del módulo
- d. Ninguno de los mencionados

25. La depuración de software es un conjunto de actividades que se pueden planificar por adelantado y realizar de manera sistemática.

- a. Cierto
- b. Falso
- c. No puedo decir
- d. Ninguna de las anteriores

26. ¿Cuál de los siguientes no es un software que prueba características genéricas?

- a. Diferentes técnicas de prueba son apropiadas en diferentes puntos en el tiempo.
- b. La prueba es realizada por el desarrollador del software o un grupo de prueba independiente
- c. Las pruebas y la depuración son actividades diferentes, pero la depuración debe incluirse en cualquier estrategia de prueba.
- d. Ninguno de los mencionados

27. ITG significa

- a. grupo de prueba instantánea
- b. grupo de pruebas de integración
- c. grupo de prueba individual
- d. grupo de prueba independiente

28. Al recopilar \_\_\_\_\_ durante las pruebas de software, es posible desarrollar pautas significativas para detener el proceso de prueba.

- a. Intensidad de falla
- b. Tiempo de prueba
- c. Métrica
- d. Todos los mencionados

29. ¿Cuál de los siguientes problemas debe abordarse para implementar una estrategia de prueba de software exitosa?

- a. Utilice revisiones técnicas formales efectivas como filtro antes de la prueba
- b. Desarrolle un plan de pruebas que haga hincapié en las "pruebas de ciclo rápido".
- c. Objetivos de las pruebas estatales explícitamente
- d. Todos los mencionados

30. Los casos de prueba deberían descubrir errores como

- a. Terminación de bucle inexistente
- b. Comparación de diferentes tipos de datos
- c. Operadores lógicos incorrectos o precedencia
- d. Todos los mencionados



31. La codificación y las pruebas se realizan de la siguiente manera:

- a. Ad hoc
- b. Payaso
- c. De abajo hacia arriba
- d. Corte transversal

32. La principal diferencia entre la prueba del programa y la prueba del sistema es

- a. La prueba del sistema es difícil y la prueba del programa es fácil
- b. Las pruebas del programa son más completas que las pruebas del sistema
- c. Las pruebas del sistema se enfocan en probar las interfaces entre programas, las pruebas de programas se enfocan en programas individuales
- d. Ninguna de las anteriores

33. El diccionario de datos de wrt de comprobación cruzada se refiere a

- a. El hallazgo de que no hay incoherencia se cita en un archivo de texto y, por lo tanto, la referencia aparece en el archivo de referencia
- b. Enlace de documentos a través de hipertexto, en caso de que el sistema se ocupe de la imagen del documento.
- c. Determinación de dónde se utilizan los datos en el sistema.
- d. Ninguna de las anteriores

34. En ejecución paralela de pruebas de s / w

- a. Ejecución solo informatizada
- b. Sólo se ejecuta el sistema físico
- c. Tanto el sistema físico como el computarizado funcionan simultáneamente.
- d. Ninguna de las anteriores

35. El costo de la corrección de errores es de al menos

- a. Etapa de diseño
- b. Etapa de desarrollo
- c. Etapa de implementación
- d. Etapa de análisis de requerimientos

36. Variación en la depuración: se ha informado que la capacidad de codificación es

- a. 1: 1
- b. 1: 3
- c. 1: 7
- d. 1:10

37. En casos extremos, la variación de la codificación: se ha informado que la capacidad de depuración de los programadores es

- a. 1: 2
- b. 1: 6
- c. 1:15
- d. 1:29

38. Una sola entrada, una única salida se contrae con if, while, enunciados de secuencia y compuesto en C y simula

- a. un hipercomputador
- b. una máquina de tornear
- c. solo autómatas finitos
- d. sólo un autómata de empuje hacia abajo

39. Forma la especificación detallada del diseño. Si la codificación se realiza en C ++: C: ensamblador, los tamaños de código resultantes estarán en el promedio

- a. 1: 3: 4
- b. 10: 3: 1
- c. 1: 3: 10
- d. 1: 10: 100

40. Probando un programa a fondo

- a. Sólo se encontrarán algunos errores.
- b. Garantiza que todos los defectos serán encontrados.
- c. Garantiza que todos los errores serán encontrados.
- d. Ninguna de las anteriores



41. La gestión de proyectos de software consta de una serie de actividades, que contienen \_\_\_\_\_.

- a. Planificación de proyectos
- b. Gestión del alcance
- c. Estimación del proyecto
- d. Todo lo mencionado anteriormente

42. La gestión eficaz de proyectos de software se centra en las cuatro P's. ¿Cuáles son esas cuatro p?

- a. Personas, rendimiento, pago, producto.
- b. Personas, producto, proceso, proyecto.
- c. Personas, producto, performance, proyecto.
- d. Todas las anteriores

43. Dar los factores del mundo real que afectan el costo de mantenimiento.

- a. A medida que la tecnología avanza, resulta costoso mantener el software antiguo.
- b. La edad estándar de cualquier software se considera hasta 10 a 15 años.
- c. La mayoría de los ingenieros de mantenimiento son novatos y utilizan el método de prueba y error para corregir el problema.
- d. Todo lo mencionado anteriormente

44. ¿Qué factores finales del software afectan el costo de mantenimiento?

- a. Estructura del programa de software
- b. Lenguaje de programación
- c. Dependencia del entorno externo.
- d. Todo lo mencionado anteriormente

45. La naturaleza siempre creciente y adaptable del software depende en gran medida del entorno en el que el usuario trabaja en \_\_\_\_\_.

- a. Costo
- b. Naturaleza dinámica
- c. Gestión de la calidad
- d. Escalabilidad

46. Cuando el cliente puede solicitar nuevas características o funciones en el software, ¿qué significa en el mantenimiento del software?

- a. Modificaciones de host
- b. Requisitos del cliente
- c. Condiciones de mercado
- d. Cambios organizativos

47. ¿Qué se utiliza para implementar los cambios en los requisitos nuevos o existentes del usuario en el mantenimiento del software?

- a. Mantenimiento preventivo
- b. Mantenimiento perfecto
- c. Mantenimiento correctivo
- d. Mantenimiento adaptativo

48. Lehman dio ocho leyes para la evolución del software y dividió el software en tres categorías. En qué categoría funciona el software estrictamente de acuerdo con las especificaciones y soluciones definidas.

- a. Tipo estático
- b. Tipo incrustado
- c. Tipo práctico
- d. Ninguna de las anteriores

49. ¿Qué herramientas se utilizan en la implementación, prueba y mantenimiento?

- a. Herramientas mayúsculas
- b. Herramientas de caja integradas
- c. Herramientas de caja baja
- d. Ninguna de las anteriores

50. La gestión de proyectos de software es el proceso de gestión de todas las actividades que están involucradas en el desarrollo de software, son \_\_\_\_\_.

- a. Hora
- b. Costo
- c. Gestión de la calidad
- d. Todo lo mencionado anteriormente



51. Las técnicas que permiten que un ingeniero de software entienda cómo se completa un proceso de trabajo cuando se incluye a varias personas, se llama \_\_\_\_\_.

- a. Análisis de flujo de trabajo
- b. No rastrea riesgos potenciales
- c. Cubra las áreas problemáticas antes de que se vuelvan críticas.
- d. No ajusta el flujo de trabajo ni las tareas.

52 ¿Cuál de los siguientes proporciona soporte semiautomático y automático a los métodos en una tecnología de capas?

- a. Métodos
- b. Herramientas
- c. Proceso
- d. Enfoque de calidad

53. Si el proceso del software no estuviera basado en conceptos científicos y de ingeniería, sería más fácil recrear un nuevo software que escalar uno existente, se conoce como\_\_\_\_\_.

- a. Costo
- b. Gestión dinámica
- c. Software grande
- d. Escalabilidad

54. ¿Cuáles de las siguientes son razones válidas para recopilar los comentarios de los clientes sobre el software entregado?

- a. Permite a los desarrolladores realizar cambios en el incremento entregado
- b. El horario de entrega puede ser revisado para reflejar los cambios
- c. Los desarrolladores pueden identificar cambios para incorporar en el siguiente incremento
- d. Todas las anteriores

55. ¿Cuál de los siguientes puede ser elementos de sistemas basados en computadora?

- a. documentación
- b. software
- c. gente
- d. Todas las anteriores

56. ¿Cuál de los siguientes no es el objetivo de la gestión del proyecto?

- a. Mantener los costos generales dentro del presupuesto
- b. Entrega del software al cliente a la hora acordada.
- c. Mantener un equipo de desarrollo feliz y que funcione bien.
- d. Evitar las quejas de los clientes.

57. Los gerentes de proyectos tienen que evaluar los riesgos que pueden afectar un proyecto.

- a. Cierto
- b. Falso
- c. No puedo decir
- d. Ninguna de las anteriores

58. ¿Cuál de los siguientes no se considera un riesgo en la gestión de proyectos?

- a. Retrasos en las especificaciones
- b. Concurso de productos
- c. Pruebas
- d. Rotación de personal

59. El proceso que sigue cada gerente durante la vida de un proyecto se conoce como

- a. Gestión de proyectos
- b. Ciclo de vida del gerente
- c. Gestión del proyecto Ciclo de vida
- d. Todos los mencionados

60. Se considera como riesgo un 66.6%.

- a. muy bajo
- b. bajo
- c. moderar
- d. alto



61. ¿Qué técnica es aplicable cuando se han completado otros proyectos en el mismo dominio de aplicación de analogía?

- a. Modelado algorítmico de costos
- b. Opinión de expertos
- c. Estimación por analogía
- d. Ley de Parkinson

62. ¿Qué modelo asume que los sistemas se crean a partir de componentes reutilizables, secuencias de comandos o programación de bases de datos?

- a. Un modelo de aplicación-composición.
- b. Un modelo post-arquitectura.
- c. Un modelo de reutilización.
- d. Un modelo de diseño temprano.

63. ¿Cuál de los siguientes estados en que el trabajo se expande para llenar el tiempo disponible?

- a. Herramientas de caja
- b. Precios para ganar
- c. Ley de Parkinson
- d. Opinión de expertos

64. ¿Qué modelo se usa en las primeras etapas del diseño del sistema después de que se hayan establecido los requisitos?

- a. Un modelo de aplicación-composición.
- b. Un modelo post-arquitectura.
- c. Un modelo de reutilización.
- d. Un modelo de diseño temprano.

65. ¿Qué modelo se usa para calcular el esfuerzo requerido para integrar componentes reutilizables o código de programa que se genera automáticamente mediante el diseño o las herramientas de traducción de programas?

- a. Un modelo de aplicación-composición.
- b. Un modelo post-arquitectura.
- c. Un modelo de reutilización.
- d. Un modelo de diseño temprano.

66. El modelo COCOMO tiene en cuenta diferentes enfoques para el desarrollo de software, la reutilización, etc.

- a. Cierto
- b. Falso
- c. No puedo decir
- d. Ninguna de las anteriores

67. ¿Cuál de los siguientes utiliza fórmulas derivadas empíricamente para predecir el esfuerzo en función de LOC o FP?

- a. Estimación basada en FP
- b. Estimación basada en el proceso
- c. COCOMO
- d. Tanto la estimación basada en FP como COCOMO

68. Los datos empíricos que respaldan la mayoría de los modelos de estimación se derivan de una amplia muestra de proyectos.

- a. Cierto
- b. Falso
- c. No puedo decir
- d. Ninguna de las anteriores

69. ¿Qué versión de COCOMO establece que una vez que los requisitos se han estabilizado, se ha establecido la arquitectura básica del software?

- a. Modelo de etapa de diseño temprano
- b. Modelo post-arquitectura-escenario
- c. Modelo de composición de la aplicación
- d. Todos los mencionados

70. Qué modelo se usó durante las primeras etapas de la ingeniería de software, cuando la creación de un prototipo de las interfaces de usuario, la consideración del software y la interacción del sistema, la evaluación del rendimiento y la evaluación de la madurez de la tecnología fueron primordiales.

- a. Modelo de etapa de diseño temprano
- b. Modelo post-arquitectura-escenario
- c. Modelo de composición de la aplicación
- d. Todos los mencionados



71. ¿Qué factores afectan las consecuencias probables si se produce un riesgo?

- a. Evitación de riesgo
- b. Monitoreo de riesgos
- c. Tiempo de riesgo
- d. Planificación de contingencias

72. La rotación de personal, la mala comunicación con el cliente son riesgos que se extrapolan de la experiencia pasada y se llaman \_\_\_\_.

- a. Riesgos de negocio
- b. Riesgos predecibles
- c. Riesgos del proyecto
- d. Riesgos técnicos

73. ¿Qué riesgo da el grado de incertidumbre y se mantendrá el cronograma del proyecto para que el producto se entregue a tiempo?

- a. Riesgo del negocio
- b. Riesgo técnico
- c. Riesgo de horario
- d. Riesgo del proyecto

74. ¿En qué modelo se considera el factor de riesgo del proyecto?

- a. Modelo espiral
- b. Modelo de cascada
- c. Modelo de prototipado
- d. Ninguna de las anteriores

75. De lo siguiente se dan tres categorías principales de riesgo,

- 1) Horario de riesgo
- 2) Riesgo de proyecto
- 3) riesgo técnico
- 4) Riesgo de negocio

- a. 1,2 y 3
- b. 2,3 y 4
- c. 1,2 y 4
- d. 1,3 y 4

76. En el proceso de gestión de riesgos, ¿qué hace una nota de todos los riesgos posibles que pueden ocurrir en el proyecto?

- a. Gestionar
- b. Monitor
- c. Clasificar por categorías
- d. Identificación

77. ¿Qué riesgos identifican los posibles problemas de diseño, implementación, interfaz, verificación y mantenimiento?

- a. Riesgo del proyecto
- b. Riesgo del negocio
- c. Riesgo tecnico
- d. Riesgo de horario

78. Construir un producto o sistema excelente para que nadie realmente quiera un riesgo es una \_\_\_\_\_.

- a. Riesgo tecnico
- b. Riesgo de horario
- c. Riesgo del negocio
- d. Riesgo de rendimiento

79. ¿Cuáles son las características del riesgo del software?

- a. Incertidumbre
- b. Pérdida
- c. Tanto a como B
- d. Ninguna de las anteriores

80. ¿Qué todo tiene que ser identificado como por identificación de riesgo?

- a. Amenazas
- b. Vulnerabilidades
- c. Consecuencias
- d. Todos los mencionados



81. ¿Cuál no es una actividad de gestión de riesgos?

- a. Evaluación de riesgos
- b. Generacion de riesgo
- c. Control de riesgo
- d. Ninguno de los mencionados

82. ¿Cuál es el producto de la probabilidad de incurrir en una pérdida debido al riesgo y la magnitud potencial de esa pérdida?

- a. Riesgo de exposición
- b. Priorización de riesgos
- c. Análisis de riesgo
- d. Todos los mencionados

83. ¿Qué amenaza la calidad y la puntualidad del software que se va a producir?

- a. Riesgos conocidos
- b. Riesgos de negocio
- c. Riesgos del proyecto
- d. Riesgos tecnicos

84. ¿Qué amenaza la viabilidad del software a construir?

- a. Riesgos conocidos
- b. Riesgos de negocio
- c. Riesgos del proyecto
- d. Riesgos tecnicos

85. ¿Cuál de los siguientes no es un riesgo comercial?

- a. Construyendo un excelente producto o sistema que nadie realmente quied.
- b. perder el apoyo de la alta gerencia debido a un cambio de enfoque o cambio en las personas
- c. Falta de requisitos documentados o alcance del softwad.
- d. perder presupuesto o compromiso personal

86. ¿Cuál de los siguientes es un intento sistemático de especificar amenazas al plan del proyecto?

- a. Identificación de riesgo
- b. Riesgo de rendimiento
- c. Riesgo de apoyo
- d. Proyección de riesgos

87. ¿Qué riesgos están asociados con el tamaño total del software que se creará o modificará?

- a. Riesgos de impacto empresarial
- b. Definición de procesos riesgos
- c. Riesgos del tamaño del producto
- d. Riesgos del entorno de desarrollo.

88. ¿Qué riesgos están asociados con las restricciones impuestas por la administración o el mercado?

- a. Riesgos de impacto empresarial
- b. Definición de procesos riesgos
- c. Riesgos del tamaño del producto
- d. Riesgos del entorno de desarrollo.

89. ¿Cuál de los siguientes términos se define mejor en la declaración: "el grado de incertidumbre de que el producto cumplirá con sus requisitos y será adecuado para su uso previsto"?

- a. Riesgo de rendimiento
- b. Riesgo de costo
- c. Riesgo de apoyo
- d. Riesgo de horario

90. La gestión de riesgos es uno de los trabajos más importantes para un

- a. Cliente
- b. Inversor
- c. Equipo de producción
- d. Gerente de proyecto



91. ¿Cuál de los siguientes riesgos es la falla de un componente comprado para funcionar como se espera?

- a. Riesgo de producto
- b. Riesgo del proyecto
- c. Riesgo del negocio
- d. Riesgo de programación

92. ¿Cuál de los siguientes términos se define mejor en la declaración: "Habrá un cambio en la administración de la organización con diferentes prioridades"?

- a. Rotación de personal
- b. Cambio tecnológico
- c. Cambio de gestión
- d. Concurso de productos

93. ¿Cuál de los siguientes términos se define mejor en la declaración: "La tecnología subyacente en la que se construye el sistema está reemplazada por la nueva tecnología"?

- a. Cambio tecnológico
- b. Concurso de productos
- c. Cambio de requisitos
- d. Ninguno de los mencionados

94. ¿Qué evalúa el riesgo y sus planes para la mitigación del riesgo y revíselos cuando sepa más sobre el riesgo?

- a. Monitoreo de riesgos
- b. Planificación de riesgos
- c. Análisis de riesgo
- d. Identificación de riesgo

95. ¿Cuál de los siguientes riesgos se deriva del entorno organizacional donde se está desarrollando el software?

- a. Personas riesgos
- b. Riesgos tecnológicos
- c. Riesgos de estimación
- d. Riesgos organizacionales

96. ¿Cuál de los siguientes riesgos se deriva de las tecnologías de software o hardware que se utilizan para desarrollar el sistema?

- a. Riesgos gerenciales
- b. Riesgos tecnológicos
- c. Riesgos de estimación
- d. Riesgos organizacionales

97. ¿Cuál de los siguientes términos se define mejor en la declaración: “Derive información de trazabilidad para maximizar la información que se oculta en el diseño”?

- a. Tiempo de desarrollo subestimado
- b. Reestructuración organizacional
- c. Cambios de requisitos
- d. Ninguno de los mencionados

98. ¿Cuál de las siguientes estrategias significa que se reducirá el impacto del riesgo?

- a. Estrategias de evitación
- b. Estrategias de minimización
- c. Planes de Contingencia
- d. Todos los mencionados

99. La gestión de riesgos ahora se reconoce como una de las tareas de gestión de proyectos más importantes.

- a. Cierto
- b. Falso
- c. No puedo decir
- d. Ninguna de las anteriores

100. Una evaluación del peor daño posible que podría resultar de un peligro particular se conoce como

- a. Riesgo
- b. Probabilidad de peligro
- c. Gravedad del peligro
- d. Percance



## RESPUESTAS DE LA TERCERA PRÁCTICA DE EXAMEN

1. (b) Cohesión secuencial
2. (c). Ambos ayb
3. (c) .Diagram herramientas
4. (d). Tanto a y b
5. (c) .Datos elementos
6. (b). Permitir una serie de operaciones diferentes para compartir el mismo nombd.
7. (d) flujo de transformación.
8. (c) flujo de transacción.
9. (b) Derivación de la jerarquía de control.
10. (d). Tanto b y c
11. (d) .Todos los mencionados anteriormente.
12. (a) Evaluación estricta.
13. (d) .Testing
14. (b). Pruebas de regresión.
15. (d). Pruebas basadas en subprocessos.
16. (a) .Penetra el sistema.
17. (c) .Caja negra
18. (d) .Todos los mencionados
19. (b). Condiciones previas.
20. (b). Prueba de caja blanca
21. (b) .Desarrolladores
22. (b). Prueba de caja negra
23. (b). Análisis de valor límite.
24. (c) Tamaño del módulo.
25. (b) Falso
26. (a) .Las diferentes técnicas de prueba son apropiadas en diferentes momentos
27. (d) grupo de prueba independiente
28. (c) .Metricia
29. (d) .Todos los mencionados
30. (a). Terminación de bucle no persistente
31. (d). Corte transversal.
32. (c). Las pruebas de sistemas se enfocan en probar las interfaces entre programas, las pruebas de programas se enfocan en programas individuales
33. (a) .Encontrar, que no hay incoherencia se cita en un archivo de texto y, por lo tanto, la referencia aparece en el archivo de referencia
34. (c). Tanto el sistema físico como el sistema computarizado funcionan simultáneamente.
35. (d) Etapa de análisis de requerimientos.
36. (d) .1: 10
37. (a) .1: 2
38. (d) .a solo presione hacia abajo los autómatas
39. (d) .1: 10: 100
40. (a) .Sólo se encontrarán algunos errores.
41. (d) .Todos los mencionados anteriormente.
42. (b). Personas, producto, proceso, proyecto.
43. (d) .Todos los mencionados anteriormente.
44. (d) .Todos los mencionados anteriormente.
45. (b). Naturaleza dinámica.

Esta guía de estudio, fue vendida por [www.guiaceneval.mx](http://www.guiaceneval.mx) Todos los Derechos Reservados.

46. (b). Requisitos del cliente.
47. (b). Mantenimiento perfecto.
48. (b). Tipo embebido
49. (c). Herramientas de caja baja
50. (d) .Todos los mencionados anteriormente.
51. (a). Análisis de flujo de trabajo.
- 52 (b) .Herramientas
53. (d) .Escalabilidad
54. (d) .Todo lo anterior
55. (d) .Todos los anteriores
56. (d) .Evitar quejas del cliente.
57. (b). Falso
58. (c) .Testing
59. (c). Ciclo de vida de la gestión de proyectos
60. (d) .alta
61. (c) .Estimación por analogía.
62. (a). Un modelo de composición de aplicación.
63. (c) Ley de Parkinson.
64. (d). Un modelo de diseño temprano.
65. (c). Un modelo de reutilización.
66. (b). Falso
67. (d). Ambas estimaciones basadas en PF y COCOMO
68. (b). Falso
69. (a) Modelo de escenario de diseño temprano
70. (c). Modelo de composición de la aplicación.
71. (c). Tiempo de riesgo
72. (b). Riesgos predecibles.
73. Horario de riesgo.
74. (a) .Espiral modelo.
75. (b) .2,3 y 4
76. (d) .Identificación
77. (c). Riesgo técnico.
78. (c). Riesgo empresarial.
79. (c). Ambos ayb
80. (d) .Todos los mencionados
81. (b). Generación de riesgos.
82. (a) Exposición al riesgo.
83. (d). Riesgos técnicos.
84. (b). Riesgos empresariales.
85. (c). Falta de requisitos documentados o alcance del software
86. (d) Proyección de riesgos.
87. (c). Riesgos del tamaño del producto.
88. (a). Riesgos de impacto empresarial.
89. (a). Riesgo de rendimiento.
90. (d). Gerente de proyectos.
91. (a). Riesgo del producto.
92. (c). Cambio de gestión.
93. (a). Cambio de tecnología.
94. (a) Monitoreo de riesgos.
95. (d) Riesgos organizacionales.



96. (b). Riesgos tecnológicos.
97. (c) .Requisitos de cambios.
98. (b). Estrategias de minimización.
99. (a). Ciento
100. (c). Gravedad de peligro

## CUARTA PRÁCTICA DE EXAMEN

1. ¿Cuál de los siguientes términos es una medida de la probabilidad de que el sistema cause un accidente?
  - a. Riesgo
  - b. Probabilidad de peligro
  - c. Accidente
  - d. Dañar
  
2. Una debilidad en un sistema informático que puede ser explotada para causar pérdidas o daños se conoce como?
  - a. Vulnerabilidad
  - b. Ataque
  - c. Amenaza
  - d. Exposición
  
3. Un sistema de verificación de contraseñas que no permite contraseñas de usuarios que sean nombres propios o palabras que normalmente se incluyen en un diccionario es un ejemplo de \_\_\_\_\_ con respecto a los sistemas de seguridad.
  - a. riesgo
  - b. controlar
  - c. ataque
  - d. activo
  
4. ¿Cuántas etapas hay en la especificación de requisitos impulsada por el riesgo?
  - a. Tres
  - b. cuatro
  - c. cinco
  - d. seis
  
5. ¿Cómo se define la confiabilidad del software?
  - a. hora
  - b. eficiencia
  - c. calidad
  - d. velocidad



6. Idoneidad, precisión, interpolabilidad y seguridad: ¿qué tipo de atributo de calidad es ISO 9126?

- a. Confiabilidad
- b. Eficiencia
- c. Funcionalidad
- d. Usabilidad

7. El comportamiento del tiempo y el comportamiento de los recursos se encuentran bajo el atributo de calidad de ISO 9126?

- a. Confiabilidad
- b. Eficiencia
- c. Funcionalidad
- d. Usabilidad

8. NHPP significa

- a. Producto de Poisson no homogéneo
- b. Producto de Poisson no heterogéneo
- c. Proceso de Poisson no heterogéneo
- d. Proceso de Poisson no homogéneo

9. El modelo CMM es una técnica para

- a. Mantener automáticamente la fiabilidad del softwad.
- b. Mejorar el proceso del softwad.
- c. prueba el software
- d. todos los mencionados

10. ¿Qué tipo de falla permanece en el sistema por algún tiempo y luego desaparece?

- a. Permanente
- b. Transitorio
- c. Intermitente
- d. Todos los mencionados

11. ¿Cuál de los siguientes enfoques se utilizan para lograr sistemas confiables?

- a. Prevención de fallos
- b. Eliminación de fallos
- c. Tolerancia a fallos
- d. Todos los mencionados

12. Se dice que un sistema que mantiene su integridad mientras acepta una interrupción temporal de su funcionamiento se encuentra en un estado de

- a. Tolerancia total a los fallos
- b. Degradación elegante
- c. Fail Soft
- d. A prueba de fallos

13. ¿Cuál de las siguientes comprobaciones de detección de errores no forma parte de la detección de aplicaciones?

- a. Cheques de hardware
- b. Controles de tiempo
- c. Verificaciones de reversión
- d. Controles de codificación

14. El manejo de excepciones es un tipo de

- a. mecanismo de recuperación de errores hacia adelante
- b. mecanismo de recuperación de errores hacia atrás
- c. Todos los mencionados
- d. Ninguno de los mencionados

15. La no ocurrencia de alteración impropia de la información se conoce como

- a. Confiabilidad disponible
- b. Confiabilidad Confidencial
- c. Confiabilidad mantenible
- d. Confiabilidad Integral



16. En la programación de la versión N, que es la generación independiente de N, el valor de N es

- a. mayor que 1
- b. menos que 1
- c. mayor que 2
- d. menos de 2

17. En la tolerancia a fallos basada en registros, los registros de eventos no determinados se guardan y se reproducen en caso de error.

- a. Cierto
- b. Falso
- c. No puedo decir
- d. Ninguna de las anteriores

18. Todas las técnicas tolerantes a fallas se basan en

- a. Integridad
- b. Confianza
- c. Redundancia
- d. Ninguno de los mencionados

19. Es imperativo que los procesos de comunicación alcancen puntos de recuperación consistentes para evitar el efecto \_\_\_\_\_, con un mecanismo de recuperación de errores hacia atrás.

- a. Estático
- b. Dinámica
- c. dominó
- d. Torbellino

20. ¿Cuál no es un modelo de calidad de software?

- a. ISO 9000
- b. Modelo de McCall
- c. Modelo boehm
- d. ISO 9126

21. ¿Cuántos niveles están presentes en CMM?

- a. Tres
- b. cuatro
- c. cinco
- d. seis

22. ¿Qué nivel de CMM es para la gestión de procesos?

- a. Inicial
- b. Repetible
- c. Definido
- d. Optimizando

23. En ISO 9126, el comportamiento del tiempo y la utilización de recursos son parte de

- a. mantenibilidad
- b. portabilidad
- c. eficiencia
- d. usabilidad

24. ¿Cuál de los siguientes no es un modelo probabilístico?

- a. Error de siembra
- b. NHPP
- c. Dominio de entrada
- d. Métrica del software de Halstead

25. La confiabilidad del software se define con respecto a

- a. hora
- b. loco
- c. fallas
- d. calidad



26. Failure In Time (FIT) es otra forma de informar
- a. MTTR
  - b. MTTF
  - c. MTSF
  - d. MTBF
27. El tiempo medio de reparación (MTTR) es el tiempo necesario para reparar un módulo de hardware defectuoso.
- a. Cierto
  - b. Falso
  - c. No puedo decir
  - d. Ninguna de las anteriores
28. IMC Networks es un fabricante líder y certificado de soluciones de conectividad LAN / WAN para redes ópticas y para empresas, telecomunicaciones y proveedores de servicios.
- a. Sistemas de telco
  - b. D-Link
  - c. Arista Networks
  - d. ISO 9001
29. ¿Cuáles son las cualidades de un buen software?
- a. Portabilidad
  - b. Reusabilidad
  - c. Interoperabilidad
  - d. Todas las anteriores
30. La ingeniería de software basada en la reutilización es una estrategia de ingeniería de software donde el proceso de desarrollo está orientado a reutilizar el software existente.
- a. Cierto
  - b. Falso
  - c. No puedo decir
  - d. Ninguna de las anteriores

31. El movimiento de código abierto ha significado que hay una gran base de código reutilizable disponible en

- a. libre de costo
- b. bajo costo
- c. Alto costo
- d. corto periodo de tiempo

32. Considere el ejemplo y categorícelo en consecuencia, "Un sistema de coincidencia de patrones desarrollado como parte de un sistema de procesamiento de texto puede reutilizarse en un sistema de gestión de bases de datos".

- a. Reutilización del sistema de aplicación.
- b. Reutilización de componentes
- c. Reutilización de objetos y funciones.
- d. Ninguno de los mencionados

33. COTS significa

- a. Sistemas comerciales listos para usar
- b. Estados comerciales fuera de la plataforma
- c. Estado fuera de sistema comercial
- d. Ninguno de los mencionados

34. Medios de reutilización de productos COTS

- a. Las bibliotecas de clases y funciones que implementan abstracciones de uso común están disponibles para su reutilización
- b. Los componentes compartidos se entrelazan en una aplicación en diferentes lugares cuando se compila el programa
- c. Los sistemas a gran escala que encapsulan la funcionalidad y las reglas de negocios genéricos están configurados para una organización
- d. Los sistemas se desarrollan configurando e integrando sistemas de aplicaciones existentes.

35. .NET son específicos para cada plataforma?

- a. Java
- b. Mac OS
- c. Microsoft
- d. LINUX



36. ¿Cuál de las siguientes es una estructura genérica que se extiende para crear un subsistema o aplicación más específica?

- a. Reutilización de software
- b. Lenguaje de programación orientado a objetos
- c. Marco de referencia
- d. Ninguno de los mencionados

37. "Un sistema de pedido puede adaptarse para hacer frente a un proceso de pedido centralizado en una empresa y un proceso distribuido en otra". ¿A qué categoría pertenece el ejemplo?

- a. Proceso de especialización
- b. Especialización de plataforma
- c. Especialización ambiental
- d. Especialización funcional

38. ¿Qué son los sistemas de aplicación genéricos que pueden diseñarse para soportar un tipo de negocio, actividad particular o, en ocasiones, una empresa completa?

- a. Sistemas de solución COTS
- b. Sistemas integrados COTS
- c. Sistemas ERP
- d. Tanto la solución COTS como los sistemas integrados COTS

39. ¿Cuál de las siguientes no es una ventaja de la reutilización del software?

- a. costos mas bajos
- b. desarrollo de software más rápido
- c. alta efectividad
- d. menores riesgos

40. Los marcos son un enfoque efectivo para la reutilización, pero son \_\_\_\_\_ para introducir en los procesos de desarrollo de software.

- a. difícil
- b. costoso
- c. no fidedigno
- d. difícil y caro

41. ¿Cuál de las siguientes opciones no se proporciona mediante métodos formales?

- a. proporcionando marcos
- b. sistemas de verificación
- c. proporcionar inversores
- d. Proporcionando marcos y sistemas de verificación.

42. \_\_\_\_\_ son declaraciones que se pueden interpretar de varias maneras.

- a. Contradicciones
- b. Ambigüedades
- c. Vaguedad
- d. Comentarios

43. ¿Qué define las circunstancias en que una operación particular es válida?

- a. Contradicciones
- b. Post-condición
- c. Vaguedad
- d. Ninguno de los mencionados

44. ¿Cuál de las siguientes es una manera de hacer una declaración sobre los elementos de un conjunto que es verdadero para cada miembro del conjunto?

- a. Conjunto
- b. Secuencia
- c. Cuantificación universal
- d. Ambos Set y secuencia

45. ¿Cuál de las siguientes opciones ocurre a menudo debido al volumen de un documento de especificación del sistema?

- a. Contradicciones
- b. Ambigüedades
- c. Vaguedad
- d. Incompleto



46. El \_\_\_\_\_ de un lenguaje de especificación formal a menudo se basa en una sintaxis que se deriva de la notación de la teoría de conjuntos estándar y el cálculo de predicados.

- a. dominio semantico
- b. dominio sintáctico
- c. secuencia
- d. conjunto

47. ¿Cuál de los siguientes proporciona un método conciso, no ambiguo y consistente para documentar los requisitos del sistema?

- a. CMM
- b. ISO 9001
- c. Herramientas de caja
- d. Metodos formales

48. El \_\_\_\_\_ de un lenguaje de especificación indica cómo el lenguaje representa los requisitos del sistema.

- a. dominio semantico
- b. dominio sintáctico
- c. secuencia
- d. conjunto

49. ¿Cuál de las siguientes opciones es esencial para el éxito, cuando se utilizan métodos formales por primera vez?

- a. Formación de expertos
- b. Consultante
- c. Conocimientos previos
- d. Formación de expertos y consultoría.

50. La filosofía de Cleanroom SE se centra en la eliminación de defectos en lugar de la disponibilidad de defectos.

- a. Cierto
- b. Falso
- c. No puedo decir
- d. Ninguna de las anteriores

51. ¿Cuál de los siguientes equipos de proceso de sala limpia desarrolla un conjunto de pruebas estadísticas para ejercer el software después del desarrollo?

- a. Equipo de especificación
- b. Equipo de desarrollo
- c. Equipo de certificación
- d. Todos los mencionados

52. Un elemento de software se ajusta a un modelo de componente estándar y se puede implementar y componer de forma independiente sin modificaciones de acuerdo con un estándar de composición.

- a. Cierto
- b. Falso
- c. No puedo decir
- d. Ninguna de las anteriores

53. ¿Cuál de las siguientes es una característica de CBSE?

- a. Aumenta la calidad
- b. CBSE acorta el tiempo de entrega
- c. CBSE aumenta la productividad
- d. Todos los mencionados

54. ¿Cuál de los siguientes términos se define mejor por la declaración: "Para que un componente sea compostable, todas las interacciones externas deben

- a. Estandarizado
- b. Independiente
- c. Composable
- d. Documentado

55. Un modelo de componente define estándares para

- a. propiedades
- b. métodos
- c. mecanismos
- d. todos los mencionados



56. ¿Cuál de los siguientes no es un ejemplo de tecnología de componentes?

- a. EJB
- b. COM +
- c. .RED
- d. Ninguno de los mencionados

57. ¿Cuál de los siguientes términos se define mejor por la declaración: "Las operaciones en cada lado de la interfaz tienen el mismo nombre pero sus tipos de parámetros o la cantidad de parámetros son diferentes"?

- a. Parámetro incompatibilidad
- b. Operación incompleta
- c. Incompatibilidad de operación
- d. Ninguno de los mencionados

58. ¿Cuál de los siguientes términos se define mejor por la declaración: "Los nombres de las operaciones en las interfaces 'provee' y 'requiere' son diferentes"?

- a. Parámetro incompatibilidad
- b. Operación incompleta
- c. Incompatibilidad de operación
- d. Ninguno de los mencionados

59. Un \_\_\_\_\_ define un conjunto de estándares para componentes, incluidos estándares de interfaz, estándares de uso y estándares de implementación.

- a. Ingeniería de software basada en componentes
- b. Composición de componentes
- c. Modelo de componente
- d. Interfaces de componentes

60. \_\_\_\_\_ es una forma de proporcionar funcionalidad en un servidor remoto con acceso de cliente a través de un navegador web.

- a. SaaS
- b. SOA
- c. Configurabilidad
- d. Tanto SaaS como Configurabilidad

61. ¿Qué arquitecturas descentralizadas de arquitectura en las que no hay clientes y servidores distinguidos?

- a. Arquitectura cliente-servidor multinivel
- b. Arquitectura maestro-esclavo
- c. Arquitectura de componentes distribuidos
- d. Arquitectura de igual a igual

62. La arquitectura orientada a servicios (SOA) es

- a. Fuertemente acoplado
- b. Débilmente acoplado
- c. Fuertemente cohesivo
- d. Libremente cohesivo

63. ¿Cuál de los siguientes es un principio esencial de una arquitectura?

- a. Consistencia
- b. Confiabilidad
- c. Escalabilidad
- d. Todos los mencionados

64. Organice las siguientes actividades para construir una SOA.

- a. i, ii, iii, iv
- b. iii, ii, i, iv
- c. ii, iii, i, iv
- d. ii, iii, iv, i

65. ¿En qué se diferencia SOA de la arquitectura OO?

- a. Fuerte acoplamiento entre objetos
- b. Las comunicaciones son prescriptivas más que descriptivas.
- c. Los datos están separados de un servicio o comportamiento.
- d. Los datos y métodos están integrados en un solo objeto.



66. ¿Qué arquitectura se construirá sobre una SOA?

- a. La arquitectura de la aplicación
- b. La arquitectura de servicio
- c. La arquitectura de componentes
- d. Ninguno de los mencionados

67. ¿Cuál de las siguientes utilidades no es parte de Application Service Layer?

- a. Implementación de políticas
- b. QoS
- c. Seguridad
- d. Verificar factura

68. ¿Cuál de las siguientes utilidades no es parte de Business Service Layer?

- a. Servicio centrado en la tarea.
- b. Servicios de envoltura
- c. Obtener información de la cuenta
- d. Servicio centrado en la entidad

69. Podemos construir una arquitectura orientada a servicios (SOA) usando lenguaje orientado a objetos (OO)

- a. Cierto
- b. Falso
- c. No puedo decir
- d. Ninguna de las anteriores

70. Si detecta un fallo de alimentación al controlar un nivel de voltaje, debe realizar más de una observación para detectar que el voltaje está disminuyendo.

- a. Cierto
- b. Falso
- c. No puedo decir
- d. Ninguna de las anteriores

71. El tiempo de ejecución promedio del proceso del monitor de potencia debe ser inferior a

- a. 1 ms
- b. 10ms
- c. 100ms
- d. ninguno de los mencionados

72. ¿Cuál de los siguientes diagramas puede ayudar a detectar cortes de puntos?

- a. Diagrama de clase
- b. Diagrama de objetos
- c. Diagrama de secuencia
- d. Diagrama ER

73. ¿Cuál de los siguientes se representa como un aspecto que solicita un nombre de inicio de sesión y una contraseña?

- a. Clase
- b. Objeto
- c. Autenticacion de usuario
- d. Todos los mencionados

74. La investigación y el desarrollo en orientación de aspectos se han centrado

- a. reingeniería de software
- b. programación artificial
- c. Programación Orientada a Aspectos
- d. todos los mencionados

Respuesta: (c). Orientación orientada a la perspectiva.

- a. Separación de intereses
- b. Aspectos de escritura
- c. Encontrar la complejidad del código
- d. Ninguno de los mencionados



76. ¿Cuál de los siguientes no es un tipo de interés de las partes interesadas?

- a. Preocupaciones funcionales
- b. Preocupaciones por la calidad del servicio.
- c. Preocupación política
- d. Preocupación no funcional

77. ¿Cuál de las siguientes preocupaciones se adapta mejor a la siguiente declaración: "El sistema de banca por Internet incluye nuevos requisitos del cliente, requisitos de cuenta, requisitos de gestión del cliente, requisitos de seguridad, requisitos de recuperación, etc."?

- a. Preocupaciones funcionales
- b. Preocupaciones por la calidad del servicio.
- c. Preocupaciones del sistema
- d. Preocupaciones transversales

78. ¿Cuál de las siguientes es la preocupación principal en el sistema de gestión de registros médicos?

- a. mantener registros de pacientes
- b. diagnostico y tratamientos
- c. consultas
- d. todos los mencionados

79. Un evento en un programa de ejecución donde el asesoramiento asociado con un aspecto puede ejecutarse se conoce como

- a. aspecto
- b. punto de unión
- c. modelo de punto de unión
- d. punto de corte

80. ¿Cuál de los siguientes servicios no es proporcionado por un objeto?

- a. Activar y desactivar objetos
- b. Características de seguridad
- c. Archivos que implementan las entidades identificadas dentro del ERD.
- d. Registro de implementación de objeto.

81. ¿Cuál de los siguientes términos se define mejor por la declaración: "Cuando un objeto invoca a otro objeto independiente, se pasa un mensaje entre los dos objetos"?

- a. Pareja de control
- b. Objeto de aplicación
- c. Pareja de datos
- d. Objeto de base de datos

82. CORBA significa

- a. Solicitud de objeto común Construir arquitectura
- b. Arquitectura de agente de solicitud de objetos comunes
- c. Solicitud de objeto común Break Architecture
- d. Todos los mencionados

83. ¿Qué atributo de la aplicación web está definido por la declaración: "Un gran número de usuarios pueden acceder a la aplicación web a la vez"?

- a. Carga impredecible
- b. Actuación
- c. Concurrencia
- d. Intensividad de la red

84. ¿Qué atributo de la aplicación web se define en la declaración: "La calidad y la naturaleza estética del contenido sigue siendo un determinante importante de la calidad de una aplicación web"?

- a. Disponibilidad
- b. Datos impulsados
- c. Contenido sensible
- d. Evolución continua

85. Si el usuario consulta una colección de grandes bases de datos y extrae información de la aplicación web, la aplicación se clasifica en

- a. Aplicación orientada al servicio
- b. Aplicación de acceso a la base de datos
- c. Aplicación de portal
- d. Aplicación de almacenamiento de datos



86. ¿Qué modelo de proceso se debe utilizar en prácticamente todas las situaciones de ingeniería web?

- a. Modelo incremental
- b. Modelo de cascada
- c. Modelo espiral
- d. Ninguno de los mencionados

87. ¿Qué análisis forma parte del modelo de análisis del marco del proceso de ingeniería web?

- a. Análisis de contenido
- b. Analisis de interaccion
- c. Análisis funcional
- d. Todos los mencionados

88. El desarrollo web y el desarrollo de software son lo mismo.

- a. Ciento
- b. Falso
- c. No puedo decir
- d. Ninguna de las anteriores

89. Los sistemas basados en la web a menudo están orientados a documentos y contienen contenido estático o dinámico.

- a. Ciento
- b. Falso
- c. No puedo decir
- d. Ninguna de las anteriores

90. El núcleo de la ingeniería inversa es una actividad llamada

- a. código de reestructuración
- b.direccionalidad
- c.extraer abstracciones
- d.interactividad

91. ¿Qué se ha convertido en una exigencia para los productos y sistemas informáticos de todo tipo?

- a. GUIs
- b. Llaves del candidato
- c. Modelo de objeto
- d. Todos los mencionados

92. La ingeniería delantera también se conoce como

- a. extraer abstracciones
- b. renovación
- c. recuperación
- d. Renovación y recuperación.

93. La ingeniería inversa es el proceso de derivar el diseño y la especificación del sistema de su

- a. GUI
- b. Base de datos
- c. Código fuente
- d. Todos los mencionados

94. Las técnicas de ingeniería inversa para datos de programas internos se centran en la definición de clases de objetos.

- a. Ciento
- b. Falso
- c. No puedo decir
- d. Ninguna de las anteriores

95. ¿Cuál de los siguientes pasos no se puede usar para definir el modelo de datos existente como un precursor para rediseñar un nuevo modelo de base de datos?

- a. Construir un modelo de objeto inicial
- b. Determinar las claves del candidato
- c. Refinar las clases tentativas.
- d. Descubrir interfaces de usuario



96. Gran parte de la información necesaria para crear un modelo de comportamiento puede obtenerse observando la manifestación externa de los existentes.

- a. claves candidatas
- b.interfaz
- c.estructura de base de datos
- d.ninguno de los mencionados

97. A veces se llama extracción de elementos de datos y objetos para obtener información sobre el flujo de datos y para comprender las estructuras de datos existentes que se han implementado.

- a. análisis de los datos
- b.direccionalidad
- c.extracción de datos
- d.aplicaciones cliente

98. La ingeniería inversa y la reingeniería son procesos equivalentes de la ingeniería de software.

- a. Ciento
- b.Falso
- c.No puedo decir
- d.Ninguna de las anteriores

99. La transformación de un sistema de una forma de representación a otra se conoce como

- a. Re-factoring
- b.Reestructuración
- c.Ingeniería delantera
- d.Re-factoring y Reestructuración

100. Tabla de decisiones

- a. muestra los caminos de decisión
- b.representa un flujo de información
- c.Obtiene una imagen precisa del sistema.
- d.Reglas de documentos que seleccionan una o más acciones basadas en una o más condiciones de un conjunto de condiciones posibles

Respuesta: (d) reglas de .documents que seleccionan una o más acciones basadas en una o más condiciones de un conjunto de condiciones posibles

**Esta guía de estudio, fue vendida por [www.guiaaceneval.mx](http://www.guiaaceneval.mx) Todos los Derechos Reservados.**

## RESPUESTAS DE LA CUARTA PRÁCTICA DE EXAMEN

1. Respuesta: (a) .Riesgo
2. Respuesta: (a) .Vulnerabilidad
3. Respuesta: (b) .control
4. Respuesta: (b) .cuatro
5. Respuesta: (a). hora
6. Respuesta: (c). Funcionalidad.
7. Respuesta: (b) .Eficiencia
8. Respuesta: (d). Proceso no homogéneo de Poisson
9. Respuesta: (b). Mejorar el proceso del softwad.
10. Respuesta: (b) .Transient
11. Respuesta: (d) .Todos los mencionados
12. Respuesta: (d) .Fail Safe
13. Respuesta: (a) cheques de hardware
14. Respuesta: (a) mecanismo de recuperación de errores hacia adelante
15. Respuesta: (d). Confiabilidad integral.
16. Respuesta: (c). Mayor que 2
17. Respuesta: (a). Ciento
18. Respuesta: (c) .Redundancia
19. Respuesta: (c) .Domino
20. Respuesta: (a) .ISO 9000
21. Respuesta: (c) .five
22. Respuesta: (d) .Optimización
23. Respuesta: (c) .eficiencia
24. Respuesta: (d). Métrica del software de Halstead
25. Respuesta: (a). hora
26. Respuesta: (d) .MTBF
27. Respuesta: (a). Ciento
28. Respuesta: (a) .Telco Systems
29. Respuesta: (d) .Todo lo anterior
30. Respuesta: (a).
31. Respuesta: (b) costo bajo
32. Respuesta: (b). Reutilización de componentes.
33. Respuesta: (a) .Sistemas comerciales listos para usar
34. Respuesta: (d). Los sistemas se desarrollan mediante la configuración e integración de los sistemas de aplicación existentes.
35. Respuesta: (c) .Microsoft
36. Respuesta: (c) .Framework
37. Respuesta: (a) .Proceso de especialización.
38. Respuesta: (a) sistemas de solución .COTS
39. Respuesta: (c). Alta efectividad.
40. Respuesta: (d). Difícil y costoso.
41. Respuesta: (d). Ambos proporcionan marcos y sistemas de verificación.
42. Respuesta: (a) .Contradicciones.
43. Respuesta: (d) .Ninguno de los mencionados
44. Respuesta: (c). Cuantificación universal.
45. Respuesta: (c). Vaguedad.
46. Respuesta: (b) dominio .sintáctico



47. Respuesta: (d). Métodos normales.
48. Respuesta: (a) dominio .semantic
49. Respuesta: (d). Tanto la formación de expertos como la consultoría.
50. Respuesta: (b). Falso
51. Respuesta: (b). Equipo de desarrollo.
52. Respuesta: (a). Ciento
53. Respuesta: (d) .Todos los mencionados
54. Respuesta: (c) .Componible
55. Respuesta: (d) .Todos los mencionados
56. Respuesta: (d) .Ninguno de los mencionados
57. Respuesta: (a) .Incompatibilidad del parámetro.
58. Respuesta: (c) .Ocompatibilidad operativa.
59. Respuesta: (c) .Componente modelo.
60. Respuesta: (a) .SaaS
61. Respuesta: (d) Arquitectura de igual a igual hitecture
62. Respuesta: (b). Débilmente acoplado
63. Respuesta: (d) .Todos los mencionados
64. Respuesta: (c) .ii, iii, i, iv
65. Respuesta: (c). Los datos están separados de un servicio o comportamiento
66. Respuesta: (a). La arquitectura de la aplicación
67. Respuesta: (d) .Verificar factura.
68. Respuesta: (b). Servicios de envoltura.
69. Respuesta: (a). Ciento
70. Respuesta: (a). Ciento
71. Respuesta: (a) .1ms
72. Respuesta: (b). Diagrama de objeto.
73. Respuesta: (c) .User autenticación
74. Respuesta: (c). Orientación orientada a la perspectiva.
75. Respuesta: (a). Separación de inquietudes.
76. Respuesta: (a). Preocupaciones funcionales.
77. Respuesta: (d). Preocupaciones transversales
78. Respuesta: (a) Mantener registros de pacientes.
79. Respuesta: (b) .juntar punto
80. Respuesta: (c). Archivos que implementan las entidades identificadas dentro del ERD.
81. Respuesta: (c). Pareja de datos.
82. Respuesta: (b). Arquitectura de intermediario de solicitud de objetos comunes
83. Respuesta: (c). Conurrencia
84. Respuesta: (c) .Contenido sensible
85. Respuesta: (d) .Aplicación de almacenamiento de datos.
86. Respuesta: (a). Modelo incremental
87. Respuesta: (d) .Todos los mencionados
88. Respuesta: (b). Falso
89. Respuesta: (a). Ciento
90. Respuesta: (c). Extraer abstracciones.
91. Respuesta: (a) .GUIs
92. Respuesta: (d) .Buena renovación y recuperación.
93. Respuesta: (c). Código fuente
94. Respuesta: (a). Ciento
95. Respuesta: (d) .Descubrir interfaces de usuario.
96. Respuesta: (b) .Interfaz

97. Respuesta: (a) análisis de datos.

98. Respuesta: (b). Falso

99. Respuesta: (d). Tanto la re-factorización como la reestructuración.

100. Respuesta: (d) reglas de documentos que seleccionan una o más acciones basadas en una o más condiciones de un conjunto de condiciones posibles



## **ANTOLOGÍA**

*Esta guía de estudio, fue vendida por [www.guiaceneval.mx](http://www.guiaceneval.mx) Todos los Derechos Reservados.*

# CONTENIDO

- Lectura 1. Principios que guían la práctica
- Lectura 2. Comprensión de los requerimientos
- Lectura 3. Modelado de los requerimientos: flujos
- Lectura 4. Modelado de los requerimientos: escenarios
- Lectura 5. Diseño e implementación 1
- Lectura 6. Diseño e implementación 2
- Lectura 7. Gestión de proyectos
- Lectura 8. Gestión de la calidad
- Lectura 9. Arquitectura de redes de información
- Lectura 10. Modelos de gestión de red
- Lectura 11. Gestión y planificación de redes
- Lectura 12. Gestión de Redes
- Lectura 13. SISTEMAS OPERATIVOS



## **Lectura 1. Principios que guían la práctica**

CAPÍTULO

4

## PRINCIPIOS QUE GUÍAN LA PRÁCTICA



**SEP**

SECRETARÍA DE  
EDUCACIÓN PÚBLICA



CENTRO NACIONAL  
DE EVALUACIÓN PARA  
LA EDUCACIÓN SUPERIOR, A.C.



**EDITORIAL:**  
**guiaceneval.mx**



## CONCEPTOS CLAVE

Principios fundamentales ....	83
Principios que gobiernan lo siguiente:	
codificación .....	94
comunicación .....	86
despliegue .....	96
diseño .....	92
modelado .....	90
planeación .....	88
pruebas.....	95
requerimientos .....	91

n un libro que explora las vidas y pensamientos de los **ingenieros** de software, Ellen Ullman [Ull97] ilustra una parte de su vida con el relato de lo que piensa un profesional del software cuando está bajo presión:

No tengo idea de la hora que es. En esta oficina no hay ventanas ni reloj, sólo la pantalla de un horno de microondas que parpadea su LED de color rojo: 12:00, 12:00, 12:00. Joel y yo hemos estado programando durante varios días. Tenemos una falla, endemoniada y testaruda. Así que nos sentimos bien con el pulso rojo sin tiempo, como si fuera un pasmo de nuestros cerebros, de algún modo sincronizados al mismo ritmo del parpadeo...

¿En qué estamos trabajando? Los detalles se me escapan. Tal vez ayudamos a personas pobres y enfermas o mejoramos un conjunto de rutinas de bajo nivel de un protocolo de base de datos distribuida, no me importa. Debería importarme; en otra parte de mi ser —más tarde, quizás cuando salga de este cuarto lleno de computadoras— me preocuparé mucho de por qué y para quién y con qué propósito estoy escribiendo software. Pero ahora, no. He cruzado una membrana tras la que el mundo real y sus asuntos ya no importan. Soy ingeniera de software.

La anterior es una imagen tenebrosa de la práctica de la ingeniería de software, pero si se detienen un poco a pensarla, muchos de los lectores de este libro se verán reflejados en ella.

Las personas que elaboran software de cómputo practican el arte, artesanía o disciplina<sup>1</sup> conocida como ingeniería de software. Pero, ¿qué es la “práctica” de la ingeniería de software? En un sentido general, es un conjunto de conceptos, principios, métodos y herramientas a los que un ingeniero de software recurre en forma cotidiana. La práctica permite que los gerentes

## UNA

### MIRADA RÁPIDA

**¿Qué es?** La práctica de la ingeniería de software es un conjunto amplio de principios, conceptos, métodos y herramientas que deben considerarse al planear y desarrollar software.

**¿Quién lo hace?** Los profesionales (ingenieros de software) y sus gerentes realizan varias tareas de ingeniería de software.

**¿Por qué es importante?** El proceso de software proporciona a todos los involucrados en la creación de un sistema o producto basado en computadora un mapa para llegar con éxito al destino. La práctica proporciona los detalles que se necesitarán para circular por la carretera. Indica dónde se localizan los puentes, los caminos cerrados y las bifurcaciones. Ayuda a entender los conceptos y principios que deben entenderse y seguirse a fin de llegar con seguridad y rapidez. Enseña a manejar, dónde disminuir la velocidad y en qué lugares acelerar. En el contexto de la ingeniería de software, la práctica es lo que se hace día tras día conforme el software evoluciona de idea a realidad.

**¿Cuáles son los pasos?** Son tres los elementos de la práctica que se aplican sin importar el modelo de proceso que se elija. Se trata de: principios, conceptos y métodos. Un cuarto elemento de la práctica —las herramientas— da apoyo a la aplicación de los métodos.

**¿Cuál es el producto final?** La práctica incluye las actividades técnicas que generan todos los productos del trabajo definidos por el modelo del proceso de software que se haya escogido.

**¿Cómo me aseguro de que lo hice bien?** En primer lugar, hay que tener una comprensión sólida de los principios que se aplican al trabajo (por ejemplo, el diseño) en cuestión. Después, asegúrese de que se escogió el método apropiado para el trabajo, use herramientas automatizadas cuando sean adecuadas para la tarea y sea firme respecto de la necesidad de técnicas de aseguramiento de la calidad de los productos finales que se generen.

1 Algunos escritores afirman que cualquiera de estos términos excluye a los otros. En realidad, la ingeniería de software es las tres cosas.

administren proyectos de software y que los ingenieros de software elaboren programas de cómputo. La práctica da al modelo del proceso de software el saber técnico y administrativo para realizar el trabajo. La práctica transforma un enfoque caprichoso y disperso en algo más organizado, más eficaz y con mayor probabilidad de alcanzar el éxito.

A lo largo de lo que resta del libro se estudiarán distintos aspectos de la práctica de la ingeniería de software. En este capítulo, la atención se pone en los principios y conceptos que la guían en lo general.

## 4.1 CONOCIMIENTO DE LA INGENIERÍA DE SOFTWARE

En un editorial publicado hace diez años en *IEEE Software*, Steve McConnell [McC99] hizo el siguiente comentario:

Muchos trabajadores del software piensan que el conocimiento de la ingeniería de software casi exclusivamente consiste en tecnologías específicas: Java, Perl, html, C++, Linux, Windows NT, etc. Para programar computadoras es necesario conocer los detalles tecnológicos específicos. Si alguien pide al lector que escriba un programa en C++, tiene que saber algo sobre este lenguaje a fin de que el programa funcione.

Es frecuente escuchar que el conocimiento del desarrollo de software tiene una vida media de tres años, lo que significa que la mitad de lo que es necesario saber hoy será obsoleto dentro de tres años. En el dominio del conocimiento relacionado con la tecnología es probable que eso se cumpla. Pero hay otra clase de conocimiento de desarrollo de software —algo que el autor considera como los “principios de la ingeniería de software”— que no tiene una vida media de tres años. Es factible que dichos principios sirvan al programador profesional durante toda su carrera.

McConnell continúa y plantea que el cuerpo de conocimientos de la ingeniería de software (alrededor del año 2000) ha evolucionado para convertirse en un “núcleo estable” que representa cerca de “75% del conocimiento necesario para desarrollar un sistema complejo”. Pero, ¿qué es lo que hay dentro de ese núcleo estable?

Como dice McConnell, los principios fundamentales —ideas elementales que guían a los ingenieros de software en el trabajo que realizan— dan ahora un fundamento a partir del cual pueden aplicarse y evaluarse los modelos, métodos y herramientas de ingeniería.

## 4.2 PRINCIPIOS FUNDAMENTALES



Cita:

“En teoría no hay diferencia entre la teoría y la práctica. Pero en la práctica sí la hay.”

Jan van de Snepscheut

La práctica de la ingeniería de software está guiada por un conjunto de principios fundamentales que ayudan en la aplicación del proceso de software significativo y en la ejecución de métodos eficaces de ingeniería de software. En el nivel del proceso, los principios fundamentales establecen un fundamento filosófico que guía al equipo de software cuando realiza actividades estructurales y sombrilla, cuando navega por el flujo del proceso y elabora un conjunto de productos del trabajo de la ingeniería de software. En el nivel de la práctica, los principios fundamentales definen un conjunto de valores y reglas que sirven como guía cuando se analiza un problema, se diseña una solución, se implementa y prueba ésta y cuando, al final, se entrega el software a la comunidad de usuarios.

En el capítulo 1 se identificó un conjunto de principios generales que amplían el proceso y práctica de la ingeniería de software: 1) agregar valor para los usuarios finales, 2) mantenerlo sencillo, 3) fijar la visión (del producto y el proyecto), 4) reconocer que otros consumen (y deben entender) lo que usted produce, 5) abrirse al futuro, 6) planear la reutilización y 7) ¡pensar! Aunque estos principios generales son importantes, se caracterizan en un nivel tan alto de abstractitud que en ocasiones son difíciles de traducir en la práctica cotidiana de la ingeniería de



software. En las subsecciones que siguen se analizan con más detalle los principios fundamentales que guían el proceso y la práctica.

### Principios que guían el proceso

En la parte 1 de este libro se estudia la importancia del proceso de software y se describen los abundantes modelos de proceso que se han propuesto para hacer el trabajo de ingeniería de software. Sin que importe que un modelo sea lineal o iterativo, prescriptivo o ágil, puede caracterizarse con el empleo de la estructura general del proceso aplicable a todos los modelos de proceso. Los siguientes principios fundamentales se aplican a la estructura y, por extensión, a todo proceso de software:



*Todo proyecto y equipo son únicos. Esto significa que debe adaptar el proceso para que se ajuste mejor a sus necesidades.*

**Principio 1. Ser ágil.** Ya sea que el modelo de proceso que se elija sea prescriptivo o ágil, son los principios básicos del desarrollo ágil los que deben gobernar el enfoque. Todo aspecto del trabajo que se haga debe poner el énfasis en la economía de acción: en mantener el enfoque técnico tan sencillo como sea posible, hacer los productos del trabajo que se generan tan concisos como se pueda y tomar las decisiones localmente, siempre que sea posible.

**Principio 2. En cada etapa, centrarse en la calidad.** La condición de salida para toda actividad, acción y tarea del proceso debe centrarse en la calidad del producto del trabajo que se ha generado.

**Principio 3. Estar listo para adaptar.** El proceso no es una experiencia religiosa, en él no hay lugar para el dogma. Cuando sea necesario, adapte su enfoque a las restricciones impuestas por el problema, la gente y el proyecto en sí.

**Principio 4. Formar un equipo eficaz.** El proceso y práctica de la ingeniería de software son importantes, pero el objetivo son las personas. Forme un equipo con organización propia en el que haya confianza y respeto mutuos.

**Principio 5. Establecer mecanismos para la comunicación y coordinación.** Los proyectos fallan porque la información importante cae en las grietas o porque los participantes no coordinan sus esfuerzos para crear un producto final exitoso. Éstos son aspectos de la administración que deben enfrentarse.

**Principio 6. Administrar el cambio.** El enfoque puede ser formal o informal, pero deben establecerse mecanismos para administrar la forma en la que los cambios se solicitan, evalúan, aprueban e implementan.

**Principio 7. Evaluar el riesgo.** Son muchas las cosas que pueden salir mal cuando se desarrolla software. Es esencial establecer planes de contingencia.

**Principio 8. Crear productos del trabajo que agreguen valor para otros.** Sólo genere aquellos productos del trabajo que agreguen valor para otras actividades, acciones o tareas del proceso. Todo producto del trabajo que se genere como parte de la práctica de ingeniería de software pasará a alguien más. La lista de las funciones y características requeridas se dará a la persona (o personas) que desarrollará(n) un diseño, el diseño pasará a quienes generan código y así sucesivamente. Asegúrese de que el producto del trabajo imparte la información necesaria sin ambigüedades u omisiones.

La parte 4 de este libro se centra en aspectos de la administración del proyecto y del proceso, y analiza en detalle varios aspectos de cada uno de dichos principios.

### Principios que guían la práctica

La práctica de la ingeniería de software tiene un solo objetivo general: entregar a tiempo software operativo de alta calidad que contenga funciones y características que satisfagan las ne-

cesidades de todos los participantes. Para lograrlo, debe adoptarse un conjunto de principios fundamentales que guien el trabajo técnico. Estos principios tienen mérito sin que importen los métodos de análisis y diseño que se apliquen, ni las técnicas de construcción (por ejemplo, el lenguaje de programación o las herramientas automatizadas) que se usen o el enfoque de verificación y validación que se elija. Los siguientes principios fundamentales son vitales para la práctica de la ingeniería de software:

## PUNTO CLAVE

Los problemas son más fáciles de resolver cuando se subdividen en entidades separadas, distintas entre sí, solucionables individualmente y verificables.

**Principio 1. Divide y vencerás.** Dicho en forma más técnica, el análisis y el diseño siempre deben enfatizar la *separación de entidades* (SdE). Un problema grande es más fácil de resolver si se divide en un conjunto de elementos (o *entidades*). Lo ideal es que cada entidad entregue funcionalidad distinta que pueda desarrollarse, y en ciertos casos validarse, independientemente de otras entidades.

**Principio 2. Entender el uso de la abstracción.** En su parte medular, una abstracción es una simplificación de algún elemento complejo de un sistema usado para comunicar significado en una sola frase. Cuando se usa la abstracción *hoja de cálculo*, se supone que se comprende lo que es una hoja de cálculo, la estructura general de contenido que presenta y las funciones comunes que se aplican a ella. En la práctica de la ingeniería de software, se usan muchos niveles diferentes de abstracción, cada uno de los cuales imparte o implica significado que debe comunicarse. En el trabajo de análisis y diseño, un equipo de software normalmente comienza con modelos que representan niveles elevados de abstracción (por ejemplo, una hoja de cálculo) y poco a poco los refina en niveles más bajos de abstracción (como una *columna* o la función *SUM*).

Joel Spolsky [Spo02] sugiere que “todas las abstracciones no triviales hasta cierto punto son esquivas”. El objetivo de una abstracción es eliminar la necesidad de comunicar detalles. Pero, en ocasiones, los efectos problemáticos precipitados por estos detalles se “filtran” por todas partes. Sin la comprensión de los detalles, no puede diagnosticarse con facilidad la causa de un problema.

**Principio 3. Buscar la coherencia.** Ya sea que se esté creando un modelo de los requerimientos, se desarrolle un diseño de software, se genere código fuente o se elaboren casos de prueba, el principio de coherencia sugiere que un contexto familiar hace que el software sea más fácil de usar. Como ejemplo, considere el diseño de una interfaz de usuario para una *webapp*. La colocación consistente de opciones de menú, el uso de un esquema coherencia de color y el uso coherencia de íconos reconocibles ayudan a hacer que la interfaz sea muy buena en el aspecto ergonómico.

**Principio 4. Centrarse en la transferencia de información.** El software tiene que ver con la transferencia de información: de una base de datos a un usuario final, de un sistema heredado a una *webapp*, de un usuario final a una interfaz gráfica de usuario (GUI, por sus siglas en inglés), de un sistema operativo a una aplicación, de un componente de software a otro... la lista es casi interminable. En todos los casos, la información fluye a través de una interfaz, y como consecuencia hay posibilidades de cometer errores, omisiones o ambigüedades. Este principio implica que debe ponerse atención especial al análisis, diseño, construcción y prueba de las interfaces.

**Principio 5. Construir software que tenga modularidad eficaz.** La separación de entidades (principio 1) establece una filosofía para el software. La *modularidad* proporciona un mecanismo para llevar a cabo dicha filosofía. Cualquier sistema complejo puede dividirse en módulos (componentes), pero la buena práctica de la ingeniería de software demanda más. La modularidad debe ser *eficaz*. Es decir, cada módulo debe centrarse exclusivamente en un aspecto bien delimitado del sistema: debe ser cohesivo en su función o restringido en el contenido que representa. Además, los módulos deben estar interconectados en forma





*Use patrones (véase el capítulo 12) a fin de acumular conocimiento y experiencia para las futuras generaciones de ingenieros de software.*

relativamente sencilla: cada módulo debe tener poco acoplamiento con otros módulos, fuentes de datos y otros aspectos ambientales.

**Principio 6. Buscar patrones.** Brad Appleton [App00] sugiere que:

El objetivo de los patrones dentro de la comunidad de software es crear un cúmulo de bibliografía que ayude a los desarrolladores de software a resolver problemas recurrentes que surgen a lo largo del desarrollo. Los patrones ayudan a crear un lenguaje compartido para comunicar perspectiva y experiencia acerca de dichos patrones y sus soluciones. La codificación formal de estas soluciones y sus relaciones permite acumular con éxito el cuerpo de conocimientos que define nuestra comprensión de las buenas arquitecturas que satisfacen las necesidades de sus usuarios.

**Principio 7. Cuando sea posible, representar el problema y su solución desde varias perspectivas diferentes.** Cuando un problema y su solución se estudian desde varias perspectivas distintas, es más probable que se tenga mayor visión y que se detecten los errores y omisiones. Por ejemplo, un modelo de requerimientos puede representarse con el empleo de un punto de vista orientado a los datos, a la función o al comportamiento (véanse los capítulos 6 y 7).

Cada uno brinda un punto de vista diferente del problema y de sus requerimientos.

**Principio 8. Tener en mente que alguien dará mantenimiento al software.** El software será corregido en el largo plazo, cuando se descubran sus defectos, se adapte a los cambios de su ambiente y se mejore en el momento en el que los participantes pidan más capacidades. Estas actividades de mantenimiento resultan más fáciles si se aplica una práctica sólida de ingeniería de software a lo largo del proceso de software.

Estos principios no son todo lo que se necesita para elaborar software de alta calidad, pero establecen el fundamento para todos los métodos de ingeniería de software que se estudian en este libro.

### 4.3 PRINCIPIOS QUE GUÍAN TODA ACTIVIDAD ESTRUCTURAL



Cita:

"El ingeniero ideal es una mezcla... no es un científico, no es un matemático, no es un sociólogo ni un escritor; pero para resolver problemas de ingeniería utiliza conocimiento y técnicas de algunas o de todas esas disciplinas."

N. W. Dougherty

En las secciones que siguen se consideran los principios que tienen mucha relevancia para el éxito de cada actividad estructural genérica, definida como parte del proceso de software. En muchos casos, los principios que se estudian para cada una de las actividades estructurales son un refinamiento de los principios presentados en la sección 4.2. Tan sólo son principios fundamentales planteados en un nivel más bajo de abstracción.

#### municación

Antes de que los requerimientos del cliente se analicen, modelen o especifiquen, deben recaerse a través de la actividad de comunicación. Un cliente tiene un problema que parece abordable mediante una solución basada en computadora. Usted responde a la solicitud de ayuda del cliente. Ha comenzado la comunicación. Pero es frecuente que el camino que lleva de la comunicación a la comprensión esté lleno de agujeros.

La comunicación efectiva (entre colegas técnicos, con el cliente y otros participantes, y con los gerentes de proyecto) se encuentra entre las actividades más difíciles que deben enfrentarse. En este contexto, aquí se estudian principios de comunicación aplicados a la comunicación con el cliente. Sin embargo, muchos de ellos se aplican por igual en todas las formas de comunicación que ocurren dentro de un proyecto de software.

**Principio 1. Escuchar.** Trate de centrarse en las palabras del hablante en lugar de formular su respuesta a dichas palabras. Si algo no está claro, pregunte para aclararlo, pero evite las interrupciones constantes. Si una persona habla,  *nunca parezca usted beligerante en sus palabras o actos (por ejemplo, con giros de los ojos o movimientos de la cabeza).*



Antes de comunicarse, asegúrese de que entiende el punto de vista de la otra parte, conozca un poco sus necesidades y después escuche.



#### Cita:

"Las preguntas directas y las respuestas directas son el camino más corto hacia las mayores perplejidades."

Mark Twain

**Principio 2. Antes de comunicarse, prepararse.** Dedique algún tiempo a entender el problema antes de reunirse con otras personas. Si es necesario, haga algunas investigaciones para entender el vocabulario propio del negocio. Si tiene la responsabilidad de conducir la reunión, prepare una agenda antes de que ésta tenga lugar.

**Principio 3. Alguien debe facilitar la actividad.** Toda reunión de comunicación debe tener un líder (facilitador) que: 1) mantenga la conversación en movimiento hacia una dirección positiva, 2) sea un mediador en cualquier conflicto que ocurra y 3) garantice que se sigan otros principios.

**Principio 4. Es mejor la comunicación cara a cara.** Pero por lo general funciona mejor cuando está presente alguna otra representación de la información relevante. Por ejemplo, un participante quizás genere un dibujo o documento en "borrador" que sirva como centro de la discusión.

**Principio 5. Tomar notas y documentar las decisiones.** Las cosas encuentran el modo de caer en las grietas. Alguien que participe en la comunicación debe servir como "secretario" y escribir todos los temas y decisiones importantes.

**Principio 6. Perseguir la colaboración.** La colaboración y el consenso ocurren cuando el conocimiento colectivo de los miembros del equipo se utiliza para describir funciones o características del producto o sistema. Cada pequeña colaboración sirve para generar confianza entre los miembros del equipo y crea un objetivo común para el grupo.

**Principio 7. Permanecer centrado; hacer módulos con la discusión.** Entre más personas participen en cualquier comunicación, más probable es que la conversación salte de un tema a otro. El facilitador debe formar módulos de conversación para abandonar un tema sólo después de que se haya resuelto (sin embargo, considere el principio 9).

**Principio 8. Si algo no está claro, hacer un dibujo.** La comunicación verbal tiene sus límites. Con frecuencia, un esquema o dibujo arroja claridad cuando las palabras no bastan para hacer el trabajo.

**Principio 9. a) Una vez que se acuerde algo, avanzar. b) Si no es posible ponerse de acuerdo en algo, avanzar. c) Si una característica o función no está clara o no puede aclararse en el momento, avanzar.** La comunicación, como cualquier actividad de ingeniería de software, requiere tiempo. En vez de hacer iteraciones sin fin, las personas que participan deben reconocer que hay muchos temas que requieren análisis (véase el principio 2) y que "avanzar" es a veces la mejor forma de tener agilidad en la comunicación.

**Principio 10. La negociación no es un concurso o un juego. Funciona mejor cuando las dos partes ganan.** Hay muchas circunstancias en las que usted y otros participantes deben negociar funciones y características, prioridades y fechas de entrega. Si el equipo ha



¿Qué pasa si no puede llegarse a un acuerdo con el cliente en algún aspecto relacionado con el proyecto?



#### La diferencia entre los clientes y los usuarios finales

Los ingenieros de software se comunican con muchos participantes diferentes, pero los clientes y los usuarios finales son quienes tienen el efecto más significativo en el trabajo técnico que se desarrollará. En ciertos casos, el cliente y el usuario final son la misma persona, pero para muchos proyectos son individuos distintos que trabajan para diferentes gerentes en distintas organizaciones de negocios.

Un cliente es la persona o grupo que 1) solicitó originalmente que se construyera el software, 2) define los objetivos generales del negocio para el software, 3) proporciona los requerimientos básicos del

producto y 4) coordina la obtención de fondos para el proyecto. En un negocio de productos o sistema, es frecuente que el cliente sea el departamento de mercadotecnia. En un ambiente de tecnologías de la información (TI), el cliente tal vez sea un componente o departamento del negocio.

Un usuario final es la persona o grupo que 1) usará en realidad el software que se elabore para lograr algún propósito del negocio y 2) definirá los detalles de operación del software de modo que se alcance el propósito del negocio.

#### INFORMACIÓN



## CASA SEGURA



### Errores de comunicación

**La escena:** Lugar de trabajo del equipo de ingeniería de software.

**Participantes:** Jamie Lazar, Vinod Roman y Ed Robins, miembros del equipo de software.

#### La conversación:

**Ed:** ¿Qué has oído sobre el proyecto CasaSegura?

**Vinod:** La reunión de arranque está programada para la semana siguiente.

**Jamie:** Traté de investigar algo, pero no salió bien.

**Ed:** ¿Qué quieres decir?

**Jamie:** Bueno, llamé a Lisa Pérez. Ella es la encargada de mercadotecnia en esto.

**Vinod:** ¿Y...?

**Jamie:** Yo quería que me dijera las características y funciones de CasaSegura... esa clase de cosas. En lugar de ello, comenzó a hacerme preguntas sobre sistemas de seguridad, de vigilancia... No soy experto en eso.

**Vinod:** ¿Qué te dice eso?

(Jamie se encoge de hombros.)

**Vinod:** Será que mercadotecnia quiere que actuemos como consultores y mejor que hagamos alguna tarea sobre esta área de productos antes de nuestra junta de arranque. Doug dijo que quería que "colaboráramos" con nuestro cliente, así que será mejor que aprendamos cómo hacerlo.

**Ed:** Tal vez hubiera sido mejor ir a su oficina. Las llamadas por teléfono simplemente no sirven para esta clase de trabajos.

**Jamie:** Están en lo correcto. Tenemos que actuar juntos o nuestras primeras comunicaciones serán una batalla.

**Vinod:** Yo vi a Doug leyendo un libro acerca de "requerimientos de ingeniería". Apuesto a que enlista algunos principios de buena comunicación. Voy a pedírselo prestado.

**Jamie:** Buena idea... luego nos enseñas.

**Vinod (sonríe):** Sí, de acuerdo.

colaborado bien, todas las partes tendrán un objetivo común. Aun así, la negociación de- mandará el compromiso de todas las partes.

## Principios de planeación

La actividad de comunicación ayuda a definir las metas y objetivos generales (por supuesto, sujetos al cambio conforme pasa el tiempo). Sin embargo, la comprensión de estas metas y objetivos no es lo mismo que definir un plan para lograrlo. La actividad de planeación incluye un conjunto de prácticas administrativas y técnicas que permiten que el equipo de software defina un mapa mientras avanza hacia su meta estratégica y sus objetivos tácticos.

Créalo, es imposible predecir con exactitud cómo se desarrollará un proyecto de software. No existe una forma fácil de determinar qué problemas técnicos se encontrarán, qué información importante permanecerá oculta hasta que el proyecto esté muy avanzado, qué malos entendidos habrá o qué aspectos del negocio cambiarán. No obstante, un buen equipo de software debe planear con este enfoque.

Hay muchas filosofías de planeación.<sup>2</sup> Algunas personas son "minimalistas" y afirman que es frecuente que el cambio elimine la necesidad de hacer un plan detallado. Otras son "tradicionalistas" y dicen que el plan da un mapa eficaz y que entre más detalles tenga menos probable será que el equipo se pierda. Otros más son "agilistas" y plantean que tal vez sea necesario un "juego de planeación" rápido, pero que el mapa surgirá a medida que comience el "trabajo real" con el software.

¿Qué hacer? En muchos proyectos, planear en exceso consume tiempo y es estéril (porque son demasiadas las cosas que cambian), pero planear poco es una receta para el caos. Igual que la mayoría de cosas de la vida, la planeación debe ser tomada con moderación, suficiente para que dé una guía útil al equipo, ni más ni menos. Sin importar el rigor con el que se haga la planeación, siempre se aplican los principios siguientes:

**Cita:**

"Al prepararme para una batalla siempre descubro que los planes son inútiles, pero que la planeación es indispensable."

General Dwight D. Eisenhower

### WebRef

En la dirección [www.4pm.com/repository.htm](http://www.4pm.com/repository.htm), hay excelentes materiales informativos sobre la planeación y administración de proyectos.

**Principio 1. Entender el alcance del proyecto.** Es imposible usar el mapa si no se sabe a dónde se va. El alcance da un destino al equipo de software.

**Principio 2. Involucrar en la actividad de planeación a los participantes del software.** Los participantes definen las prioridades y establecen las restricciones del proyecto. Para incluir estas realidades, es frecuente que los ingenieros de software deban negociar la orden de entrega, los plazos y otros asuntos relacionados con el proyecto.

**Principio 3. Reconocer que la planeación es iterativa.** Un plan para el proyecto nunca está grabado en piedra. Para cuando el trabajo comience, es muy probable que las cosas hayan cambiado. En consecuencia, el plan deberá ajustarse para incluir dichos cambios.

Además, los modelos de proceso iterativo incrementales dictan que debe repetirse la planeación después de la entrega de cada incremento de software, con base en la retroalimentación recibida de los usuarios.

**Principio 4. Estimar con base en lo que se sabe.** El objetivo de la estimación es obtener un índice del esfuerzo, costo y duración de las tareas, con base en la comprensión que tenga el equipo sobre el trabajo que va a realizar. Si la información es vaga o poco confiable, entonces las estimaciones tampoco serán confiables.

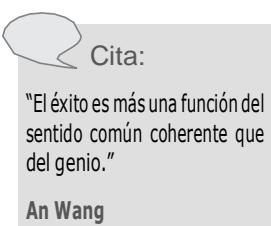
**Principio 5. Al definir el plan, tomar en cuenta los riesgos.** Si ha identificado riesgos que tendrían un efecto grande y es muy probable que ocurran, entonces es necesario elaborar planes de contingencia. Además, el plan del proyecto (incluso la programación de actividades) deberá ajustarse para que incluya la posibilidad de que ocurran uno o más de dichos riesgos.

**Principio 6. Ser realista.** Las personas no trabajan 100% todos los días. En cualquier comunicación humana hay ruido. Las omisiones y ambigüedad son fenómenos de la vida. Los cambios ocurren. Aun los mejores ingenieros de software cometen errores. Éstas y otras realidades deben considerarse al establecer un proyecto.

**Principio 7. Ajustar la granularidad cuando se define el plan.** La granularidad se refiere al nivel de detalle que se adopta cuando se desarrolla un plan. Un plan con “muchas granularidades” proporciona detalles significativos en las tareas para el trabajo que se planea, en incrementos durante un periodo relativamente corto (por lo que el seguimiento y control suceden con frecuencia). Un plan con “poca granularidad” da tareas más amplias para el trabajo que se planea, para plazos más largos. En general, la granularidad va de poca a mucha conforme el tiempo avanza. En las siguientes semanas o meses, el proyecto se planea con detalles significativos. Las actividades que no ocurrirán en muchos meses no requieren mucha granularidad (hay demasiadas cosas que pueden cambiar).

**Principio 8. Definir cómo se trata de asegurar la calidad.** El plan debe identificar la forma en la que el equipo de software busca asegurar la calidad. Si se realizan revisiones técnicas,<sup>3</sup> deben programarse. Si durante la construcción va a utilizarse programación por parejas (véase el capítulo 3), debe definirse en forma explícita en el plan.

**Principio 9. Describir cómo se busca manejar el cambio.** Aun la mejor planeación puede ser anulada por el cambio sin control. Debe identificarse la forma en la que van a reabrirse los cambios a medida que avanza el trabajo de la ingeniería de software. Por ejemplo, ¿el cliente tiene la posibilidad de solicitar un cambio en cualquier momento? Si se solicita uno, ¿está obligado el equipo a implementarlo de inmediato? ¿Cómo se evalúan el efecto y el costo del cambio?



## PONTO CLAVE

El término *granularidad* se refiere al detalle con el que se representan o efectúan algunos elementos de la planeación.



**Principio 10.** *Dar seguimiento al plan con frecuencia y hacer los ajustes que se requieran.* Los proyectos de software se atrasan respecto de su programación. Por tanto, tiene sentido evaluar diariamente el avance, en busca de áreas y situaciones problemáticas en las que las actividades programadas no se apeguen al avance real. Cuando se detecten desviaciones, hay que ajustar el plan en consecuencia.

Para ser más eficaz, cada integrante del equipo de software debe participar en la actividad de planeación. Sólo entonces sus miembros “firmarán” el plan.

### Principios de modelado

Se crean modelos para entender mejor la entidad real que se va a construir. Cuando ésta es física (por ejemplo, un edificio, un avión, una máquina, etc.), se construye un modelo de forma idéntica pero a escala. Sin embargo, cuando la entidad que se va a construir es software, el modelo debe adoptar una forma distinta. Debe ser capaz de representar la información que el software transforma, la arquitectura y las funciones que permiten que esto ocurra, las características que desean los usuarios y el comportamiento del sistema mientras la transformación tiene lugar. Los modelos deben cumplir estos objetivos en diferentes niveles de abstracción, en primer lugar con la ilustración del software desde el punto de vista del cliente y después con su representación en un nivel más técnico.

En el trabajo de ingeniería de software se crean dos clases de modelos: de requerimientos y de diseño. Los *modelos de requerimientos* (también conocidos como *modelos de análisis*) representan los requerimientos del cliente mediante la ilustración del software en tres dominios diferentes: el de la información, el funcional y el de comportamiento. Los *modelos de diseño* representan características del software que ayudan a los profesionales a elaborarlo con eficacia: arquitectura, interfaz de usuario y detalle en el nivel de componente.

En su libro sobre modelado ágil, Scott Ambler y Ron Jeffries [Amb02b] definen un conjunto de principios de modelado<sup>4</sup> dirigidos a todos aquellos que usan el modelo de proceso ágil (véase el capítulo 3), pero que son apropiados para todos los ingenieros de software que efectúan acciones y tareas de modelado:

**Principio 1. El equipo de software tiene como objetivo principal elaborar software, no crear modelos.** Agilidad significa entregar software al cliente de la manera más rápida posible. Los modelos que contribuyan a esto son benéficos, pero deben evitarse aquellos que hagan lento el proceso o que den poca perspectiva.

**Principio 2. Viajar ligero, no crear más modelos de los necesarios.** Todo modelo que se cree debe actualizarse si ocurren cambios. Más importante aún es que todo modelo nuevo exige tiempo, que de otra manera se destinaría a la construcción (codificación y pruebas). Entonces, cree sólo aquellos modelos que hagan más fácil y rápido construir el software.

**Principio 3. Tratar de producir el modelo más sencillo que describa al problema o al software.** No construya software en demasía [Amb02b]. Al mantener sencillos los modelos, el software resultante también lo será. El resultado es que se tendrá un software fácil de integrar, de probar y de mantener (para que cambie). Además, los modelos sencillos son más fáciles de entender y criticar por parte de los miembros del equipo, lo que da como resultado un formato funcional de retroalimentación que optimiza el resultado final.

**Principio 4. Construir modelos susceptibles al cambio.** Suponga que sus modelos cambiarán, pero vigile que esta suposición no lo haga descuidado. Por ejemplo, como los



## CLAVE

Los modelos de requerimientos representan los requerimientos del cliente. Los modelos del diseño dan una especificación concreta para la construcción del software.



El objetivo de cualquier modelo es comunicar información. Para lograr esto, use un formato consistente. Suponga que usted no estará para explicar el modelo. Por eso, el modelo debe describirse por sí solo.

<sup>4</sup> Para fines de este libro, se han abreviado y reescrito los principios mencionados en esta sección.

requerimientos se modificarán, hay una tendencia a ignorar los modelos. ¿Por qué? Porque se sabe que de todos modos cambiarán. El problema con esta actitud es que sin un modelo razonablemente completo de los requerimientos, se creará un diseño (modelo de diseño) que de manera invariable carecerá de funciones y características importantes.

**Principio 5. Ser capaz de enunciar un propósito explícito para cada modelo que se cree.** Cada vez que cree un modelo, pregúntese por qué lo hace. Si no encuentra una razón sólida para la existencia del modelo, no pierda tiempo en él.

**Principio 6. Adaptar los modelos que se desarrollan al sistema en cuestión.** Tal vez sea necesario adaptar a la aplicación la notación del modelo o las reglas; por ejemplo, una aplicación de juego de video quizás requiera una técnica de modelado distinta que el software incrustado que controla el motor de un automóvil en tiempo real.

**Principio 7. Tratar de construir modelos útiles, pero olvidarse de elaborar modelos perfectos.** Cuando un ingeniero de software construye modelos de requerimientos y diseño, alcanza un punto de rendimientos decrecientes. Es decir, el esfuerzo requerido para terminar por completo el modelo y hacerlo internamente consistente deja de beneficiarse por tener dichas propiedades. ¿Se sugiere que el modelado debe ser pobre o de baja calidad? La respuesta es “no”. Pero el modelado debe hacerse con la mirada puesta en las siguientes etapas de la ingeniería de software. Las iteraciones sin fin para obtener un modelo “perfecto” no cumplen la necesidad de agilidad.

**Principio 8. No ser dogmático respecto de la sintaxis del modelo. Si se tiene éxito para comunicar contenido, la representación es secundaria.** Aunque cada miembro del equipo de software debe tratar de usar una notación consistente durante el modelado, la característica más importante del modelo es comunicar información que permita la realización de la siguiente tarea de ingeniería. Si un modelo tiene éxito en hacer esto, es perdurable la sintaxis incorrecta.

**Principio 9. Si su instinto dice que un modelo no es el correcto a pesar de que se vea bien en el papel, hay razones para estar preocupado.** Si usted es un ingeniero de software experimentado, confíe en su instinto. El trabajo de software enseña muchas lecciones, algunas en el nivel del inconsciente. Si algo le dice que un modelo de diseño está destinado a fracasar (aun cuando esto no pueda demostrarse en forma explícita), hay razones para dedicar más tiempo a su estudio o a desarrollar otro distinto.

**Principio 10. Obtener retroalimentación tan pronto como sea posible.** Todo modelo debe ser revisado por los miembros del equipo. El objetivo de estas revisiones es obtener retroalimentación para utilizarla a fin de corregir los errores de modelado, cambiar las interpretaciones equivocadas y agregar las características o funciones omitidas inadvertidamente.

**Requerimientos de los principios de modelado.** En las últimas tres décadas se han desarrollado numerosos métodos de modelado de requerimientos. Los investigadores han identificado los problemas del análisis de requerimientos y sus causas, y han desarrollado varias notaciones de modelado y los conjuntos heurísticos correspondientes para resolver aquéllos. Cada método de análisis tiene un punto de vista único. Sin embargo, todos están relacionados por ciertos principios operacionales:

**Principio 1. Debe representarse y entenderse el dominio de información de un problema.** El dominio de información incluye los datos que fluyen hacia el sistema (usuarios finales, otros sistemas o dispositivos externos), los datos que fluyen fuera del sistema (por la interfaz de usuario, interfaces de red, reportes, gráficas y otros medios) y los almacenamientos de datos que recaban y organizan objetos persistentes de datos (por ejemplo, aquellos que se conservan en forma permanente).





## CLAVE

El modelado del análisis se centra en tres atributos del software: la información que se va a procesar, la función que se va a entregar y el comportamiento que va a suceder.



### Cita:

"En cualquier trabajo de diseño, el primer problema del ingeniero es descubrir cuál es realmente el problema."

Autor desconocido

**Principio 2. Deben definirse las funciones que realizará el software.** Las funciones del software dan un beneficio directo a los usuarios finales y también brindan apoyo interno para las características que son visibles para aquéllos. Algunas funciones transforman los datos que fluyen hacia el sistema. En otros casos, las funciones activan algún nivel de control sobre el procesamiento interno del software o sobre los elementos externos del sistema. Las funciones se describen en muchos y distintos niveles de abstracción, que van de un enunciado de propósito general a la descripción detallada de los elementos del procesamiento que deben invocarse.

**Principio 3. Debe representarse el comportamiento del software (como consecuencia de eventos externos).** El comportamiento del software de computadora está determinado por su interacción con el ambiente externo. Las entradas que dan los usuarios finales, el control de los datos efectuado por un sistema externo o la vigilancia de datos reunidos en una red son el motivo por el que el software se comporta en una forma específica.

**Principio 4. Los modelos que representen información, función y comportamiento deben dividirse de manera que revelen los detalles en forma estratificada (o jerárquica).**

El modelado de los requerimientos es el primer paso para resolver un problema de ingeniería de software. Eso permite entender mejor el problema y proporciona una base para la solución (diseño). Los problemas complejos son difíciles de resolver por completo. Por esta razón, debe usarse la estrategia de *divide y vencerás*. Un problema grande y complejo se divide en subproblemas hasta que cada uno de éstos sea relativamente fácil de entender. Este concepto se llama *partición o separación de entidades*, y es una estrategia clave en el modelado de requerimientos.

**Principio 5. El trabajo de análisis debe avanzar de la información esencial hacia la implementación en detalle.** El modelado de requerimientos comienza con la descripción del problema desde la perspectiva del usuario final. Se describe la "esencia" del problema sin considerar la forma en la que se implementará la solución. Por ejemplo, un juego de video requiere que la jugadora "enseñe" a su protagonista en qué dirección avanzar cuando se mueve hacia un laberinto peligroso. Ésa es la esencia del problema. La implementación detallada (normalmente descrita como parte del modelo del diseño) indica cómo se desarrollará la esencia. Para el juego de video, quizás se use una entrada de voz, o se escriba un comando en un teclado, o tal vez un joystick (o mouse) apunte en una dirección específica, o quizás se mueva en el aire un dispositivo sensible al movimiento.

Con la aplicación de estos principios, un ingeniero de software aborda el problema en forma sistemática. Pero, ¿cómo se aplican estos principios en la práctica? Esta pregunta se responderá en los capítulos 5 a 7.

**Principios del modelado del diseño.** El modelo del diseño del software es análogo a los planos arquitectónicos de una casa. Se comienza por representar la totalidad de lo que se va a construir (por ejemplo, un croquis tridimensional de la casa) que se refina poco a poco para que guíe la construcción de cada detalle (por ejemplo, la distribución de la plomería). De manera similar, el modelo del diseño que se crea para el software da varios puntos de vista distintos del sistema.

No escasean los métodos para obtener los distintos elementos de un diseño de software. Algunos son activados por datos, lo que hace que sea la estructura de éstos la que determine la arquitectura del programa y los componentes de procesamiento resultantes. Otros están motivados por el patrón, y usan información sobre el dominio del problema (el modelo de requerimientos) para desarrollar estilos de arquitectura y patrones de procesamiento. Otros más están orientados a objetos, y utilizan objetos del dominio del problema como impulsores de la creación de estructuras de datos y métodos que los manipulan. No obstante la variedad, todos ellos se apegan a principios de diseño que se aplican sin importar el método empleado.



### Cita:

"Vea primero que el diseño es sabio y justo: eso comprobado, siga resueltamente; no para uno renunciar a rechazar el propósito de que ha resuelto llevara cabo."

William Shakespeare

**Esta guía de estudio, fue vendida por [www.guiaconeved.mx](http://www.guiaconeved.mx) Todos los Derechos Reservados.**

**Principio 1. El diseño debe poderse rastrear hasta el modelo de requerimientos.** El modelo de requerimientos describe el dominio de información del problema, las funciones visibles para el usuario, el comportamiento del sistema y un conjunto de clases de requerimientos que agrupa los objetos del negocio con los métodos que les dan servicio. El modelo de diseño traduce esta información en una arquitectura, un conjunto de subsistemas que implementan las funciones principales y un conjunto de componentes que son la realización de las clases de requerimientos. Los elementos del modelo de diseño deben poder rastrearse en el modelo de requerimientos.

**WebRef**

En la dirección [cs.wwc.edu/~aabyan/Design/](http://cs.wwc.edu/~aabyan/Design/), se encuentran comentarios profundos sobre el proceso de diseño, así como un análisis de la estética del diseño.

**Principio 2. Siempre tomar en cuenta la arquitectura del sistema que se va a construir.** La arquitectura del software (véase el capítulo 9) es el esqueleto del sistema que se va a construir. Afecta interfaces, estructuras de datos, flujo de control y comportamiento del programa, así como la manera en la que se realizarán las pruebas, la susceptibilidad del sistema resultante a recibir mantenimiento y mucho más. Por todas estas razones, el diseño debe comenzar con consideraciones de la arquitectura. Sólo después de establecida ésta deben considerarse los aspectos en el nivel de los componentes.

**Principio 3. El diseño de los datos es tan importante como el de las funciones de procesamiento.** El diseño de los datos es un elemento esencial del diseño de la arquitectura. La forma en la que los objetos de datos se elaboran dentro del diseño no puede dejarse al azar. Un diseño de datos bien estructurado ayuda a simplificar el flujo del programa, hace más fácil el diseño e implementación de componentes de software y más eficiente el procesamiento conjunto.

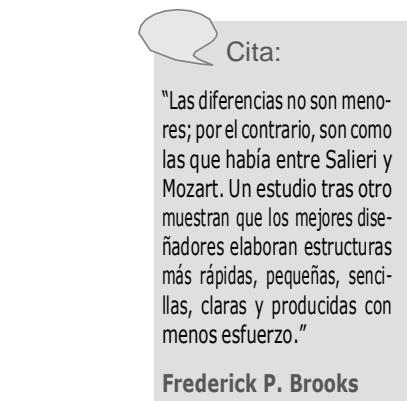
**Principio 4. Las interfaces (tanto internas como externas) deben diseñarse con cuidado.** La manera en la que los datos fluyen entre los componentes de un sistema tiene mucho que ver con la eficiencia del procesamiento, la propagación del error y la simplicidad del diseño. Una interfaz bien diseñada hace que la integración sea más fácil y ayuda a quien la somete a prueba a validar las funciones componentes.

**Principio 5. El diseño de la interfaz de usuario debe ajustarse a las necesidades del usuario final. Sin embargo, en todo caso debe resaltar la facilidad de uso.** La interfaz de usuario es la manifestación visible del software. No importa cuán sofisticadas sean sus funciones internas, ni lo incluyentes que sean sus estructuras de datos, ni lo bien diseñada que esté su arquitectura, un mal diseño de la interfaz con frecuencia conduce a la percepción de que el software es “malo”.

**Principio 6. El diseño en el nivel de componentes debe tener independencia funcional.** La independencia funcional es una medida de la “mentalidad única” de un componente de software. La funcionalidad que entrega un componente debe ser cohesiva, es decir, debe centrarse en una y sólo una función o subfunción.<sup>5</sup>

**Principio 7. Los componentes deben estar acoplados con holgura entre sí y con el ambiente externo.** El acoplamiento se logra de muchos modos: con una interfaz de componente, con mensajería, por medio de datos globales, etc. A medida que se incrementa el nivel de acoplamiento, también aumenta la probabilidad de propagación del error y disminuye la facilidad general de dar mantenimiento al software. Entonces, el acoplamiento de componentes debe mantenerse tan bajo como sea razonable.

**Principio 8. Las representaciones del diseño (modelos) deben entenderse con facilidad.** El propósito del diseño es comunicar información a los profesionales que generarán el código, a los que probarán el software y a otros que le darán mantenimiento en el futuro. Si el diseño es difícil de entender, no servirá como medio de comunicación eficaz.



**Principio 9.** *El diseño debe desarrollarse en forma iterativa. El diseñador debe buscar más sencillez en cada iteración.* Igual que ocurre con casi todas las actividades creativas, el diseño ocurre de manera iterativa. Las primeras iteraciones sirven para mejorar el diseño y corregir errores, pero las posteriores deben buscar un diseño tan sencillo como sea posible.

Cuando se aplican en forma apropiada estos principios de diseño, se crea uno que exhibe factores de calidad tanto externos como internos [Mye78]. Los *factores de calidad externos* son aquellas propiedades del software fácilmente observables por los usuarios (por ejemplo, velocidad, confiabilidad, corrección y usabilidad). Los *factores de calidad internos* son de importancia para los ingenieros de software. Conducen a un diseño de alta calidad desde el punto de vista técnico. Para obtener factores de calidad internos, el diseñador debe entender los conceptos básicos del diseño (véase el capítulo 8).

### Principios de construcción



Cita:

"Durante gran parte de mi vida he sido un mirón del software, y observo furtivamente el código sucio de otras personas. A veces encuentro una verdadera joya, un programa bien estructurado escrito en un estilo consistente, libre de errores, desarrollado de modo que cada componente es sencillo y organizado, y que está diseñado de modo que el producto es fácil de cambiar."

David Parnas

La actividad de construcción incluye un conjunto de tareas de codificación y pruebas que lleva a un software operativo listo para entregarse al cliente o usuario final. En el trabajo de ingeniería de software moderna, la codificación puede ser 1) la creación directa de lenguaje de programación en código fuente (por ejemplo, Java), 2) la generación automática de código fuente que usa una representación intermedia parecida al diseño del componente que se va a construir o 3) la generación automática de código ejecutable que utiliza un "lenguaje de programación de cuarta generación" (por ejemplo, Visual C++).

Las pruebas dirigen su atención inicial al componente, y con frecuencia se denomina *prueba unitaria*. Otros niveles de pruebas incluyen 1) *de integración* (realizadas mientras el sistema está en construcción), 2) *de validación*, que evalúan si los requerimientos se han satisfecho para todo el sistema (o incremento de software) y 3) *de aceptación*, que efectúa el cliente en un esfuerzo por utilizar todas las características y funciones requeridas. Los siguientes principios y conceptos son aplicables a la codificación y prueba:

**Principios de codificación.** Los principios que guían el trabajo de codificación se relacionan de cerca con el estilo, lenguajes y métodos de programación. Sin embargo, puede enunciarse cierto número de principios fundamentales:

**Principios de preparación:** *Antes de escribir una sola línea de código, asegúrese de:*

- Entender el problema que se trata de resolver.
- Comprender los principios y conceptos básicos del diseño.
- Elegir un lenguaje de programación que satisfaga las necesidades del software que se va a elaborar y el ambiente en el que operará.
- Seleccionar un ambiente de programación que disponga de herramientas que hagan más fácil su trabajo.
- Crear un conjunto de pruebas unitarias que se aplicarán una vez que se haya terminado el componente a codificar.



CONSEJO  
Evite desarrollar un programa elegante que resuelva el problema equivocado. Ponga especial atención al primer principio de preparación.

**Principios de programación:** *Cuando comience a escribir código, asegúrese de:*

- Restringir sus algoritmos por medio del uso de programación estructurada [Boh00].
- Tomar en consideración el uso de programación por parejas.
- Seleccionar estructuras de datos que satisfagan las necesidades del diseño.
- Entender la arquitectura del software y crear interfaces que son congruentes con ella.
- Mantener la lógica condicional tan sencilla como sea posible.

- Crear lazos anidados en forma tal que se puedan probar con facilidad.
- Seleccionar nombres significativos para las variables y seguir otros estándares locales de codificación.
- Escribir código que se documente a sí mismo.
- Crear una imagen visual (por ejemplo, líneas con sangría y en blanco) que ayude a entender.

**Principios de validación:** *Una vez que haya terminado su primer intento de codificación, asegúrese de:*

- Realizar el recorrido del código cuando sea apropiado.
- Llevar a cabo pruebas unitarias y corregir los errores que se detecten.
- Rediseñar el código.

#### WebRef

En la dirección [www.literateprogramming.com/festyle.html](http://www.literateprogramming.com/festyle.html), hay una amplia variedad de vínculos a estándares de codificación.

Se han escrito más libros sobre programación (codificación) y sobre los principios y conceptos que la guían que sobre cualquier otro tema del proceso de software. Los libros sobre el tema incluyen obras tempranas sobre estilo de programación [Ker78], construcción de software práctico [McC04], perlas de programación [Ben99], el arte de programar [Knu98], temas pragmáticos de programación [Hun99] y muchísimos temas más. El análisis exhaustivo de estos principios y conceptos está más allá del alcance de este libro. Si tiene interés en profundizar, estudie una o varias de las referencias que se mencionan.

**Principios de la prueba.** En un libro clásico sobre las pruebas de software, Glen Myers [Mye79] enumera algunas reglas que sirven bien como objetivos de prueba:

- La prueba es el proceso que ejecuta un programa con objeto de encontrar un error.
- Un buen caso de prueba es el que tiene alta probabilidad de encontrar un error que no se ha detectado hasta el momento.
- Una prueba exitosa es la que descubre un error no detectado hasta el momento.

Estos objetivos implican un cambio muy grande en el punto de vista de ciertos desarrolladores de software. Ellos avanzan contra la opinión común de que una prueba exitosa es aquella que no encuentra errores en el software. El objetivo es diseñar pruebas que detecten de manera sistemática diferentes clases de errores, y hacerlo con el mínimo tiempo y esfuerzo.

Si las pruebas se efectúan con éxito (de acuerdo con los objetivos ya mencionados), descubrirán errores en el software. Como beneficio secundario, la prueba demuestra que las funciones de software parecen funcionar de acuerdo con las especificaciones, y que los requerimientos de comportamiento y desempeño aparentemente se cumplen. Además, los datos obtenidos conforme se realiza la prueba dan una buena indicación de la confiabilidad del software y ciertas indicaciones de la calidad de éste como un todo. Pero las pruebas no pueden demostrar la inexistencia de errores y defectos; sólo demuestran que hay errores y defectos. Es importante recordar esto (que de otro modo parecería muy pesimista) cuando se efectúe una prueba.

Davis [Dav95b] sugiere algunos principios para las pruebas,<sup>6</sup> que se han adaptado para usarlos en este libro:

**Principio 1. Todas las pruebas deben poder rastrearse hasta los requerimientos del cliente.**<sup>7</sup> El objetivo de las pruebas de software es descubrir errores. Entonces, los defec-

6 Aquí sólo se mencionan pocos de los principios de prueba de Davis. Para más información, consulte [Dav95b].

7 Este principio se refiere a las *pruebas funcionales*, por ejemplo, aquellas que se centran en los requerimientos. Las *pruebas estructurales* (las que se centran en los detalles de arquitectura o lógica) tal vez no aborden directamente los



tos más severos (desde el punto de vista del cliente) son aquellos que hacen que el programa no cumpla sus requerimientos.

**Principio 2. Las pruebas deben planearse mucho antes de que den comienzo.** La planeación de las pruebas (véase el capítulo 17) comienza tan pronto como se termina el modelo de requerimientos. La definición detallada de casos de prueba principia apenas se ha concluido el modelo de diseño. Por tanto, todas las pruebas pueden planearse y diseñarse antes de generar cualquier código.

**Principio 3. El principio de Pareto se aplica a las pruebas de software.** En este contexto, el principio de Pareto implica que 80% de todos los errores no detectados durante las pruebas se relacionan con 20% de todos los componentes de programas. Por supuesto, el problema es aislar los componentes sospechosos y probarlos a fondo.

**Principio 4. Las pruebas deben comenzar “en lo pequeño” y avanzar hacia “lo grande”.** Las primeras pruebas planeadas y ejecutadas por lo general se centran en componentes individuales. Conforme avanzan las pruebas, la atención cambia en un intento por encontrar errores en grupos integrados de componentes y, en última instancia, en todo el sistema.

**Principio 5. No son posibles las pruebas exhaustivas.** Hasta para un programa de tamaño moderado, el número de permutaciones de las rutas es demasiado grande. Por esta razón, durante una prueba es imposible ejecutar todas las combinaciones de rutas. Sin embargo, es posible cubrir en forma adecuada la lógica del programa y asegurar que se han probado todas las condiciones en el nivel de componentes.

## Principios de despliegue

Como se dijo en la parte 1 del libro, la actividad del despliegue incluye tres acciones: entrega, apoyo y retroalimentación. Como la naturaleza de los modelos del proceso del software moderno es evolutiva o incremental, el despliegue ocurre no una vez sino varias, a medida que el software avanza hacia su conclusión. Cada ciclo de entrega pone a disposición de los clientes y usuarios finales un incremento de software operativo que brinda funciones y características utilizables. Cada ciclo de apoyo provee documentación y ayuda humana para todas las funciones y características introducidas durante los ciclos de despliegue realizados hasta ese momento. Cada ciclo de retroalimentación da al equipo de software una guía importante que da como resultado modificaciones de las funciones, de las características y del enfoque adoptado para el siguiente incremento.

La entrega de un incremento de software representa un punto de referencia importante para cualquier proyecto de software. Cuando el equipo se prepara para entregar un incremento, deben seguirse ciertos principios clave:



Asegúrese de que su cliente sabe lo que puede esperar antes de que se entregue un incremento de software. De otra manera, puede apostar a que el cliente espera más de lo que usted le dará.

**Principio 1. Deben manejarse las expectativas de los clientes.** Con demasiada frecuencia, el cliente espera más de lo que el equipo ha prometido entregar, y la desilusión llega de inmediato. Esto da como resultado que la retroalimentación no sea productiva y arruine la moral del equipo. En su libro sobre la administración de las expectativas, Naomi Kartan [Kar94] afirma que “el punto de inicio de la administración de las expectativas es ser más consciente de lo que se comunica y de la forma en la que esto se hace”. Ella sugiere que el ingeniero de software debe tener cuidado con el envío de mensajes conflictivos al cliente (por ejemplo, prometer más de lo que puede entregarse de manera razonable en el plazo previsto, o entregar más de lo que se prometió en un incremento de software y para el siguiente entregar menos).

**Principio 2. Debe ensamblarse y probarse el paquete completo que se entregará.** Debe ensamblarse en un CD-ROM u otro medio (incluso descargas desde web) todo el software ejecutable, archivos de datos de apoyo, documentos de ayuda y otra información relevante.

vante, para después hacer una prueba beta exhaustiva con usuarios reales. Todos los *scripts* de instalación y otras características de operación deben ejecutarse por completo en tantas configuraciones diferentes de cómputo como sea posible (por ejemplo, hardware, sistemas operativos, equipos periféricos, configuraciones de red, etcétera).

**Principio 3. Antes de entregar el software, debe establecerse un régimen de apoyo.** Un usuario final espera respuesta e información exacta cuando surja una pregunta o problema. Si el apoyo es *ad hoc*, o, peor aún, no existe, el cliente quedará insatisfecho de inmediato. El apoyo debe planearse, los materiales respectivos deben prepararse y los mismos apropiados de registro deben establecerse a fin de que el equipo de software realice una evaluación categórica de las clases de apoyo solicitado.

**Principio 4. Se deben proporcionar a los usuarios finales materiales de aprendizaje apropiados.** El equipo de software entrega algo más que el software en sí. Deben desarrollarse materiales de capacitación apropiados (si se requirieran); es necesario proveer lineamientos para solución de problemas y, cuando sea necesario, debe publicarse “lo que es diferente en este incremento de software”.<sup>8</sup>

**Principio 5. El software defectuoso debe corregirse primero y después entregarse.**

Cuando el tiempo apremia, algunas organizaciones de software entregan incrementos de baja calidad con la advertencia de que los errores “se corregirán en la siguiente entrega”. Esto es un error. Hay un adagio en el negocio del software que dice así: “Los clientes olvidarán pronto que entregaste un producto de alta calidad, pero nunca olvidarán los problemas que les causó un producto de mala calidad. El software se los recuerda cada día.”

El software entregado brinda beneficios al usuario final, pero también da retroalimentación útil para el equipo que lo desarrolló. Cuando el incremento se libere, debe invitarse a los usuarios finales a que comenten acerca de características y funciones, facilidad de uso, confiabilidad y cualesquier otras características.

#### 4.4 RESUMEN

La práctica de la ingeniería de software incluye principios, conceptos, métodos y herramientas que los ingenieros de software aplican en todo el proceso de desarrollo. Todo proyecto de ingeniería de software es diferente. No obstante, existe un conjunto de principios generales que se aplican al proceso como un todo y a cada actividad estructural, sin importar cuál sea el proyecto o el producto.

Existe un conjunto de principios fundamentales que ayudan en la aplicación de un proceso de software significativo y en la ejecución de métodos de ingeniería de software eficaz. En el nivel del proceso, los principios fundamentales establecen un fundamento filosófico que guía al equipo de software cuando avanza por el proceso del software. En el nivel de la práctica, los principios fundamentales establecen un conjunto de valores y reglas que sirven como guía al analizar el diseño de un problema y su solución, al implementar ésta y al someterla a prueba para, finalmente, desplegar el software en la comunidad del usuario.

Los principios de comunicación se centran en la necesidad de reducir el ruido y mejorar el ancho de banda durante la conversación entre el desarrollador y el cliente. Ambas partes deben colaborar a fin de lograr la mejor comunicación.

Los principios de planeación establecen lineamientos para elaborar el mejor mapa del proceso hacia un sistema o producto terminado. El plan puede diseñarse sólo para un incremento

<sup>8</sup> Durante la actividad de comunicación, el equipo de software debe determinar los tipos de materiales de ayuda que quiere el usuario.



CENTRO NACIONAL  
DE EVALUACIÓN PARA  
LA EDUCACIÓN SUPERIOR, A.C.



**EDITORIAL:**  
**guiaceneval.mx**



del software, o para todo el proyecto. Sin que esto importe, debe definir lo que se hará, quién lo hará y cuándo se terminará el trabajo.

El modelado incluye tanto el análisis como el diseño, y describe representaciones cada vez más detalladas del software. El objetivo de los modelos es afirmar el entendimiento del trabajo que se va a hacer y dar una guía técnica a quienes implementarán el software. Los principios de modelado dan fundamento a los métodos y notación que se utilizan para crear representaciones del software.

La construcción incorpora un ciclo de codificación y pruebas en el que se genera código fuente para cierto componente y es sometido a pruebas. Los principios de codificación definen las acciones generales que deben tener lugar antes de que se escriba el código, mientras se escribe y una vez terminado. Aunque hay muchos principios para las pruebas, sólo uno predomina: la prueba es el proceso que lleva a ejecutar un programa con objeto de encontrar un error.

El despliegue ocurre cuando se presenta al cliente un incremento de software, e incluye la entrega, apoyo y retroalimentación. Los principios clave para la entrega consideran la administración de las expectativas del cliente y darle información de apoyo adecuada sobre el software. El apoyo demanda preparación anticipada. La retroalimentación permite al cliente sugerir cambios que tengan valor para el negocio y que brinden al desarrollador información para el ciclo iterativo siguiente de ingeniería de software.

## PROBLEMAS Y PUNTOS POR EVALUAR

Toda vez que la búsqueda de la calidad reclama recursos y tiempo, ¿es posible ser ágil y centrarse en ella?

De los ocho principios fundamentales que guían el proceso (lo que se estudió en la sección 4.2.1), ¿cuál cree que sea el más importante?

Describa con sus propias palabras el concepto de *separación de entidades*.

Un principio de comunicación importante establece que hay que “prepararse antes de comunicarse”.

¿Cómo debe manifestarse esta preparación en los primeros trabajos que se hacen? ¿Qué productos del trabajo son resultado de la preparación temprana?

Haga algunas investigaciones acerca de cómo “facilitar” la actividad de comunicación (use las referencias que se dan u otras distintas) y prepare algunos lineamientos que se centren en la facilitación.

¿En qué difiere la comunicación ágil de la comunicación tradicional de la ingeniería de software? ¿En qué se parecen?

¿Por qué es necesario “avanzar”?

Investigue sobre la “negociación” para la actividad de comunicación y prepare algunos lineamientos que se centren sólo en ella.

Describa lo que significa *granularidad* en el contexto de la programación de actividades de un proyecto.

¿Por qué son importantes los modelos en el trabajo de ingeniería de software? ¿Siempre son necesarios? ¿Hay calificadores para la respuesta que se dio sobre esta necesidad?

¿Cuáles son los tres “dominios” considerados durante el modelado de requerimientos?

Trate de agregar un principio adicional a los que se mencionan en la sección 4.3.4 para la codificación.

¿Qué es una prueba exitosa?

Diga si está de acuerdo o en desacuerdo con el enunciado siguiente: “Como entregamos incrementos múltiples al cliente, no debiéramos preocuparnos por la calidad en los primeros incrementos; en las iteraciones posteriores podemos corregir los problemas. Explique su respuesta.

¿Por qué es importante la retroalimentación para el equipo de software?

## LECTURAS Y FUENTES DE INFORMACIÓN ADICIONALES

La comunicación con el cliente es una actividad de importancia crítica en la ingeniería de software, pero pocos de sus practicantes dedican tiempo a leer sobre ella. Withall (*Software Requirements Patterns*, Microsoft Press, 2007) presenta varios patrones útiles que analizan problemas en la comunicación. Sutliff (*User-Centred Requirements Engineering*, Springer, 2002) se centra mucho en los retos relacionados con la comunicación. Los libros de Weigert (*Software Requirements*, 2a. ed., Microsoft Press, 2003), Pardee (*To Satisfy and Delight Your Customer*, Dorset House, 1996) y Karten [Kar94] analizan a profundidad los métodos para tener una interacción eficaz con el cliente. Aunque su libro no se centra en el software, Hooks y Farry (*Customer Centred Products*, American Management Association, 2000) presentan lineamientos generales útiles para la comunicación con los clientes. Young (*Effective Requirements Practices*, Addison-Wesley, 2001) pone el énfasis en un "equipo conjunto" de clientes y desarrolladores que recaban los requerimientos en colaboración. Somerville y Kotonya (*Requirements Engineering: Processes and Techniques*, Wiley, 1998) analizan el concepto de "provocación" y las técnicas y otros requerimientos de los principios de ingeniería.

Los conceptos y principios de la comunicación y planeación son estudiados en muchos libros de administración de proyectos. Entre los más útiles se encuentran los de Bechtold (*Essentials of Software Project Management*, 2a. ed., Management Concepts, 2007), Wysocki (*Effective Project Management: Traditional, Adaptive, Extreme*, 4a. ed., Wiley, 2006), Leach (*Lean Project Management: Eight Principles for Success*, BookSurge Publishing, 2006) Hughes (*Software Project Management*, McGraw-Hill, 2005) y Stellman y Greene (*Applied Software Project Management*, O'Reilly Media, Inc., 2005).

Davis [Dav95] hizo una compilación excelente de referencias sobre principios de la ingeniería de software. Además, virtualmente todo libro al respecto contiene un análisis útil de los conceptos y principios para análisis, diseño y prueba. Entre los más utilizados (además de éste, claro) se encuentran los siguientes:

- Abran, A., y J. Moore, *SWEBOk: Guide to the Software Engineering Body of Knowledge*, IEEE, 2002. Christensen, M., y R. nayer, *A Project Manager's Guide to Software Engineering Best Practices*, IEEE-CS Press (Wiley), 2002.
- Jalote, P., *An Integrated Approach to Software Engineering*, Springer, 2006.
- Pfleeger, S., *Software Engineering: Theory and Practice*, 3a. ed., Prentice-Hall, 2005. Schach, S., *Object-Oriented and Classical Software Engineering*, McGraw-Hill, 7a. ed., 2006.
- Sommerville, I., *Software Engineering*, 8a. ed., Addison-Wesley, 2006

Estos libros también presentan análisis detallados sobre los principios de modelado y construcción.

Los principios de modelado se estudian en muchos libros dedicados al análisis de requerimientos o diseño de software. Los libros de Lieberman (*The Art of Software Modeling*, Auerbach, 2007), Rosenberg y Stephens (*Use Case Driven Object Modeling with UML: Theory and Practice*, Apress, 2007), Roques (*UML in Practice*, Wiley, 2004) y Penker y Eriksson (*Business Modeling with UML: Business Patterns at Work*, Wiley, 2001) analizan los principios y métodos de modelado.

Todo ingeniero de software que trate de hacer diseño está obligado a leer el texto de Norman (*The Design of Everyday Things*, Currency/Doubleday, 1990). Winograd y sus colegas (*Bringing Design to Software*, Addison-Wesley, 1996) editaron una excelente colección de ensayos sobre aspectos prácticos del diseño de software. Constantine y Lockwood (*Software for Use*, Addison-Wesley, 1999) presenta los conceptos asociados con el "diseño centrado en el usuario". Tognazzini (*Tog on Software Design*, Addison-Wesley, 1995) presenta una reflexión filosófica útil sobre la naturaleza del diseño y el efecto que tienen las decisiones sobre la calidad y la capacidad del equipo para producir software que agregue mucho valor para su cliente. Stahl y sus colegas (*Model-Driven Software Development: Technology, Engineering*, Wiley, 2006) estudian los principios del desarrollo determinado por el modelo.

Son cientos los libros que abordan uno o más elementos de la actividad de construcción. Kernighan y Plauger [Ker78] escribieron un texto clásico sobre el estilo de programación, McConnell [McC93] presenta lineamientos prácticos para la construcción de software, Bentley [Ben99] sugiere una amplia variedad de perlas de la programación, Knuth [Knu99] escribió una serie clásica de tres volúmenes acerca del arte de programar y Hunt [Hun99] sugiere lineamientos pragmáticos para la programación.

Myers y sus colegas (*The Art of Software Testing*, 2a. ed., Wiley, 2004) desarrollaron una revisión importante de su texto clásico y muchos principios importantes para la realización de pruebas. Los libros de Perry (*Effective Methods for Software Testing*, 3a. ed., Wiley 2006), Whittaker (*How to Break Software*, Addison-Wesley, 2002), Kaner y sus colegas (*Lessons Learned in Software Testing*, Wiley, 2001) y Marick (*The Craft of Software Testing*, 2a. ed., Wiley, 2004) presentan una amplia variedad de perlas de la programación.



Prentice-Hall, 1997) presentan por separado conceptos y principios importantes para hacer pruebas, así como muchas guías prácticas.

En internet existe una amplia variedad de fuentes de información sobre la práctica de ingeniería de software. En el sitio web del libro se encuentra una lista actualizada de referencias en la Red Mundial que son relevantes para la ingeniería de software: [www.mhe.com/engcs/compsciprofessional/olc/ser.htm](http://www.mhe.com/engcs/compsciprofessional/olc/ser.htm)



## **Lectura 2. Comprensión de los requerimientos**



# COMPRENSIÓN DE LOS REQUERIMIENTOS

C ONCEPTOS CLAVE	
administración de los requerimientos .....	105
casos de uso .....	113
colaboración .....	107
concepción .....	102
despliegue de la función de calidad .....	111
elaboración .....	117
especificación .....	104
indagación .....	103
indagación de los requerimientos .....	108
ingeniería de requerimientos .....	102
modelo del análisis .....	117
negociación .....	121
participantes .....	106
patrones de análisis .....	120
productos del trabajo .....	112
puntos de vista .....	107
validación .....	105
validación de los requerimientos .....	122

U  
NA  
MIRADA  
RÁPIDA

**¿Qué es?** Antes de comenzar cualquier trabajo técnico es una buena idea aplicar un conjunto de tareas de ingeniería a los requerimientos. Éstas llevarán a la comprensión de

cómo será el efecto que tendrá el software en el negocio, qué es lo que quiere el cliente y cómo interactuarán los usuarios finales con el software.

**¿Quién lo hace?** Los ingenieros de software (que en el mundo de las tecnologías de información a veces son llamados *ingenieros de sistemas o analistas*) y todos los demás participantes del proyecto (gerentes, clientes y usuarios) intervienen en la ingeniería de requerimientos.

**¿Por qué es importante?** Diseñar y construir un elegante programa de cómputo que resuelva el problema equivocado no satisface las necesidades de nadie. Por eso es importante entender lo que el cliente desea antes de comenzar a diseñar y a construir un sistema basado en computadora.

a definir lo que se requiere. Después sigue la elaboración, donde se refinan y modifican los requerimientos básicos. Cuando los participantes definen el problema, tiene lugar una negociación: ¿cuáles son las prioridades, qué es lo

**¿Cuáles son los pasos?** La ingeniería de requerimientos comienza con la concepción, tarea que define el alcance y la naturaleza del problema que se va a resolver. Va seguida de la indagación, labor que ayuda a los participantes

esencial, cuándo se requiere? Por último, se especifica el problema de algún modo y luego se revisa o valida para garantizar que hay coincidencia entre la comprensión que usted tiene del problema y la que tienen los participantes.

**¿Cuál es el producto final?** El objetivo de los requerimientos de ingeniería es proporcionar a todas las partes un entendimiento escrito del problema. Esto se logra por medio de varios productos del trabajo: escenarios de uso, listas de

funciones y de características, modelos de requerimientos o especificaciones.

**¿Cómo me aseguro de que lo hice bien?** Se revisan con los participantes los productos del trabajo de la ingeniería de requerimientos a fin de asegurar que lo que se aprendió es lo que ellos quieren decir en realidad. Aquí cabe una advertencia: las cosas cambiarán aun después de que todas las partes estén de acuerdo, y seguirán cambiando durante todo el proyecto.



Es razonable afirmar que las técnicas que se estudiarán en este capítulo no son una “solución” verdadera para los retos que se mencionaron, pero sí proveen de un enfoque sólido para enfrentarlos.

## 5.1 INGENIERÍA DE REQUERIMIENTOS

### Cita:

“La parte más difícil al construir un sistema de software es decidir qué construir. Ninguna parte del trabajo invalida tanto al sistema resultante si ésta se hace mal. Nada es más difícil de corregir después.”

Fred Brooks

El diseño y construcción de software de computadora es difícil, creativo y sencillamente divertido. En realidad, elaborar software es tan atractivo que muchos desarrolladores de software quieren ir directo a él antes de haber tenido el entendimiento claro de lo que se necesita. Argumentan que las cosas se aclararán a medida que lo elaboren, que los participantes en el proyecto podrán comprender sus necesidades sólo después de estudiar las primeras iteraciones del software, que las cosas cambian tan rápido que cualquier intento de entender los requerimientos en detalle es una pérdida de tiempo, que las utilidades salen de la producción de un programa que funcione y que todo lo demás es secundario. Lo que hace que estos argumentos sean tan seductores es que tienen algunos elementos de verdad.<sup>1</sup> Pero todos son erróneos y pueden llevar un proyecto de software al fracaso.

El espectro amplio de tareas y técnicas que llevan a entender los requerimientos se denomina *ingeniería de requerimientos*. Desde la perspectiva del proceso del software, la ingeniería de requerimientos es una de las acciones importantes de la ingeniería de software que comienza durante la actividad de comunicación y continúa en la de modelado. Debe adaptarse a las necesidades del proceso, del proyecto, del producto y de las personas que hacen el trabajo.

La ingeniería de requerimientos tiende un puente para el diseño y la construcción. Pero, ¿dónde se origina el puente? Podría argumentarse que principia en los pies de los participantes en el proyecto (por ejemplo, gerentes, clientes y usuarios), donde se definen las necesidades del negocio, se describen los escenarios de uso, se delinean las funciones y características y se identifican las restricciones del proyecto. Otros tal vez sugieran que empieza con una definición más amplia del sistema, donde el software no es más que un componente del dominio del sistema mayor. Pero sin importar el punto de arranque, el recorrido por el puente lo lleva a uno muy alto sobre el proyecto, lo que le permite examinar el contexto del trabajo de software que debe realizarse; las necesidades específicas que deben abordar el diseño y la construcción; las prioridades que guían el orden en el que se efectúa el trabajo, y la información, las funciones y los comportamientos que tendrán un profundo efecto en el diseño resultante.

La ingeniería de requerimientos proporciona el mecanismo apropiado para entender lo que desea el cliente, analizar las necesidades, evaluar la factibilidad, negociar una solución razonable, especificar la solución sin ambigüedades, validar la especificación y administrar los requerimientos a medida de que se transforman en un sistema funcional [na97]. Incluye siete tareas diferentes: concepción, indagación, elaboración, negociación, especificación, validación y administración. Es importante notar que algunas de estas tareas ocurren en paralelo y que todas se adaptan a las necesidades del proyecto.

**Concepción.** ¿Cómo inicia un proyecto de software? ¿Existe un solo evento que se convierte en el catalizador de un nuevo sistema o producto basado en computadora o la necesidad evoluciona en el tiempo? No hay respuestas definitivas a estas preguntas. En ciertos casos, una conversación casual es todo lo que se necesita para desencadenar un trabajo grande de ingeniería de software. Pero en general, la mayor parte de proyectos comienzan cuando se identifica una necesidad del negocio o se descubre un nuevo mercado o servicio potencial. Los parti-

<sup>1</sup> Esto es cierto en particular para los proyectos pequeños (menos de un mes) y muy pequeños, que requieren relativamente poco esfuerzo de software sencillo. A medida que el software crece en tamaño y complejidad, estos argumentos comienzan a ser falsos.



## CLAVE

La ingeniería de requerimientos establece una base sólida para el diseño y la construcción. Sin ésta, el software resultante tiene alta probabilidad de no satisfacer las necesidades del cliente.



Espere hacer un poco de diseño al recabar los requerimientos, y un poco de requerimientos durante el trabajo de diseño.

### Cita:

“Las semillas de los desastres enormes del software por lo general se vislumbran en los tres primeros meses del inicio del proyecto.”

Coper Jones

pantes de la comunidad del negocio (por ejemplo, los directivos, personal de mercadotecnia, gerentes de producto, etc.) definen un caso de negocios para la idea, tratan de identificar el ritmo y profundidad del mercado, hacen un análisis de gran visión de la factibilidad e identifican una descripción funcional del alcance del proyecto. Toda esta información está sujeta a cambio, pero es suficiente para desencadenar análisis con la organización de ingeniería de software.<sup>2</sup>

En la concepción del proyecto,<sup>3</sup> se establece el entendimiento básico del problema, las personas que quieren una solución, la naturaleza de la solución que se desea, así como la eficacia de la comunicación y colaboración preliminares entre los otros participantes y el equipo de software.

**Indagación.** En verdad que parece muy simple: preguntar al cliente, a los usuarios y a otras personas cuáles son los objetivos para el sistema o producto, qué es lo que va a lograrse, cómo se ajusta el sistema o producto a las necesidades del negocio y, finalmente, cómo va a usarse el sistema o producto en las operaciones cotidianas. Pero no es simple: es muy difícil.

Christel y Kang [Cri92] identificaron cierto número de problemas que se encuentran cuando ocurre la indagación:



¿Por qué es difícil llegar al entendimiento claro de lo que quiere el cliente?

- **Problemas de alcance.** La frontera de los sistemas está mal definida o los clientes o usuarios finales especifican detalles técnicos innecesarios que confunden, más que clarifican, los objetivos generales del sistema.
- **Problemas de entendimiento.** Los clientes o usuarios no están completamente seguros de lo que se necesita, comprenden mal las capacidades y limitaciones de su ambiente de computación, no entienden todo el dominio del problema, tienen problemas para comunicar las necesidades al ingeniero de sistemas, omiten información que creen que es “obvia”, especifican requerimientos que están en conflicto con las necesidades de otros clientes o usuarios, o solicitan requerimientos ambiguos o que no pueden someterse a prueba.
- **Problemas de volatilidad.** Los requerimientos cambian con el tiempo.

Para superar estos problemas, debe enfocarse la obtención de requerimientos en forma organizada.

**Elaboración.** La información obtenida del cliente durante la concepción e indagación se expande y refina durante la elaboración. Esta tarea se centra en desarrollar un modelo refinado de los requerimientos (véanse los capítulos 6 y 7) que identifique distintos aspectos de la función del software, su comportamiento e información.

La elaboración está motivada por la creación y mejora de escenarios de usuario que describen cómo interactuará el usuario final (y otros actores) con el sistema. Cada escenario de usuario se enumera con sintaxis apropiada para extraer clases de análisis, que son entidades del dominio del negocio visibles para el usuario final. Se definen los atributos de cada clase de análisis y se identifican los servicios<sup>4</sup> que requiere cada una de ellas. Se identifican las relaciones y colaboración entre clases, y se producen varios diagramas adicionales.

**Negociación.** No es raro que los clientes y usuarios pidan más de lo que puede lograrse dado lo limitado de los recursos del negocio. También es relativamente común que distintos clientes

- 
- 2 Si va a desarrollarse un sistema basado en computadora, los análisis comienzan en el contexto de un proceso de ingeniería de sistemas. Para más detalles de la ingeniería de sistemas, visite el sitio web de esta obra.
  - 3 Recuerde que el proceso unificado (véase el capítulo 2) define una “fase de concepción” más amplia que incluye las fases de concepción, indagación y elaboración, que son estudiadas en dicho capítulo.
  - 4 Un servicio manipula los datos agrupados por clase. También se utilizan los términos *operación* y *método*. Si no está familiarizado con conceptos de la orientación a objetos, consulte el apéndice 2, en el que se presenta una introducción básica.





*En una negociación eficaz no debe haber ganador ni perdedor. Ambos lados ganan porque un "trato" con el que ambas partes pueden vivir algo sólido.*



La formalidad y el formato de una especificación varían con el tamaño y complejidad del software que se va a construir.

o usuarios propongan requerimientos conflictivos con el argumento de que su versión es “esencial para nuestras necesidades especiales”.

Estos conflictos deben reconciliarse por medio de un proceso de negociación. Se pide a clientes, usuarios y otros participantes que ordenen sus requerimientos según su prioridad y que después analicen los conflictos. Con el empleo de un enfoque iterativo que da prioridad a los requerimientos, se evalúa su costo y riesgo, y se enfrentan los conflictos internos; algunos requerimientos se eliminan, se combinan o se modifican de modo que cada parte logre cierto grado de satisfacción.

**Especificación.** En el contexto de los sistemas basados en computadora (y software), el término *especificación* tiene diferentes significados para distintas personas. Una especificación puede ser un documento escrito, un conjunto de modelos gráficos, un modelo matemático formal, un conjunto de escenarios de uso, un prototipo o cualquier combinación de éstos.

Algunos sugieren que para una especificación debe desarrollarse y utilizarse una “plantilla estándar” [Som97], con el argumento de que esto conduce a requerimientos presentados en forma consistente y por ello más comprensible. Sin embargo, en ocasiones es necesario ser flexible cuando se desarrolla una especificación. Para sistemas grandes, el mejor enfoque puede ser un documento escrito que combine descripciones en un lenguaje natural con modelos gráficos. No obstante, para productos o sistemas pequeños que residan en ambientes bien entendidos, quizás todo lo que se requiera sea escenarios de uso.

## INFORMACIÓN



### Formato de especificación de requerimientos de software

Una *especificación de requerimientos de software* (ERS) es un documento que se crea cuando debe especificarse una

descripción detallada de todos los aspectos del software que se va a elaborar, antes de que el proyecto comience. Es importante notar que una ERS formal no siempre está en forma escrita. En realidad, hay muchas circunstancias en las que el esfuerzo dedicado a la ERS estaría mejor aprovechado en otras actividades de la ingeniería de software. Sin embargo, se justifica la ERS cuando el software va a ser desarrollado por una tercera parte, cuando la falta de una especificación crearía problemas severos al negocio, si un sistema es complejo en extremo o si se trata de un negocio de importancia crítica.

Karl Wiegers [Wie03], de la empresa Process Impact Inc., desarrolló un formato útil (disponible en [www.processimpact.com/process\\_assets/srs\\_template.doc](http://www.processimpact.com/process_assets/srs_template.doc)) que sirve como guía para aquellos que deben crear una ERS completa. Su contenido normal es el siguiente:

#### Tabla de contenido

#### Revisión de la historia

##### 1. Introducción

Propósito

Convenciones del documento

Audiencia objetivo y sugerencias de lectura

Alcance del proyecto

Referencias

Características del producto

Clases y características del usuario

Ambiente de operación

Restricciones de diseño e implementación

Documentación para el usuario

Suposiciones y dependencias

##### 3. Características del sistema

Característica 1 del sistema

Característica 2 del sistema (y así sucesivamente)

##### 4. Requerimientos de la interfaz externa

Interfaces de usuario

Interfaces del hardware

Interfaces del software

Interfaces de las comunicaciones

##### 5. Otros requerimientos no funcionales

Requerimientos de desempeño

Requerimientos de seguridad

Requerimientos de estabilidad

Atributos de calidad del software

##### 6. Otros requerimientos

###### Apéndice A: Glosario

###### Apéndice B: Modelos de análisis

###### Apéndice C: Lista de conceptos

Puede obtenerse una descripción detallada de cada ERS si se descarga el formato desde la URL mencionada antes.



*Un aspecto clave durante la validación de los requerimientos es la consistencia. Utilice el modelo de análisis para asegurar que los requerimientos se han enunciado de manera consistente.*

**Validación.** La calidad de los productos del trabajo que se generan como consecuencia de la ingeniería de los requerimientos se evalúa durante el paso de validación. La validación de los requerimientos analiza la especificación<sup>5</sup> a fin de garantizar que todos ellos han sido enunciados sin ambigüedades; que se detectaron y corrigieron las inconsistencias, las omisiones y los errores, y que los productos del trabajo se presentan conforme a los estándares establecidos para el proceso, el proyecto y el producto.

El mecanismo principal de validación de los requerimientos es la revisión técnica (véase el capítulo 15). El equipo de revisión que los valida incluye ingenieros de software, clientes, usuarios y otros participantes, que analizan la especificación en busca de errores de contenido o de interpretación, de aspectos en los que tal vez se requiera hacer aclaraciones, falta de información, inconsistencias (problema notable cuando se hace la ingeniería de productos o sistemas grandes) y requerimientos en conflicto o irreales (no asequibles).



### Lista de verificación para validar requerimientos

Con frecuencia es útil analizar cada requerimiento en comparación con preguntas de verificación. A continuación se presentan algunas:

- ¿Los requerimientos están enunciados con claridad? ¿Podrían interpretarse mal?
- ¿Está identificada la fuente del requerimiento (por ejemplo, una persona, reglamento o documento)? ¿Se ha estudiado el planteamiento final del requerimiento en comparación con la fuente original?
- ¿El requerimiento está acotado en términos cuantitativos?
- ¿Qué otros requerimientos se relacionan con éste? ¿Están comparados con claridad por medio de una matriz de referencia cruzada u otro mecanismo?

### INFORMACIÓN

- ¿El requerimiento viola algunas restricciones del dominio?
- ¿Puede someterse a prueba el requerimiento? Si es así, ¿es posible especificar las pruebas (en ocasiones se denominan criterios de validación) para ensayar el requerimiento?
- ¿Puede rastrearse el requerimiento hasta cualquier modelo del sistema que se haya creado?
- ¿Es posible seguir el requerimiento hasta los objetivos del sistema o producto?
- ¿La especificación está estructurada en forma que lleva a entenderlo con facilidad, con referencias y traducción fáciles a productos del trabajo más técnicos?
- ¿Se ha creado un índice para la especificación?
- ¿Están enunciadas con claridad las asociaciones de los requerimientos con las características de rendimiento, comportamiento y operación? ¿Cuáles requerimientos parecen ser implícitos?

**Administración de los requerimientos.** Los requerimientos para sistemas basados en computadora cambian, y el deseo de modificarlos persiste durante toda la vida del sistema. La administración de los requerimientos es el conjunto de actividades que ayudan al equipo del proyecto a identificar, controlar y dar seguimiento a los requerimientos y a sus cambios en cualquier momento del desarrollo del proyecto.<sup>6</sup> Muchas de estas actividades son idénticas a las técnicas de administración de la configuración del software (TAS) que se estudian en el capítulo 22.

5 Recuerde que la naturaleza de la especificación variará con cada proyecto. En ciertos casos, la “especificación” no es más que un conjunto de escenarios de usuario. En otros, la especificación tal vez sea un documento que contiene escenarios, modelos y descripciones escritas.

6 La administración formal de los requerimientos sólo se practica para proyectos grandes que tienen cientos de requerimientos identificables. Para proyectos pequeños, esta actividad tiene considerablemente menos formalidad.



**HERRAMIENTAS DE SOFTWARE****Ingeniería de requerimientos**

**Objetivo:** Las herramientas de la ingeniería de los requerimientos ayudan a reunir éstos, a modelarlos, administrarlos y validarlos.

**Mecánica:** La mecánica de las herramientas varía. En general, éstas elaboran varios modelos gráficos (por ejemplo, UML) que ilustran los aspectos de información, función y comportamiento de un sistema.

Estos modelos constituyen la base de todas las demás actividades del proceso de software.

**Herramientas representativas:<sup>7</sup>**

En el sitio de Volere Requirements, en [www.volere.co.uk/tools.htm](http://www.volere.co.uk/tools.htm), se encuentra una lista razonablemente amplia (y actualizada) de herramientas para la ingeniería de requerimientos. En los capítulos 6 y 7 se estudian las herramientas que sirven para modelar aquéllos.

Las que se mencionan a continuación se centran en su administración.

**EasyRM**, desarrollada por Cybernetic Intelligence GmbH ([www.easy-rm.com](http://www.easy-rm.com)), construye un diccionario/glosario especial para proyectos, que contiene descripciones y atributos detallados de los requerimientos.

**Rational RequisitePro**, elaborada por Rational Software ([www-306.ibm.com/software/awdtools/reqpro/](http://www-306.ibm.com/software/awdtools/reqpro/)), permite a los usuarios construir una base de datos de requerimientos, representar relaciones entre ellos y organizarlos, indicar su prioridad y rastrearlos.

En el sitio de Volere ya mencionado, se encuentran muchas herramientas adicionales para administrar requerimientos, así como en la dirección [www.jiludwig.com/Requirements\\_Management\\_Tools.html](http://www.jiludwig.com/Requirements_Management_Tools.html)

**5.2 ESTABLECER LAS BASES**

En el caso ideal, los participantes e ingenieros de software trabajan juntos en el mismo equipo.<sup>8</sup> En esas condiciones, la ingeniería de requerimientos tan sólo consiste en sostener conversaciones significativas con colegas que sean miembros bien conocidos del equipo. Pero es frecuente que en la realidad esto sea muy diferente.

Los clientes o usuarios finales tal vez se encuentren en ciudades o países diferentes, quizás sólo tengan una idea vaga de lo que se requiere, puede ser que tengan opiniones en conflicto sobre el sistema que se va a elaborar, que posean un conocimiento técnico limitado o que dispondan de poco tiempo para interactuar con el ingeniero que recabará los requerimientos. Ninguna de estas posibilidades es deseable, pero todas son muy comunes y es frecuente verse forzado a trabajar con las restricciones impuestas por esta situación.

En las secciones que siguen se estudian las etapas requeridas para establecer las bases que permiten entender los requerimientos de software a fin de que el proyecto comience en forma tal que se mantenga avanzando hacia una solución exitosa.

**Identificación de los participantes**

Sommerville y Sawyer [Som97] definen *participante* como “cualquier persona que se beneficie en forma directa o indirecta del sistema en desarrollo”. Ya se identificaron los candidatos habituales: gerentes de operaciones del negocio, gerentes de producto, personal de mercadotecnia, clientes internos y externos, usuarios finales, consultores, ingenieros de producto, ingenieros de software e ingenieros de apoyo y mantenimiento, entre otros. Cada participante tiene un punto de vista diferente respecto del sistema, obtiene distintos beneficios cuando éste se desarrolla con éxito y corre distintos riesgos si fracasa el esfuerzo de construcción.

<sup>7</sup> Las herramientas mencionadas aquí no son obligatorias sino una muestra de las que hay en esta categoría. En la mayoría de los casos, los nombres de las herramientas son marcas registradas por sus respectivos desarrolladores.

<sup>8</sup> Este enfoque es ampliamente recomendable para proyectos que adoptan la filosofía de desarrollo de software ágil.



Un **participante** es cualquier persona que tenga interés directo o que se beneficie del sistema que se va a desarrollar.

Durante la concepción, debe hacerse la lista de personas que harán aportes cuando se reca- ben los requerimientos (véase la sección 5.3). La lista inicial crecerá cuando se haga contacto con los participantes porque a cada uno se le hará la pregunta: “¿A quién más piensa que debe consultarse?”



Cita:

“Ponga a tres participantes en un cuarto y pregúnteleles qué clase de sistema quieren. Es probable que escuche cuatro o más opiniones diferentes.”

Anónimo

## Reconocer los múltiples puntos de vista

Debido a que existen muchos participantes distintos, los requerimientos del sistema se explorarán desde muchos puntos de vista diferentes. Por ejemplo, el grupo de mercadotecnia se interesa en funciones y características que estimularán el mercado potencial, lo que hará que el nuevo sistema sea fácil de vender. Los gerentes del negocio tienen interés en un conjunto de características para que se elabore dentro del presupuesto y que esté listo para ocupar nichos de mercado definidos. Los usuarios finales tal vez quieran características que les resulten familiares y que sean fáciles de aprender y usar. Los ingenieros de software quizás piensen en funciones invisibles para los participantes sin formación técnica, pero que permitan una infraestructura que dé apoyo a funciones y características más vendibles. Los ingenieros de apoyo tal vez se centren en la facilidad del software para recibir mantenimiento.

Cada uno de estos integrantes (y otros más) aportará información al proceso de ingeniería de los requerimientos. A medida que se recaba información procedente de múltiples puntos de vista, los requerimientos que surjan tal vez sean inconsistentes o estén en conflicto uno con otro. Debe clasificarse toda la información de los participantes (incluso los requerimientos inconsistentes y conflictivos) en forma que permita a quienes toman las decisiones escoger para el sistema un conjunto de requerimientos que tenga coherencia interna.

## Trabajar hacia la colaboración

Si en un proyecto de software hay involucrados cinco participantes, tal vez se tengan cinco (o más) diferentes opiniones acerca del conjunto apropiado de requerimientos. En los primeros capítulos se mencionó que, para obtener un sistema exitoso, los clientes (y otros participantes) debían colaborar entre sí (sin pelear por insignificancias) y con los profesionales de la ingeniería de software. Pero, ¿cómo se llega a esta colaboración?

El trabajo del ingeniero de requerimientos es identificar las áreas de interés común (por ejemplo, requerimientos en los que todos los participantes estén de acuerdo) y las de conflicto o incongruencia (por ejemplo, requerimientos que desea un participante, pero que están en conflicto con las necesidades de otro). Es la última categoría la que, por supuesto, representa un reto.



### Uso de “puntos de prioridad”

Una manera de resolver requerimientos conflictivos y, al mismo tiempo, mejorar la comprensión de la importancia relativa de todos, es usar un esquema de “votación” con base en *puntos de prioridad*. Se da a todos los participantes cierto número de puntos de prioridad que pueden “gastarse” en cualquier número de requerimientos. Se presenta una lista de éstos y cada participante indica la importancia relativa de cada uno (desde su punto de vista)

### INFORMACIÓN

con la asignación de uno o más puntos de prioridad. Los puntos gastados ya no pueden utilizarse otra vez. Cuando un participante agota sus puntos de prioridad, ya no tiene la posibilidad de hacer algo con los requerimientos. El total de puntos asignados a cada requerimiento por los participantes da una indicación de la importancia general de cada requerimiento.

La colaboración no significa necesariamente que todos los requerimientos los defina un comité. En muchos casos, los participantes colaboran con la aportación de su punto de vista respecto de los requerimientos, pero un influyente “campeón del proyecto” (por ejemplo, el director



del negocio o un tecnólogo experimentado) toma la decisión final sobre los requerimientos que lo integrarán.

### Hacer las primeras preguntas

Las preguntas que se hacen en la concepción del proyecto deben estar “libres del contexto” [Gau89]. El primer conjunto de ellas se centran en el cliente y en otros participantes, en las metas y beneficios generales. Por ejemplo, tal vez se pregunte:

- ¿Quién está detrás de la solicitud de este trabajo?
- ¿Quién usará la solución?
- ¿Cuál será el beneficio económico de una solución exitosa?
- ¿Hay otro origen para la solución que se necesita?

Cita:  
"Es mejor conocer algunas preguntas que todas las respuestas."  
James nuber

Estas preguntas ayudan a identificar a todos los participantes con interés en el software que se va a elaborar. Además, las preguntas identifican el beneficio mensurable de una implementación exitosa y las posibles alternativas para el desarrollo de software personalizado.

Las preguntas siguientes permiten entender mejor el problema y hacen que el cliente exprese sus percepciones respecto de la solución:

- ¿Cuál sería una “buena” salida generada por una solución exitosa?
- ¿Qué problemas resolvería esta solución?
- ¿Puede mostrar (o describir) el ambiente de negocios en el que se usaría la solución?
- ¿Hay aspectos especiales del desempeño o restricciones que afecten el modo en el que se enfoque la solución?

¿Cuáles preguntas ayudarían a tener un entendimiento preliminar del problema?

Las preguntas finales se centran en la eficacia de la actividad de comunicación en sí. Gause y Weinberg [Gau89] las llaman “metapreguntas” y proponen la siguiente lista (abreviada):

- ¿Es usted la persona indicada para responder estas preguntas? ¿Sus respuestas son “oficiales”?
- ¿Mis preguntas son relevantes para el problema que se tiene?
- ¿Estoy haciendo demasiadas preguntas?
- ¿Puede otra persona dar información adicional?
- ¿Debería yo preguntarle algo más?

Cita:  
"El que hace una pregunta es tonto durante cinco minutos; el que no la hace será tonto para siempre."  
Proverbio chino

Estas preguntas (y otras) ayudarán a “romper el hielo” y a iniciar la comunicación, que es esencial para una indagación exitosa. Pero una reunión de preguntas y respuestas no es un enfoque que haya tenido un éxito apabullante. En realidad, la sesión de preguntas y respuestas sólo debe usarse para el primer encuentro y luego ser reemplazada por un formato de indagación de requerimientos que combine elementos de solución de problemas, negociación y especificación. En la sección 5.3 se presenta un enfoque de este tipo.

## 5.3 INDAGACIÓN DE LOS REQUERIMIENTOS

La indagación de los requerimientos (actividad también llamada *recabación de los requerimientos*) combina elementos de la solución de problemas, elaboración, negociación y especificación. A fin de estimular un enfoque colaborativo y orientado al equipo, los participantes trabajan juntos para identificar el problema, proponer elementos de la solución, negociar distintas visiones y especificar un conjunto preliminar de requerimientos para la solución [Zah90].<sup>9</sup>

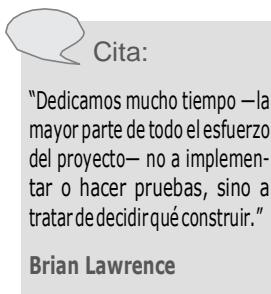
<sup>9</sup> En ocasiones se denomina a este enfoque *técnica facilitada de especificación de la aplicación* (TSEA).

## Recabación de los requerimientos en forma colaborativa

Se han propuesto muchos enfoques distintos para recabar los requerimientos en forma colaborativa. Cada uno utiliza un escenario un poco diferente, pero todos son variantes de los siguientes lineamientos básicos:

**?** ¿Cuáles son los lineamientos básicos para conducir una reunión a fin de recabar los requerimientos en forma colaborativa?

- Tanto ingenieros de software como otros participantes dirigen o intervienen en las reuniones.
- Se establecen reglas para la preparación y participación.
- Se sugiere una agenda con suficiente formalidad para cubrir todos los puntos importantes, pero con la suficiente informalidad para que estimule el libre flujo de ideas.
- Un “facilitador” (cliente, desarrollador o participante externo) controla la reunión.
- Se utiliza un “mecanismo de definición” (que pueden ser hojas de trabajo, tablas sueltas, etiquetas adhesivas, pizarrón electrónico, grupos de conversación o foro virtual).



La meta es identificar el problema, proponer elementos de la solución, negociar distintos enfoques y especificar un conjunto preliminar de requerimientos de la solución en una atmósfera que favorezca el logro de la meta. Para entender mejor el flujo de eventos conforme ocurren, se presenta un escenario breve que bosqueja la secuencia de hechos que llevan a la reunión para obtener requerimientos, a lo que sucede durante ésta y a lo que sigue después de ella.

Durante la concepción (véase la sección 5.2), hay preguntas y respuestas básicas que establecen el alcance del problema y la percepción general de lo que constituye una solución. Fuera de estas reuniones iniciales, el desarrollador y los clientes escriben una o dos páginas de “solicitud de producto”.

Se selecciona un lugar, fecha y hora para la reunión, se escoge un facilitador y se invita a asistir a integrantes del equipo de software y de otras organizaciones participantes. Antes de la fecha de la reunión, se distribuye la solicitud de producto a todos los asistentes.

Por ejemplo,<sup>10</sup> considere un extracto de una solicitud de producto escrita por una persona de mercadotecnia involucrada en el proyecto *CasaSegura*. Esta persona escribe la siguiente narración sobre la función de seguridad en el hogar que va a ser parte de *CasaSegura*:

Nuestras investigaciones indican que el mercado para los sistemas de administración del hogar crece a razón de 40% anual. La primera función de *CasaSegura* que llevemos al mercado deberá ser la de seguridad del hogar. La mayoría de la gente está familiarizada con “sistemas de alarma”, por lo que ésta deberá ser fácil de vender.

La función de seguridad del hogar protegería, o reconocería, varias “situaciones” indeseables, como acceso ilegal, incendio y niveles de monóxido de carbono, entre otros. Emplearía sensores inalámbricos para detectar cada situación. Sería programada por el propietario y telefonearía en forma automática a una agencia de vigilancia cuando detectara una situación como las descritas.

En realidad, durante la reunión para recabar los requerimientos, otros contribuirían a esta narración y se dispondría de mucha más información. Pero aun con ésta habría ambigüedad, sería probable que existieran omisiones y ocurrieran errores. Por ahora bastará la “descripción funcional” anterior.

Mientras se revisa la solicitud del producto antes de la reunión, se pide a cada asistente que elabore una lista de objetos que sean parte del ambiente que rodeará al sistema, los objetos

10 Este ejemplo (con extensiones y variantes) se usa para ilustrar métodos importantes de la ingeniería de software en muchos de los capítulos siguientes. Como ejercicio, sería provechoso que el lector realizara su propia reunión para recabar requerimientos y que desarrollara un conjunto de listas para ella.



que producirá éste y los que usará para realizar sus funciones. Además, se solicita a cada asistente que haga otra lista de servicios (procesos o funciones) que manipulen o interactúen con los objetos. Por último, también se desarrollan listas de restricciones (por ejemplo, costo, tamaño, reglas del negocio, etc.) y criterios de desempeño (como velocidad y exactitud). Se informa a los asistentes que no se espera que las listas sean exhaustivas, pero sí que reflejen la percepción que cada persona tiene del sistema.



Cita:

"Los hechos no dejan de existir porque se les ignore."

Aldous Huxley



*Evite el impulso de desechar alguna idea de un cliente con expresiones como "demasiado costosa" o "impráctica". La intención aquí es negociar una lista aceptable para todos. Para lograrlo, debe tenerse la mente abierta.*

Entre los objetos descritos por *CasaSegura* tal vez estén incluidos el panel de control, detectores de humo, sensores en ventanas y puertas, detectores de movimiento, alarma, un evento (activación de un sensor), una pantalla, una computadora, números telefónicos, una llamada telefónica, etc. La lista de servicios puede incluir *configurar el sistema, preparar la alarma, vigilar los sensores, marcar el teléfono, programar el panel de control y leer la pantalla* (observe que los servicios actúan sobre los objetos). En forma similar, cada asistente desarrollará una lista de restricciones (por ejemplo, el sistema debe reconocer cuando los sensores no estén operando, debe ser amistoso con el usuario, debe tener una interfaz directa con una línea telefónica estándar, etc.) y de criterios de desempeño (un evento en un sensor debe reconocerse antes de un segundo, debe implementarse un esquema de prioridad de eventos, etcétera).

Las listas de objetos pueden adherirse a las paredes del cuarto con el empleo de pliegos de papel grandes o con láminas adhesivas, o escribirse en un tablero. Alternativamente, las listas podrían plasmarse en un boletín electrónico, sitio web interno o en un ambiente de grupo de conversación para revisarlas antes de la reunión. Lo ideal es que cada entrada de las listas pueda manipularse por separado a fin de combinar las listas o modificar las entradas y agregar otras. En esta etapa, están estrictamente prohibidos las críticas y el debate.

Una vez que se presentan las listas individuales acerca de un área temática, el grupo crea una lista, eliminando las entradas redundantes o agregando ideas nuevas que surjan durante el análisis, pero no se elimina ninguna. Después de crear listas combinadas para todas las áreas temáticas, sigue el análisis, coordinado por el facilitador. La lista combinada se acorta, se alarga o se modifica su redacción para que refleje de manera apropiada al producto o sistema que se va a desarrollar. El objetivo es llegar a un consenso sobre la lista de objetos, servicios, restricciones y desempeño del sistema que se va a construir.

En muchos casos, un objeto o servicio descrito en la lista requerirá mayores explicaciones. Para lograr esto, los participantes desarrollan *miniespecificaciones* para las entradas en las listas.<sup>11</sup> Cada miniespecificación es una elaboración de un objeto o servicio. Por ejemplo, la correspondiente al objeto **Panel de control** de *CasaSegura* sería así:

El panel de control es una unidad montada en un muro, sus dimensiones aproximadas son de 9 por 5 pulgadas. Tiene conectividad inalámbrica con los sensores y con una PC. La interacción con el usuario tiene lugar por medio de un tablero que contiene 12 teclas. Una pantalla de cristal líquido de 3 por 3 pulgadas brinda retroalimentación al usuario. El software hace anuncios interactivos, como eco y funciones similares.

Las miniespecificaciones se presentan a todos los participantes para que sean analizadas. Se hacen adiciones, eliminaciones y otras modificaciones. En ciertos casos, el desarrollo de las miniespecificaciones descubrirá nuevos objetos, servicios o restricciones, o requerimientos de desempeño que se agregarán a las listas originales. Durante todos los análisis, el equipo debe posponer los aspectos que no puedan resolverse en la reunión. Se conserva una *lista de aspectos* para volver después a dichas ideas.

<sup>11</sup> En vez de crear una miniespecificación, muchos equipos de software eligen desarrollar escenarios del usuario llamados *casos de uso*. Éstos se estudian en detalle en la sección 5.4 y en el capítulo 6.

**CASASEGURA****Conducción de una reunión para recabar los requerimientos**

**La escena:** Sala de juntas. Está en marcha la primera reunión para recabar los requerimientos.

**Participantes:** Jamie Lazar, integrante del equipo de software; Vinod Raman, miembro del equipo de software; Ed Robbins, miembro del equipo de software; Doug Miller, gerente de ingeniería de software; tres trabajadores de mercadotecnia; un representante de ingeniería del producto, y un facilitador.

**La conversación:**

**Facilitador (apunta en un pizarrón):** De modo que ésa es la lista actual de objetos y servicios para la función de seguridad del hogar.

**Persona de mercadotecnia:** Eso la cubre, desde nuestro punto de vista.

**Vinod:** ¿No dijo alguien que quería que toda la funcionalidad de CasaSegura fuera accesible desde internet? Eso incluiría la función de seguridad, ¿o no?

**Persona de mercadotecnia:** Sí, así es... tendremos que añadir esa funcionalidad y los objetos apropiados.

**Facilitador:** ¿Agrega eso algunas restricciones?

**Jamie:** Sí, tanto técnicas como legales.

**Representante del producto:** ¿Qué significa eso?

**Jamie:** Nos tendríamos que asegurar de que un extraño no pueda ingresar al sistema, desactivarlo y robar en el lugar o hacer algo peor. Mucha responsabilidad sobre nosotros.

**Doug:** Muy cierto.

**Mercadotecnia:** Pero lo necesitamos así... sólo asegúrense de impedir que ingrese un extraño.

**Ed:** Eso es más fácil de decir que de hacer.

**Facilitador (interrumpe):** No quiero que debatamos esto ahora. Anotémoslo como un aspecto y continuemos.

(Doug, que es el secretario de la reunión, toma debida nota.)

**Facilitador:** Tengo la sensación de que hay más por considerar aquí.

(El grupo dedica los siguientes 20 minutos a mejorar y aumentar los detalles de la función de seguridad del hogar.)

**Despliegue de la función de calidad**

## PONTO CLAVE

El DFC define los requerimientos de forma que maximicen la satisfacción del cliente.



Todos desean implementar muchos requerimientos emocionantes, pero hay que tener cuidado. Así es como empiezan a "quedarse lisados los requerimientos". Pero en contrapartida, los requerimientos emocionantes llevan a un avance enorme del producto...

El *despliegue de la función de calidad* (DFC) es una técnica de administración de la calidad que traduce las necesidades del cliente en requerimientos técnicos para el software. El DFC "se concentra en maximizar la satisfacción del cliente a partir del proceso de ingeniería del software" [Zul92]. Para lograr esto, el DFC pone el énfasis en entender lo que resulta valioso para el cliente y luego despliega dichos valores en todo el proceso de ingeniería. El DFC identifica tres tipos de requerimientos [Zul92]:

**Requerimientos normales.** Objetivos y metas que se establecen para un producto o sistema durante las reuniones con el cliente. Si estos requerimientos están presentes, el cliente queda satisfecho. Ejemplos de requerimientos normales son los tipos de gráficos pendidos para aparecer en la pantalla, funciones específicas del sistema y niveles de rendimiento definidos.

**Requerimientos esperados.** Están implícitos en el producto o sistema y quizás sean tan importantes que el cliente no los mencione de manera explícita. Su ausencia causará mucha insatisfacción. Algunos ejemplos de requerimientos esperados son: fácil interacción humano/máquina, operación general correcta y confiable, y facilidad para instalar el software.

**Requerimientos emocionantes.** Estas características van más allá de las expectativas del cliente y son muy satisfactorias si están presentes. Por ejemplo, el software para un nuevo teléfono móvil viene con características estándar, pero si incluye capacidades inesperadas (como pantalla sensible al tacto, correo de voz visual, etc.) agrada a todos los usuarios del producto.

**WebRef**

En la dirección [www.qfdi.org](http://www.qfdi.org) se encuentra información útil sobre el DFC.

Aunque los conceptos del DFC son aplicables en todo el proceso del software [Par96a], hay técnicas específicas de aquél que pueden aplicarse a la actividad de indagación de los requerimientos. El DFC utiliza entrevistas con los clientes, observación, encuestas y estudio de datos históricos (por ejemplo, reportes de problemas) como materia prima para la actividad de recabación



de los requerimientos. Después, estos datos se llevan a una tabla de requerimientos —llamada *tabla de la voz del cliente*— que se revisa con el cliente y con otros participantes. Luego se emplean varios diagramas, matrices y métodos de evaluación para extraer los requerimientos esperados y tratar de percibir requerimientos emocionantes [Aka04].

### Escenarios de uso

A medida que se reúnen los requerimientos, comienza a materializarse la visión general de funciones y características del sistema. Sin embargo, es difícil avanzar hacia actividades más técnicas de la ingeniería de software hasta no entender cómo emplearán los usuarios finales dichas funciones y características. Para lograr esto, los desarrolladores y usuarios crean un conjunto de escenarios que identifican la naturaleza de los usos para el sistema que se va a construir. Los escenarios, que a menudo se llaman *casos de uso* [Jac92], proporcionan la descripción de la manera en la que se utilizará el sistema. Los casos de uso se estudian con más detalle en la sección 5.4.

## CASASEGURA



### Desarrollo de un escenario preliminar de uso

**La escena:** Una sala de juntas, donde continúa la primera reunión para recabar los requerimientos.

**Participantes:** Jamie Lazar, integrante del equipo de software; Vinod Raman, miembro del equipo de software; Ed Robbins, miembro del equipo de software; Doug Miller, gerente de ingeniería de software; tres personas de mercadotecnia; un representante de ingeniería del producto, y un facilitador.

#### La conversación:

**Facilitador:** Hemos estado hablando sobre la seguridad para el acceso a la funcionalidad de *CasaSegura* si ha de ser posible el ingreso por internet. Me gustaría probar algo. Desarrollemos un escenario de uso para entrar a la función de seguridad.

**Jamie:** ¿Cómo?

**Facilitador:** Podríamos hacerlo de dos maneras, pero de momento mantengamos las cosas informales. Díganos (señala a una persona de mercadotecnia), ¿cómo visualiza el acceso al sistema?

**Persona de mercadotecnia:** Um... bueno, es la clase de cosa que haría si estuviera fuera de casa y tuviera que dejar entrar a alguien a ella —por ejemplo, una trabajadora doméstica o un técnico de reparaciones— que no tuviera el código de seguridad.

**Facilitador (sonríe):** Ésa es la razón por la que lo hace... dígame, ¿cómo lo haría en realidad?

**Persona de mercadotecnia:** Bueno... lo primero que necesitaría sería una PC. Entraría a un sitio web que mantendríamos para todos los usuarios de *CasaSegura*. Daría mi identificación de usuario y...

**Vinod (interrumpe):** La página web tendría que ser segura, encriptada, para garantizar que estuviéramos seguros y...

**Facilitador (interrumpe):** Ésa es buena información, Vinod, pero es técnica. Centrémonos en cómo emplearía el usuario final esta capacidad, ¿está bien?

**Vinod:** No hay problema.

**Persona de mercadotecnia:** Decía que entraría a un sitio web y daría mi identificación de usuario y dos niveles de clave.

**Jamie:** ¿Qué pasa si olvido mi clave?

**Facilitador (interrumpe):** Buena observación, Jamie, pero no entraremos a ella por ahora. Lo anotaremos y la llamaremos una excepción. Estoy seguro de que habrá otras.

**Persona de mercadotecnia:** Despues de que introdujera las claves, aparecería una pantalla que representaría todas las funciones de *CasaSegura*. Seleccionaría la función de seguridad del hogar. El sistema pediría que verificara quién soy, pidiendo mi dirección o número telefónico o algo así. Entonces aparecería un dibujo del panel de control del sistema de seguridad y la lista de funciones que puede realizar —activar el sistema, desactivar el sistema o desactivar uno o más sensores—. Supongo que también me permitiría reconfigurar las zonas de seguridad y otras cosas como ésa, pero no estoy seguro.

(Mientras la persona de mercadotecnia habla, Doug toma muchas notas; esto forma la base para el primer escenario informal de uso. Alternativamente, hubiera podido pedirse a la persona de mercadotecnia que escribiera el escenario, pero esto se hubiera hecho fuera de la reunión.)

### Indagación de los productos del trabajo

Los productos del trabajo generados como consecuencia de la indagación de los requerimientos variarán en función del tamaño del sistema o producto que se va a construir. Para la mayoría de sistemas, los productos del trabajo incluyen los siguientes:

?

¿Qué información se produce como consecuencia de recabar los requerimientos?

- Un enunciado de la necesidad y su factibilidad.
- Un enunciado acotado del alcance del sistema o producto.
- Una lista de clientes, usuarios y otros participantes que intervienen en la indagación de los requerimientos.
- Una descripción del ambiente técnico del sistema.
- Una lista de requerimientos (de preferencia organizados por función) y las restricciones del dominio que se aplican a cada uno.
- Un conjunto de escenarios de uso que dan perspectiva al uso del sistema o producto en diferentes condiciones de operación.
- Cualesquiera prototipos desarrollados para definir requerimientos.

Cada uno de estos productos del trabajo es revisado por todas las personas que participan en la indagación de los requerimientos.

## 5.4 DESARROLLO DE CASOS DE USO

En un libro que analiza cómo escribir casos de uso eficaces, Alistair Cockburn [Coc01b] afirma que “un caso de uso capta un contrato [...] [que] describe el comportamiento del sistema en distintas condiciones en las que el sistema responde a una petición de alguno de sus participantes [...]. En esencia, un caso de uso narra una historia estilizada sobre cómo interactúa un usuario final (que tiene cierto número de roles posibles) con el sistema en circunstancias específicas. La historia puede ser un texto narrativo, un lineamiento de tareas o interacciones, una descripción basada en un formato o una representación diagramática. Sin importar su forma, un caso de uso ilustra el software o sistema desde el punto de vista del usuario final.

El primer paso para escribir un caso de uso es definir un conjunto de “actores” que estarán involucrados en la historia. Los *actores* son las distintas personas (o dispositivos) que usan el sistema o producto en el contexto de la función y comportamiento que va a describirse. Los actores representan los papeles que desempeñan las personas (o dispositivos) cuando opera el sistema. Con una definición más formal, un *actor* es cualquier cosa que se comunique con el sistema o producto y que sea externo a éste. Todo actor tiene uno o más objetivos cuando utiliza el sistema.

Es importante notar que un actor y un usuario final no necesariamente son lo mismo. Un usuario normal puede tener varios papeles diferentes cuando usa el sistema, mientras que un actor representa una clase de entidades externas (gente, con frecuencia pero no siempre) que sólo tiene un papel en el contexto del caso de uso. Por ejemplo, considere al operador de una máquina (un usuario) que interactúa con la computadora de control de una celda de manufactura que contiene varios robots y máquinas de control numérico. Después de una revisión cuidadosa de los requerimientos, el software para la computadora de control requiere cuatro diferentes modos (papeles) para la interacción: modo de programación, modo de prueba, modo de vigilancia y modo de solución de problemas. Por tanto, es posible definir cuatro actores: programador, probador, vigilante y solucionador de problemas. En ciertos casos, el operador de la máquina desempeñará todos los papeles. En otros, distintas personas tendrán el papel de cada actor.

Debido a que la indagación de los requerimientos es una actividad evolutiva, no todos los actores son identificados en la primera iteración. En ésta es posible identificar a los actores principales [Jac92], y a los secundarios cuando se sabe más del sistema. Los *actores principales* interactúan para lograr la función requerida del sistema y obtienen el beneficio previsto de éste. Trabajan con el software en forma directa y con frecuencia. Los *actores secundarios* dan apoyo al sistema, de modo que los primarios puedan hacer su trabajo.



### CLAVE

Los casos de uso se definen desde el punto de vista de un actor. Un actor es un papel que desempeñan las personas (usuarios) o los dispositivos cuando interactúan con el software.

#### WebRef

Un artículo excelente sobre casos de uso puede descargarse desde la dirección [www.ibm.com/developerworks/webservices/library/codesign7.html](http://www.ibm.com/developerworks/webservices/library/codesign7.html)



**SEP**  
SECRETARÍA DE  
EDUCACIÓN PÚBLICA



Una vez identificados los actores, es posible desarrollar casos de uso. Jacobson [Jac92] sugiere varias preguntas<sup>12</sup> que debe responder un caso de uso:

**?** ¿Qué se necesita saber a fin de desarrollar un caso de uso eficaz?

- ¿Quién es el actor principal y quién(es) el(s) secundario(s)?
- ¿Cuáles son los objetivos de los actores?
- ¿Qué precondiciones deben existir antes de comenzar la historia?
- ¿Qué tareas o funciones principales son realizadas por el actor?
- ¿Qué excepciones deben considerarse al describir la historia?
- ¿Cuáles variaciones son posibles en la interacción del actor?
- ¿Qué información del sistema adquiere, produce o cambia el actor?
- ¿Tendrá que informar el actor al sistema acerca de cambios en el ambiente externo?
- ¿Qué información desea obtener el actor del sistema?
- ¿Quiere el actor ser informado sobre cambios inesperados?

En relación con los requerimientos básicos de *CasaSegura*, se definen cuatro actores: **propietario de la casa** (usuario), **gerente de arranque** (tal vez la misma persona que el **propietario de la casa**, pero en un papel diferente), **sensores** (dispositivos adjuntos al sistema) y **subsistema de vigilancia y respuesta** (estación central que vigila la función de seguridad de la casa de *CasaSegura*). Para fines de este ejemplo, consideraremos sólo al actor llamado **propietario de la casa**. Éste interactúa con la función de seguridad de la casa en varias formas distintas con el empleo del panel de control de la alarma o con una PC:

- Introduce una clave que permita todas las demás interacciones.
- Pregunta sobre el estado de una zona de seguridad.
- Interroga acerca del estado de un sensor.
- En una emergencia, oprime el botón de pánico.
- Activa o desactiva el sistema de seguridad.

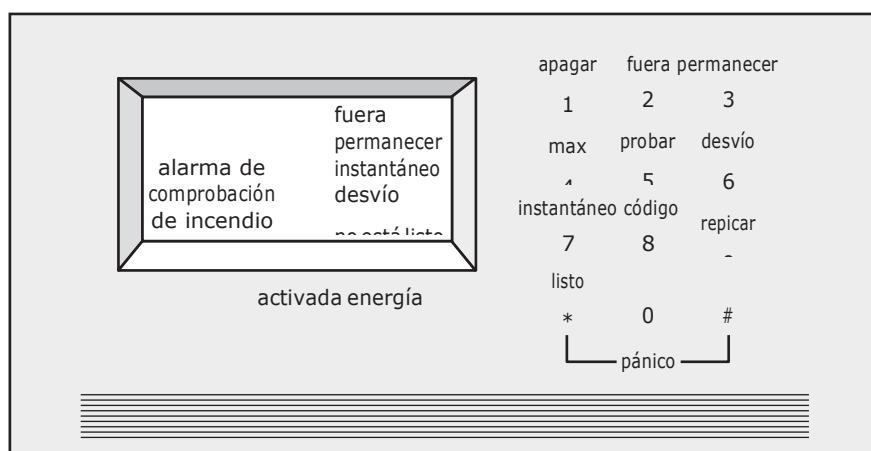
Considerando la situación en la que el propietario de la casa usa el panel de control, a continuación se plantea el caso de uso básico para la activación del sistema:<sup>13</sup>

1. El propietario observa el panel de control de *CasaSegura* (véase la figura 5.1) para determinar si el sistema está listo para recibir una entrada. Si el sistema no está listo, se muestra el mensaje *no está listo* en la pantalla de cristal líquido y el propietario debe cerrar físicamente ventanas o puertas de modo que desaparezca dicho mensaje [el mensaje *no está listo* implica que un sensor está abierto; por ejemplo, que una puerta o ventana está abierta].
2. El propietario usa el teclado para introducir una clave de cuatro dígitos. La clave se compara con la que guarda el sistema como válida. Si la clave es incorrecta, el panel de control emitirá un sonido una vez y se reiniciará para recibir una entrada adicional. Si la clave es correcta, el panel de control espera otras acciones.
3. El propietario selecciona y teclea *permanecer* o *fuerza* (véase la figura 5.1) para activar el sistema. La entrada *permanecer* activa sólo sensores perimetrales (se desactivan los sensores de detección de movimiento interior). La entrada *fuerza* activa todos los sensores.
4. Cuando ocurre una activación, el propietario observa una luz roja de alarma.

12 Las preguntas de Jacobson se han ampliado para que den una visión más completa del contenido del caso de uso.

13 Observe que este caso de uso difiere de la situación en la que se accede al sistema a través de internet. En este caso, la interacción es por medio del panel de control y no con la interfaz de usuario gráfica (GUI) que se da cuando se emplea una PC.

**FIGURA 5.1**  
**Panel de control**  
**de CasaSegura**



Es frecuente que los casos de uso se escriban de manera informal. Sin embargo, utilice el formato que se presenta aquí para asegurar que se incluyen todos los aspectos clave.

El caso de uso básico presenta una historia de alto nivel que describe la interacción entre el actor y el sistema.

En muchas circunstancias, los casos de uso son más elaborados a fin de que brinden muchos más detalles sobre la interacción. Por ejemplo, Cockburn [Coc01b] sugiere el formato siguiente para hacer descripciones detalladas de casos de uso:

**Caso de uso:** *IniciarVigilancia*

**Actor principal:** Propietario.

**Objetivo en contexto:** Preparar el sistema para que vigile los sensores cuando el propietario salga de la casa o permanezca dentro.

**Precondiciones:** El sistema se ha programado para recibir una clave y reconocer distintos sensores.

**Disparador:** El propietario decide "preparar" el sistema, por ejemplo, para que encienda las funciones de alarma.

#### **Escenario:**

1. Propietario: observa el panel de control
2. Propietario: introduce una clave
3. Propietario: selecciona "permanecer" o "fuera"
4. Propietario: observa una luz roja de alarma que indica que *CasaSegura* ha sido activada.

#### **Excepciones:**

1. El panel de control *no está listo*: el propietario verifica todos los sensores para determinar cuáles están abiertos; los cierra.
2. La clave es incorrecta (el panel de control suena una vez): el propietario introduce la clave correcta.
3. La clave no es reconocida: debe contactarse el subsistema de vigilancia y respuesta para reprogramar la clave.
4. Se elige *permanecer*: el panel de control suena dos veces y se enciende un letrero luminoso que dice *permanecer*; se activan los sensores del perímetro.

5. Se selecciona *fuera*: el panel de control suena tres veces y se enciende un letrero luminoso que dice *fuera*; se activan todos los sensores.

DEPARTAMENTO NACIONAL  
DE EVALUACIÓN PARA  
LA EDUCACIÓN SUPERIOR, A.C.



CENEVAL®



**EDITORIAL:**  
**guiaceneval.mx**



<b>Prioridad:</b>	Esencial, debe implementarse
<b>Cuándo estará disponible:</b>	En el primer incremento
<b>Frecuencia de uso:</b>	Muchas veces por día
<b>Canal para el actor:</b>	A través de la interfaz del panel de control
<b>Actores secundarios:</b>	
Técnico de apoyo, sensores	
<b>Canales para los actores secundarios:</b>	
Técnico de apoyo: línea telefónica	
Sensores: interfaces cableadas y frecuencia de radio	

#### Aspectos pendientes:

1. ¿Debe haber una forma de activar el sistema sin usar clave o con una clave abreviada?
2. ¿El panel de control debe mostrar mensajes de texto adicionales?
3. ¿De cuánto tiempo dispone el propietario para introducir la clave a partir del momento en el que se oprime la primera tecla?
4. ¿Hay una forma de desactivar el sistema antes de que se active en realidad?

Los casos de uso para otras interacciones de **propietario** se desarrollarían en una forma similar. Es importante revisar con cuidado cada caso de uso. Si algún elemento de la interacción es ambiguo, es probable que la revisión del caso de uso lo detecte.

## CASASEGURA



### Desarrollo de un diagrama de caso de uso de alto nivel

**La escena:** Sala de juntas, continúa la reunión para recabar los requerimientos.

**Participantes:** Jamie Lazar, miembro del equipo de software; Vinod Roman, integrante del equipo de software; Ed Robbins, integrante del equipo de software; Doug Miller, gerente de ingeniería de software; tres miembros de mercadotecnia; un representante de ingeniería del producto; un facilitador.

#### La conversación:

**Facilitador:** Hemos pasado un buen tiempo hablando de la función de seguridad del hogar de *CasaSegura*. Durante el receso hice un diagrama de caso de uso para resumir los escenarios importantes que forman parte de esta función. Veámoslo.

(Todos los asistentes observan la figura 5.2.)

**Jamie:** Estoy aprendiendo la notación UML.<sup>14</sup> Veo que la función de seguridad del hogar está representada por el rectángulo grande con óvalos en su interior, ¿verdad? ¿Y los óvalos representan los casos de uso que hemos escrito?

**Facilitador:** Sí. Y las figuras pegadas representan a los actores —personas o cosas que interactúan con el sistema según los describe

el caso de uso... — ; iah! usé el cuadrado para representar un actor que no es persona... en este caso, sensores.

**Doug:** ¿Es válido eso en UML?

**Facilitador:** La legalidad no es lo importante. El objetivo es comunicar información. Veo que usar una figura humana para representar un equipo sería erróneo. Así que adapté las cosas un poco. No pienso que genere problemas.

**Vinod:** Está bien, entonces tenemos narraciones de casos de uso para cada óvalo. ¿Necesitamos desarrollarlas con base en los formatos sobre los que he leído?

**Facilitador:** Es probable, pero eso puede esperar hasta que hayamos considerado otras funciones de *CasaSegura*.

**Persona de mercadotecnia:** Esperen, he estado observando este diagrama y de pronto me doy cuenta de que hemos olvidado algo.

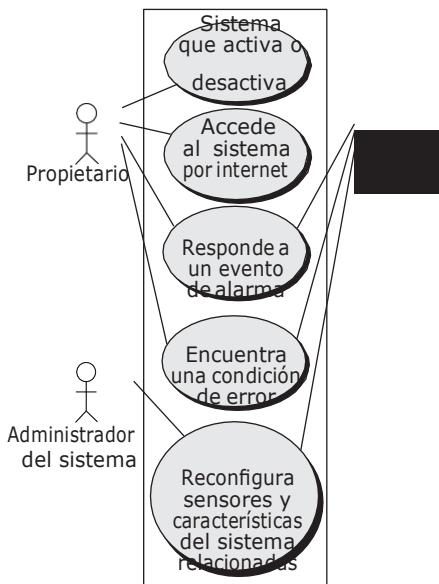
**Facilitador:** ¿De verdad? Dime, ¿qué hemos olvidado?

(La reunión continúa.)



FIGURA 5.2

Diagrama de caso de uso de UML para la función de seguridad del hogar de CasaSegura



### Desarrollo de un caso de uso

**Objetivo:** Ayudar a desarrollar casos de uso proporcionando formatos y mecanismos automatizados para evaluar la claridad y consistencia.

**Mecánica:** La mecánica de las herramientas varía. En general, las herramientas para casos de uso dan formatos con espacios en blanco para ser llenados y crear así casos eficaces. La mayor parte de la funcionalidad de los casos de uso está incrustada en un conjunto más amplio de funciones de ingeniería de los requerimientos.

### HERRAMIENTAS DE SOFTWARE

#### Herramientas representativas<sup>15</sup>

La gran mayoría de herramientas de análisis del modelado basadas en UML dan apoyo tanto de texto como gráfico para el desarrollo y modelado de casos de uso.

*Objects by Design*

([www.objectsbydesign.com/tools/umlttools\\_byCompany.html](http://www.objectsbydesign.com/tools/umlttools_byCompany.html)) proporciona vínculos exhaustivos con herramientas de este tipo.

## 5.5 ELABORACIÓN DEL MODELO DE LOS REQUERIMIENTOS<sup>16</sup>

El objetivo del modelo del análisis es describir los dominios de información, función y comportamiento que se requieren para un sistema basado en computadora. El modelo cambia en forma dinámica a medida que se aprende más sobre el sistema por construir, y otros participantes comprenden más lo que en realidad requieren. Por esa razón, el modelo del análisis es una fotografía de los requerimientos en cualquier momento dado. Es de esperar que cambie.

A medida que evoluciona el modelo de requerimientos, ciertos elementos se vuelven relativamente estables, lo que da un fundamento sólido para diseñar las tareas que sigan. Sin embargo, otros elementos del modelo son más volátiles, lo que indica que los participantes todavía no entienden bien los requerimientos para el sistema. En los capítulos 6 y 7 se presentan en

15 Las herramientas mencionadas aquí no son obligatorias, sino una muestra de las que hay en esta categoría. En la mayoría de casos, los nombres de las herramientas son marcas registradas por sus respectivos desarrolladores.

16 En este libro se usan como sinónimos las expresiones *modelar el análisis* y *modelar los requerimientos*. Ambos se



detalle el modelo del análisis y los métodos que se usan para construirlo. En las secciones siguientes se da un panorama breve.

### Elementos del modelo de requerimientos

Hay muchas formas diferentes de concebir los requerimientos para un sistema basado en computadora. Algunos profesionales del software afirman que es mejor seleccionar un modo de representación (por ejemplo, el caso de uso) y aplicarlo hasta excluir a todos los demás. Otros piensan que es más benéfico usar cierto número de modos de representación distintos para ilustrar el modelo de requerimientos. Los modos diferentes de representación fuerzan a considerar los requerimientos desde distintos puntos de vista, enfoque que tiene una probabilidad mayor de detectar omisiones, inconsistencia y ambigüedades.

Los elementos específicos del modelo de requerimientos están determinados por el método de análisis de modelado (véanse los capítulos 6 y 7) que se use. No obstante, la mayoría de modelos tiene en común un conjunto de elementos generales.



*Siempre es buena idea involucrar a los participantes. Una de las mejores formas de lograrlo es hacer que cada uno escriba casos de uso que narran el modo en el que se utilizará el software.*



*Una forma de aislar las clases es buscar sustantivos descriptivos en un caso de usuario expresado con texto. Al menos algunos de ellos serán candidatos cercanos. Sobre esto se habla más en el capítulo 8.*



Un estado es un modo de comportamiento observable desde el exterior. Los estímulos externos ocasionan transiciones entre los estados.

**Elementos basados en el escenario.** El sistema se describe desde el punto de vista del usuario con el empleo de un enfoque basado en el escenario. Por ejemplo, los casos de uso básico (véase la sección 5.4) y sus diagramas correspondientes de casos de uso (véase la figura 5.2) evolucionan hacia otros más elaborados que se basan en formatos. Los elementos del modelo de requerimientos basados en el escenario con frecuencia son la primera parte del modelo en desarrollo. Como tales, sirven como entrada para la creación de otros elementos de modelado. La figura 5.3 ilustra un diagrama de actividades UML<sup>17</sup> para indagar los requerimientos y representarlos con el empleo de casos de uso. Se aprecian tres niveles de elaboración que culminan en una representación basada en el escenario.

**Elementos basados en clases.** Cada escenario de uso implica un conjunto de objetos que se manipulan cuando un actor interactúa con el sistema. Estos objetos se clasifican en clases: conjunto de objetos que tienen atributos similares y comportamientos comunes. Por ejemplo, para ilustrar la clase **Sensor** de la función de seguridad de *Casa Segura* (véase la figura 5.4), puede utilizarse un diagrama de clase UML. Observe que el diagrama enumera los atributos de los sensores (por ejemplo, nombre, tipo, etc.) y las operaciones (por ejemplo, *identificar* y *permitir*) que se aplican para modificarlos. Además de los diagramas de clase, otros elementos de modelado del análisis ilustran la manera en la que las clases colaboran una con otra y las relaciones e interacciones entre ellas. Esto se analiza con más detalle en el capítulo 7.

**Elementos de comportamiento.** El comportamiento de un sistema basado en computadora tiene un efecto profundo en el diseño que se elija y en el enfoque de implementación que se aplique. Por tanto, el modelo de requerimientos debe proveer elementos de modelado que ilustran el comportamiento.

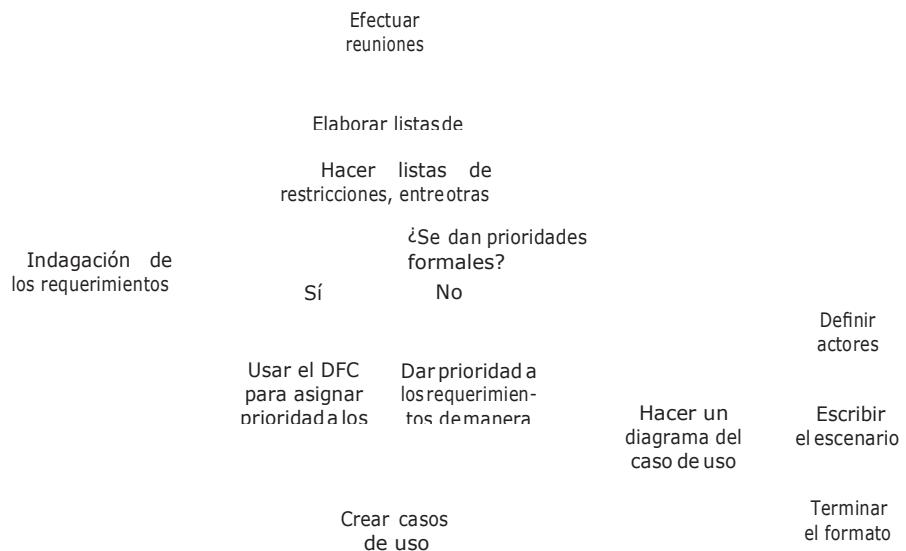
El *diagrama de estado* es un método de representación del comportamiento de un sistema que ilustra sus estados y los eventos que ocasionan que el sistema cambie de estado. Un *estado* es cualquier modo de comportamiento observable desde el exterior. Además, el diagrama de estado indica acciones (como la activación de un proceso, por ejemplo) tomadas como consecuencia de un evento en particular.

Para ilustrar el uso de un diagrama de estado, considere el software incrustado dentro del panel de control de *CasaSegura* que es responsable de leer las entradas que hace el usuario. En la figura 5.5 se presenta un diagrama de estado UML simplificado.

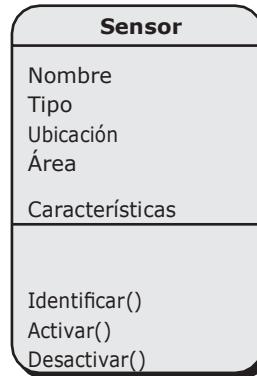
Además de las representaciones de comportamiento del sistema como un todo, también es posible modelar clases individuales. Sobre esto se presentan más análisis en el capítulo 7.

<sup>17</sup> En el apéndice 1 se presenta un instructivo breve sobre UML, para aquellos lectores que no estén familiarizados con dicha notación.

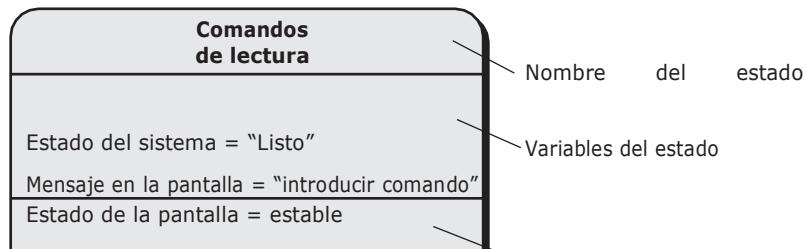
**FIGURA 5.3**  
Diagramas de actividad del UML para indagar los requerimientos



**FIGURA 5.4**  
Diagrama de clase para un sensor



**FIGURA 5.5**  
Notación UML del diagrama de estado



## CASASEGURA



### Modelado preliminar del comportamiento

**La escena:** Sala de juntas, continúa la reunión de requerimientos.

**Participantes:** Jamie Lazar, integrante del equipo de software; Vinod Raman, miembro del equipo de software; Ed Robbins, integrante del equipo de software; Doug Miller, gerente de ingeniería de software; tres trabajadores de mercadotecnia; un representante de ingeniería del producto, y un facilitador.

#### La conversación:

**Facilitador:** Estamos por terminar de hablar sobre la funcionalidad de seguridad del hogar de *CasaSegura*. Pero antes, quisiera que analizáramos el comportamiento de la función.

**Persona de mercadotecnia:** No entiendo lo que quiere decir con *comportamiento*.

**Ed (sonríe):** Es cuando le das un "tiempo fuera" al producto si se porta mal.

**Facilitador:** No exactamente. Permítanme explicarlo.

(El facilitador explica al equipo encargado de recabar los requerimientos y los fundamentos de modelado del comportamiento.)

**Persona de mercadotecnia:** Esto parece un poco técnico. No estoy seguro de ser de ayuda aquí.

**Facilitador:** Seguro que lo serás. ¿Qué comportamiento se observa desde el punto de vista de un usuario?

**Persona de mercadotecnia:** Mmm... bueno, el sistema estará vigilando los sensores. Leerá comandos del propietario. Mostrará su estado.

**Facilitador:** ¿Ves?, lo puedes hacer.

**Jamie:** También estará *interrogando* a la PC para determinar si hay alguna entrada desde ella, por ejemplo, un acceso por internet o información sobre la configuración.

**Vinod:** Sí, en realidad, *configurar* el sistema es un estado por derecho propio.

**Doug:** Muchachos, lo hacen bien. Pensemos un poco más... ¿hay alguna forma de hacer un diagrama de todo esto?

**Facilitador:** Sí la hay, pero la dejaremos para la próxima reunión.

**Elementos orientados al flujo.** La información se transforma cuando fluye a través de un sistema basado en computadora. El sistema acepta entradas en varias formas, aplica funciones para transformarla y produce salidas en distintos modos. La entrada puede ser una señal de control transmitida por un transductor, una serie de números escritos con el teclado por un operador humano, un paquete de información enviado por un enlace de red o un archivo grande de datos recuperado de un almacenamiento secundario. La transformación quizás incluya una sola comparación lógica, un algoritmo numérico complicado o un enfoque de regla de inferencia para un sistema experto. La salida quizás encienda un diodo emisor de luz o genere un informe de 200 páginas. En efecto, es posible crear un modelo del flujo para cualquier sistema basado en computadora, sin importar su tamaño y complejidad. En el capítulo 7 se hace un análisis más detallado del modelado del flujo.

### Patrones de análisis

Cualquiera que haya hecho la ingeniería de los requerimientos en varios proyectos de software ha observado que ciertos problemas son recurrentes en todos ellos dentro de un dominio de aplicación específico.<sup>18</sup> Estos *patrones de análisis* [Fow97] sugieren soluciones (por ejemplo, una clase, función o comportamiento) dentro del dominio de la aplicación que pueden volverse a utilizar cuando se modelen muchas aplicaciones.

Geyer-Schulz y Hahsler [Gey01] sugieren dos beneficios asociados con el uso de patrones de análisis:

En primer lugar, los patrones de análisis aceleran el desarrollo de los modelos de análisis abstracto que capturan los principales requerimientos del problema concreto, debido a que proveen modelos de análisis reutilizables con ejemplos, así como una descripción de sus ventajas y limitaciones. En se-

18 En ciertos casos, los problemas vuelven a suceder sin importar el dominio de la aplicación. Por ejemplo, son comunes las características y funciones usadas para resolver problemas de la interfaz de usuario sin importar el dominio de la aplicación en consideración.



gundo lugar, los patrones de análisis facilitan la transformación del modelo de análisis en un modelo del diseño, sugiriendo patrones de diseño y soluciones confiables para problemas comunes.

Los patrones de análisis se integran en el modelo del análisis, haciendo referencia al nombre del patrón. También se guardan en un medio de almacenamiento de modo que los ingenieros de requerimientos usen herramientas de búsqueda para encontrarlos y aplicarlos. La información sobre el patrón de análisis (y otros tipos de patrones) se presenta en un formato estándar [Gey01]<sup>19</sup> que se estudia con más detalle en el capítulo 12. En el capítulo 7 se dan ejemplos de patrones de análisis y más detalles de este tema.

## 5.6 REQUERIMIENTOS DE LAS NEGOCIACIONES



Cita:

"Un compromiso es el arte de dividir un pastel en forma tal que todos crean que tienen el trozo mayor."

Ludwig Erhard

En un contexto ideal de la ingeniería de los requerimientos, las tareas de concepción, indagación y elaboración determinan los requerimientos del cliente con suficiente detalle como para avanzar hacia las siguientes actividades de la ingeniería de software. Desafortunadamente, esto rara vez ocurre. En realidad, se tiene que entrar en negociaciones con uno o varios participantes. En la mayoría de los casos, se pide a éstos que evalúen la funcionalidad, desempeño y otras características del producto o sistema, en contraste con el costo y el tiempo para entrar al mercado. El objetivo de esta negociación es desarrollar un plan del proyecto que satisfaga las necesidades del participante y que al mismo tiempo refleje las restricciones del mundo real (por ejemplo, tiempo, personas, presupuesto, etc.) que se hayan establecido al equipo del software. Las mejores negociaciones buscan un resultado "ganar-ganar".<sup>20</sup> Es decir, los participantes ganan porque obtienen el sistema o producto que satisface la mayoría de sus necesidades y usted (como miembro del equipo de software) gana porque trabaja con presupuestos y plazos realistas y asequibles.

Boehm [Boe98] define un conjunto de actividades de negociación al principio de cada iteración del proceso de software. En lugar de una sola actividad de comunicación con el cliente, se definen las actividades siguientes:

1. Identificación de los participantes clave del sistema o subsistema.
2. Determinación de las "condiciones para ganar" de los participantes.



### El arte de la negociación

Aprender a negociar con eficacia le servirá en su vida personal y técnica. Es útil considerar los lineamientos que siguen:

1. *Reconocer que no es una competencia.* Para tener éxito, ambas partes tienen que sentir que han ganado o logrado algo. Las dos tienen que llegar a un compromiso.
2. *Mapear una estrategia.* Decidir qué es lo que le gustaría lograr; qué quiere obtener la otra parte y cómo hacer para que ocurran las dos cosas.
3. *Escuchar activamente.* Notrabaje en la formulación de su respuesta mientras la otra parte esté hablando. Escúchela. Es pro-

### INFORMACIÓN

bable que obtenga conocimientos que lo ayuden a negociar mejor su posición.

4. *Centrarse en los intereses de la otra parte.* Si quiere evitar conflictos, no adopte posiciones inamovibles.
5. *No lo tome en forma personal.* Céntrese en el problema que necesita resolverse.
6. *Sea creativo.* Si están empantanados, no tenga miedo de pensar fuera de los moldes.
7. *Esté listo para comprometerse.* Una vez que se llegue a un acuerdo, no titubee; comprométase con él y címplalo.

<sup>19</sup> En la bibliografía existen varias propuestas de formatos para patrones. Si el lector tiene interés, consulte [Fow97], [Gam95], [Yac03] y [Bus07], entre muchas otras fuentes.

<sup>20</sup> Se han escrito decenas de libros acerca de las aptitudes para negociar (por ejemplo [Lew06], [Rai06] y [Fis06]). Es una de las aptitudes más importantes que pueda aprender. Lea alguno.

3. Negociación de las condiciones para ganar de los participantes a fin de reconciliarlas en un conjunto de condiciones ganar-ganar para todos los que intervienen (incluso el equipo de software).

La realización exitosa de estos pasos iniciales lleva a un resultado ganar-ganar, que se convierte en el criterio clave para avanzar hacia las siguientes actividades de la ingeniería de software.

## CASASEGURA



### *El principio de una negociación*

**La escena:** Oficina de Lisa Pérez, después de la primera reunión para recabar los requerimientos.

**Participantes:** Doug Miller, gerente de ingeniería de software, y Lisa Pérez, gerente de mercadotecnia.

#### **La conversación:**

**Lisa:** Pues escuché que la primera reunión salió realmente bien.

**Doug:** En realidad, sí. Enviste buenos representantes... contribuyeron de verdad.

**Lisa (sonríe):** Sí; en realidad me dijeron que habían entrado y que no había sido una "actividad que les despejara la cabeza".

**Doug (ríe):** La próxima vez me aseguraré de quitarme la vena tecnológica... Mira, Lisa, creo que tenemos un problema para llegar a toda esa funcionalidad del sistema de seguridad para el hogar en las fechas que propone tu dirección. Sé que aún es temprano, pero hice un poco de planeación sobre las rodillas y...

**Lisa (con el ceño fruncido):** Lo debemos tener para esa fecha, Doug. ¿De qué funcionalidad hablas?

**Doug:** Supongo que podemos tener la funcionalidad completa en la fecha establecida, pero tendríamos que retrasar el acceso por internet hasta el segundo incremento.

**Lisa:** Doug, es el acceso por internet lo que da a CasaSegura su "súper" atractivo. Toda nuestra campaña de publicidad va a girar alrededor de eso. Lo tenemos que tener...

**Doug:** Entiendo la situación, de verdad. El problema es que para dar acceso por internet tendríamos que tener un sitio web por completo seguro y en operación. Esto requiere tiempo y personal. También tenemos que elaborar mucha funcionalidad adicional en la primera entrega... no creo que podamos hacerlo con los recursos que tenemos.

**Lisa (todavía frunce el ceño):** Ya veo, pero tienes que imaginar una manera de hacerlo. Tiene importancia crítica para las funciones de seguridad del hogar y también para otras... éstas podrían esperar hasta las siguientes entregas... estoy de acuerdo con eso.

Lisa y Doug parecen estar en suspense, pero todavía deben negociar una solución a este problema. ¿Pueden "ganar" los dos en este caso? Si usted fuera el mediador, ¿qué sugeriría?

## 5.7 VALIDACIÓN DE LOS REQUERIMIENTOS

A medida que se crea cada elemento del modelo de requerimientos, se estudia para detectar inconsistencias, omisiones y ambigüedades. Los participantes asignan prioridades a los requerimientos representados por el modelo y se agrupan en paquetes de requerimientos que se implementarán como incrementos del software. La revisión del modelo de requerimientos aborda las preguntas siguientes:

- ¿Es coherente cada requerimiento con los objetivos generales del sistema o producto?
- ¿Se han especificado todos los requerimientos en el nivel apropiado de abstracción? Es decir, ¿algunos de ellos tienen un nivel de detalle técnico que resulta inapropiado en esta etapa?
- El requerimiento, ¿es realmente necesario o representa una característica agregada que tal vez no sea esencial para el objetivo del sistema?
- ¿Cada requerimiento está acotado y no es ambiguo?
- ¿Tiene atribución cada requerimiento? Es decir, ¿hay una fuente (por lo general una individual y específica) clara para cada requerimiento?
- ¿Hay requerimientos en conflicto con otros?

¿Cuando se revisan los requerimientos, qué preguntas deben plantearse?



- ¿Cada requerimiento es asequible en el ambiente técnico que albergará el sistema o producto?
- Una vez implementado cada requerimiento, ¿puede someterse a prueba?
- El modelo de requerimientos, ¿refleja de manera apropiada la información, la función y el comportamiento del sistema que se va a construir?
- ¿Se ha “particionado” el modelo de requerimientos en forma que exponga información cada vez más detallada sobre el sistema?
- ¿Se ha usado el patrón de requerimientos para simplificar el modelo de éstos? ¿Se han validado todos los patrones de manera apropiada? ¿Son consistentes todos los patrones con los requerimientos del cliente?

Éstas y otras preguntas deben plantearse y responderse para garantizar que el modelo de requerimientos es una reflexión correcta sobre las necesidades del participante y que provee un fundamento sólido para el diseño.

## 5.8 RESUMEN

Las tareas de la ingeniería de requerimientos se realizan para establecer un fundamento sólido para el diseño y la construcción. La ingeniería de requerimientos ocurre durante las actividades de comunicación y modelado que se hayan definido para el proceso general del software. Los miembros del equipo de software llevan a cabo siete funciones de ingeniería de requerimientos: concepción, indagación, elaboración, negociación, especificación, validación y administración. En la concepción del proyecto, los participantes establecen los requerimientos básicos del problema, definen las restricciones generales del proyecto, así como las características y funciones principales que debe presentar el sistema para cumplir sus objetivos. Esta información se mejora y amplía durante la indagación, actividad en la que se recaban los requerimientos y que hace uso de reuniones que lo facilitan, DFC y el desarrollo de escenarios de uso.

La elaboración amplía aún más los requerimientos en un modelo: una colección de elementos basados en escenarios, clases y comportamiento, y orientados al flujo. El modelo hace referencia a patrones de análisis: soluciones para problemas de análisis que se ha observado que son recurrentes en diferentes aplicaciones.

Conforme se identifican los requerimientos y se crea su modelo, el equipo de software y otros participantes negocian la prioridad, la disponibilidad y el costo relativo de cada requerimiento. Además, se valida cada requerimiento y su modelo como un todo comparado con las necesidades del cliente a fin de garantizar que va a construirse el sistema correcto.

## PROBLEMAS Y PUNTOS POR EVALUAR

¿Por qué muchos desarrolladores de software no ponen atención suficiente a la ingeniería de requerimientos? ¿Existen algunas circunstancias que puedan ignorarse?

El lector tiene la responsabilidad de indagar los requerimientos de un cliente que dice estar demasiado ocupado para tener una reunión. ¿Qué debe hacer?

Analice algunos de los problemas que ocurren cuando los requerimientos deben indagarse para tres o cuatro clientes distintos.

¿Por qué se dice que el modelo de requerimientos representa una fotografía instantánea del sistema en el tiempo?

Suponga que ha convencido al cliente (es usted muy buen vendedor) para que esté de acuerdo con todas las demandas que usted hace como desarrollador. ¿Eso lo convierte en un gran negociador? ¿Por qué?

Desarrolle al menos tres “preguntas libres de contexto” adicionales que podría plantear a un participante durante la concepción.

Desarrolle un “kit” para recabar requerimientos. Debe incluir un conjunto de lineamientos a fin de llevar a cabo la reunión para recabar requerimientos y los materiales que pueden emplearse para facilitar la creación de listas y otros objetos que ayuden a definir los requerimientos.

Su profesor formará grupos de cuatro a seis estudiantes. La mitad de ellos desempeñará el papel del departamento de mercadotecnia y la otra mitad adoptará el del equipo para la ingeniería de software. Su trabajo es definir los requerimientos para la función de seguridad de *CasaSegura* descrita en este capítulo. Efectúe una reunión para recabar los requerimientos con el uso de los lineamientos presentados en este capítulo.

Desarrolle un caso de uso completo para una de las actividades siguientes:

- a) Hacer un retiro de efectivo en un cajero automático.
- b) Usar su tarjeta de crédito para pagar una comida en un restaurante.
- c) Comprar acciones en la cuenta en línea de una casa de bolsa.
- d) Buscar libros (sobre un tema específico) en una librería en línea.
- e) La actividad que especifique su profesor.

¿Qué representan las “excepciones” en un caso de uso?

Describa con sus propias palabras lo que es un patrón de análisis.

Con el formato presentado en la sección 5.5.2, sugiera uno o varios patrones de análisis para los siguientes dominios de aplicación:

- a) Software de contabilidad.
- b) Software de correo electrónico.
- c) Navegadores de internet.
- d) Software de procesamiento de texto.
- e) Software para crear un sitio web.
- f) El dominio de aplicación que diga su profesor.

¿Qué significa ganar-ganar en el contexto de una negociación durante la actividad de ingeniería de los requerimientos?

¿Qué piensa que pasa cuando la validación de los requerimientos detecta un error? ¿Quién está involucrado en su corrección?

## LECTURAS ADICIONALES Y FUENTES DE INFORMACIÓN

La ingeniería de requerimientos se estudia en muchos libros debido a su importancia crítica para la creación exitosa de cualquier sistema basado en computadoras. Hood *et al.* (*Requirements Management*, Springer, 2007) analizan varios aspectos de la ingeniería de los requerimientos que incluyen tanto la ingeniería de sistemas como la de software. Young (*ne Requirements Engineering Handbook*, Artech House Publishers, 2007) presenta un análisis profundo de las tareas de la ingeniería de requerimientos. Wiegert (*More About Software Requirements*, Microsoft Press, 2006) menciona muchas técnicas prácticas para recabar y administrar los requerimientos. Hull *et al.* (*Requirements Engineering*, 2a. ed., Springer-Verlag, 2004), Bray (*An Introduction to Requirements Engineering*, Addison-Wesley, 2002), Arlow (*Requirements Engineering*, Addison-Wesley, 2001), Gilb (*Requirements Engineering*, Addison-Wesley, 2000), Graham (*Requirements Engineering and Rapid Development*, Addison-Wesley, 1999) y Sommerville y Kotonya (*Requirement Engineering: Processes and Techniques*, Wiley, 1998) son sólo algunos de los muchos libros dedicados al tema. Gottesdiener (*Requirements by Collaboration: Workshops for Defining Needs*, Addison-Wesley, 2002) proporciona una guía útil para quienes deben generar un ambiente de colaboración a fin de recabar los requerimientos con los participantes.

Lauesen (*Software Requirements: Styles and Techniques*, Addison-Wesley, 2002) presenta una recopilación exhaustiva de los métodos y notación para el análisis de requerimientos. Weigert (*Software Requirements*, Microsoft Press, 1999) y Leffingwell *et al.* (*Managing Software Requirements: A Use Case Approach*, 2a. ed., Addison-Wesley, 2003) presentan una colección útil de las mejores prácticas respecto de los requerimientos y sugieren lineamientos prácticos para la mayoría de los aspectos del proceso de su ingeniería.



En Withall (*Software Requirement Patterns*, Microsoft Press, 2007) se describe la ingeniería de requerimientos desde un punto de vista basado en los patrones. Ploesch (*Assertions, Scenarios and Prototypes*, Springer-Verlag, 2003) analiza técnicas avanzadas para desarrollar requerimientos de software. Windle y Abreo (*Software Requirements Using the Unified Process*, Prentice-Hall, 2002) estudian la ingeniería de los requerimientos en el contexto del proceso unificado y la notación UML. Alexander y Steven (*Writing Better Requirements*, Addison-Wesley, 2002) presentan un conjunto abreviado de lineamientos para escribir requerimientos claros, representarlos como escenarios y revisar el resultado final.

Es frecuente que el modelado de un caso de uso sea el detonante para crear todos los demás aspectos del modelo de análisis. El tema lo estudian mucho Rosenberg y Stephens (*Use Case Driven Object Modeling with UML: Theory and Practice*, Apress, 2007), Denny (*Succeeding with Use Cases: Working Smart to Deliver Quality*, Addison-Wesley, 2005), Alexander y Maiden (eds.) (*Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*, Wiley, 2004), Leffingwell et al. (*Managing Software Requirements: A Use Case Approach*, 2a. ed., Addison-Wesley, 2003) presentan una colección útil de las mejores prácticas sobre los requerimientos. Bitner y Spence (*Use Case Modeling*, Addison-Wesley, 2002), Cockburn [Coc01], Armour y Miller (*Advanced Use Cases Modeling: Software Systems*, Addison-Wesley, 2000) y Kulak et al. (*Use Cases: Requirements in Context*, Addison-Wesley, 2000) estudian la obtención de requerimientos con énfasis en el modelado del caso de uso.

En internet hay una variedad amplia de fuentes de información acerca de la ingeniería y análisis de los requerimientos. En el sitio web del libro, [www.mhhe.com/engcs/compsi/pressman/professional/olc/ser.htm](http://www.mhhe.com/engcs/compsi/pressman/professional/olc/ser.htm), se halla una lista actualizada de referencias en web que son relevantes para la ingeniería y análisis de los requerimientos.

## Lectura 3. Modelado de los requerimientos: flujos



CAPÍTULO

7

## MODELADO DE LOS REQUERIMIENTOS: FLUJO, COMPORTAMIENTO, PATRONES Y WEBAPPS

<b>CONCEPTOS CLAVE</b>	
diagramas de secuencia .....	168
especificación del proceso... .	163
modelado de la	
navegación .....	180
modelo de	
comportamiento .....	165
modelo de configuración.....	179
modelo del contenido .....	176
modelo de flujo de control..	162
modelo de flujo de datos ...	159
modelo de interacción 177	
modelo funcional .....	178
patrones de análisis .....	169
webapps.....	174

**D**espués de estudiar en el capítulo 6 los casos de uso, modelado de datos y modelos basados en clase, es razonable preguntar: “¿no son suficientes representaciones del modelo de los requerimientos?”

La única respuesta razonable es: “depende”.

Para ciertos tipos de software, el caso de uso puede ser la única representación para modelar los requerimientos que se necesite. Para otros, se escoge un enfoque orientado a objetos y se desarrollan modelos basados en clase. Pero en otras situaciones, los requerimientos de las aplicaciones complejas demandan el estudio de la manera como se transforman los objetos de datos cuando se mueven a través del sistema; cómo se comporta una aplicación a consecuencia de eventos externos; si el conocimiento del dominio existente puede adaptarse al problema en cuestión; o, en el caso de sistemas y aplicaciones basados en web, cómo unificar el contenido y la funcionalidad para dar al usuario final la capacidad de navegar con éxito por una *webapp* fin de lograr sus objetivos.

## UNA MIRADA RÁPIDA

**¿Qué es?** El modelo de requerimientos tiene muchas dimensiones diferentes. En este capítulo, el lector aprenderá acerca de modelos orientados al flujo, de modelos de comportamiento y

de las consideraciones especiales del análisis de requerimientos que entran en juego cuando se desarrollan *webapps*. Cada una de estas representaciones de modelado complementa los casos de uso, modelos de datos y modelos basados en clases que se estudiaron en el capítulo 6.

**¿Quién lo hace?** Un ingeniero de software (a veces llamado analista) construye el modelo con el uso de los requerimientos recabados entre varios participantes.

**¿Por qué es importante?** La perspectiva de los requerimientos del software crece en proporción directa al número de dimensiones distintas del modelado de los requerimientos. Aunque quizás no se tenga el tiempo, los recursos o la inclinación para desarrollar cada representación sugerida en este capítulo y en el anterior, debe reconocerse que cada enfoque diferente de modelado proporciona una forma distinta de ver el problema. En consecuencia, el lector (y otros participantes) estará mejor preparado para evaluar si ha especificado en forma apropiada aquello que debe lograrse.

**¿Cuáles son los pasos?** El modelado orientado al flujo da una indicación de la forma en la que las funciones de procesamiento transforman los objetos de datos. El modelado del comportamiento ilustra los estados del sistema y

sus clases, así como el efecto que tienen los eventos sobre dichos estados. El modelado basado en patrones utiliza el conocimiento del dominio existente para facilitar el análisis de los requerimientos. Los modelos de requerimientos con *webapps* están adaptados especialmente para representar requerimientos relacionados con contenido, interacción, función y configuración.

**¿Cuál es el producto final?** Para el modelado de los requerimientos, es posible escoger una gran variedad de formas basadas en texto y diagramas. Cada una de estas representaciones da una perspectiva de uno o más de los elementos del modelo.

**¿Cómo me aseguro de que lo hice bien?** Debe revisarse si los productos del trabajo del modelado de los requerimientos son correctos, completos y congruentes. Deben reflejar las necesidades de todos los participantes y establecer los fundamentos desde los que se llevará a cabo el diseño.



objetos de datos fluyen por el sistema. Un segundo enfoque del modelado de análisis, llamado *análisis orientado a objetos*, se centra en la definición de clases y en el modo en el que colaboran una con otra para cumplir con los requerimientos del cliente.

Aunque el modelo de análisis que se propone en este libro combina características de ambos enfoques, es frecuente que los equipos del software elijan uno de ellos y excluyan las representaciones del otro. La pregunta no es cuál es mejor, sino qué combinación de representaciones dará a los participantes el mejor modelo de los requerimientos del software y cuál será el mejor puente para cruzar la brecha hacia el diseño del software.

## 7.2 MODELADO ORIENTADO AL FLUJO

Aunque algunos ingenieros de software perciben el modelado orientado al flujo como una técnica obsoleta, sigue siendo una de las notaciones más usadas actualmente para hacer el análisis de los requerimientos.<sup>1</sup> Si bien el *diagrama de flujo de datos* (DFD) y la información relacionada no son una parte formal del UML, se utilizan para complementar los diagramas de éste y amplían la perspectiva de los requerimientos y del flujo del sistema.



Algunas personas afirman que los DFD son obsoletos y que no hay lugar para ellos en la práctica moderna. Ese punto de vista excluye un modo potencialmente útil de representación en el nivel del análisis. Si ayuda, use DFD.

**Cita:**  
"El propósito de los diagramas de flujo de datos es proveer un puente semántico entre los usuarios y los desarrolladores de sistemas."

Kenneth Kozar

El DFD adopta un punto de vista del tipo entrada-proceso-salida para el sistema. Es decir, los objetos de datos entran al sistema, son transformados por elementos de procesamiento y los objetos de datos que resultan de ello salen del software. Los objetos de datos se representan con flechas con leyendas y las transformaciones, con círculos (también llamados burbujas). El DFD se presenta en forma jerárquica. Es decir, el primer modelo de flujo de datos (en ocasiones llamado DFD de nivel 0 o *diagrama de contexto*) representa al sistema como un todo. Los diagramas posteriores de flujo de datos mejoran el diagrama de contexto y dan cada vez más detalles en los niveles siguientes.

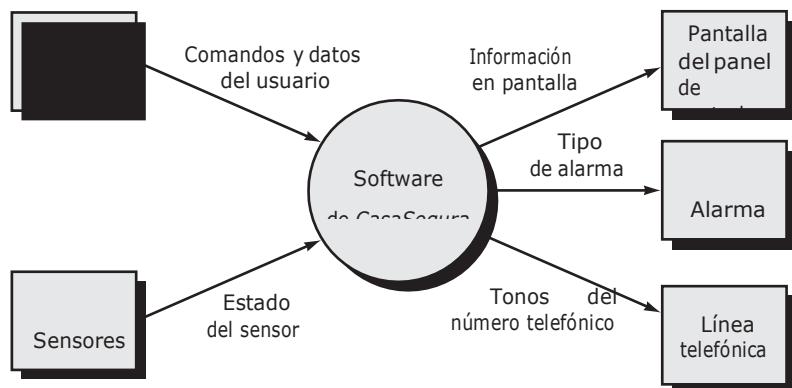
### Creación de un modelo de flujo de datos

El diagrama de flujo de datos permite desarrollar modelos del dominio de la información y del dominio funcional. A medida que el DFD se mejora con mayores niveles de detalle, se efectúa la descomposición funcional implícita del sistema. Al mismo tiempo, la mejora del DFD da como resultado el refinamiento de los datos conforme avanzan por los procesos que constituyen la aplicación.

Unos cuantos lineamientos sencillos ayudan muchísimo durante la elaboración del diagrama de flujo de los datos: 1) el nivel 0 del diagrama debe ilustrar el software o sistema como una sola

FIGURA 7.1

**DFD en el nivel de contexto para la función de seguridad de CasaSegura**



1 El modelado del flujo de datos es una actividad fundamental del *análisis estructurado*.

burbuja; 2) debe anotarse con cuidado las entradas y salidas principales; 3) la mejora debe comenzar por aislar procesos candidatos, objetos de datos y almacenamiento de éstos, para representarlos en el siguiente nivel; 4) todas las flechas y burbujas deben etiquetarse con nombres significativos; 5) de un nivel a otro, debe mantenerse la *continuidad del flujo de información*,<sup>2</sup> y 6) debe mejorarse una burbuja a la vez. Existe la tendencia natural a complicar innecesariamente el diagrama de flujo de los datos. Esto sucede cuando se trata de ilustrar demasiados detalles en una etapa muy temprana o representar aspectos de procedimiento del software en lugar del flujo de la información.

Para ilustrar el uso del DFD y la notación relacionada, consideremos de nuevo la función de seguridad de *CasaSegura*. En la figura 7.1 se muestra un DFD de nivel 0 para dicha función. Las entidades externas principales (cuadrados) producen información para uso del sistema y consumen información generada por éste. Las flechas con leyendas representan objetos de datos o jerarquías de éstos. Por ejemplo, los **comandos** y **datos del usuario** agrupan todos los comandos de configuración, todos los comandos de activación/desactivación, todas las diferentes interacciones y todos los datos que se introducen para calificar o expandir un comando.

Ahora debe expandirse el DFD de nivel 0 a un modelo de flujo de datos de nivel 1. Pero, ¿cómo hacerlo? Según el enfoque sugerido en el capítulo 6, debe aplicarse un “análisis gramatical” [Abb83] a la narración del caso de uso que describe la burbuja en el nivel del contexto. Es decir, se aíslan todos los sustantivos (y frases sustantivadas) y verbos (y frases verbales) en la narración del procesamiento de *CasaSegura* obtenida durante la primera reunión realizada para recabar los requerimientos. Recordemos el análisis gramatical del texto que narra el procesamiento presentado en la sección 6.5.1:



*El análisis gramatical no es una prueba de todo, pero da un impulso excelente para arrancar si se tienen dificultades para definir objetos de datos y las transformaciones que operan sobre ellos.*

La función de seguridad *CasaSegura* permite que el propietario configure el sistema de seguridad cuando se instala, vigila todos los sensores conectados al sistema de seguridad e interactúa con el propietario a través de internet, una PC o un panel de control.

Durante la instalación, la PC de *CasaSegura* se utiliza para programar y configurar el sistema. Se asigna a cada sensor un número y un tipo, se programa una clave maestra para activar y desactivar el sistema, y se introducen números telefónicos para marcar cuando ocurre un evento de sensor.

Cuando se reconoce un evento de sensor, el software invoca una alarma audible instalada en el sistema. Después de un tiempo de retraso que especifica el propietario durante las actividades de configuración del sistema, el software marca un número telefónico de un servicio de monitoreo, proporciona información acerca de la ubicación y reporta la naturaleza del evento detectado. El teléfono vuelve a marcarse cada 20 segundos hasta que se obtiene la conexión telefónica.

El propietario recibe información de seguridad a través de un panel de control, la PC o un navegador, lo que en conjunto se llama interfaz. La interfaz despliega en el panel de control, en la PC o en la ventana del navegador mensajes de aviso e información del estado del sistema. La interacción del propietario tiene la siguiente forma...



*Asegúrese de que la narración del procesamiento que se va a analizar gramaticalmente está escrita con el mismo nivel de abstracción.*

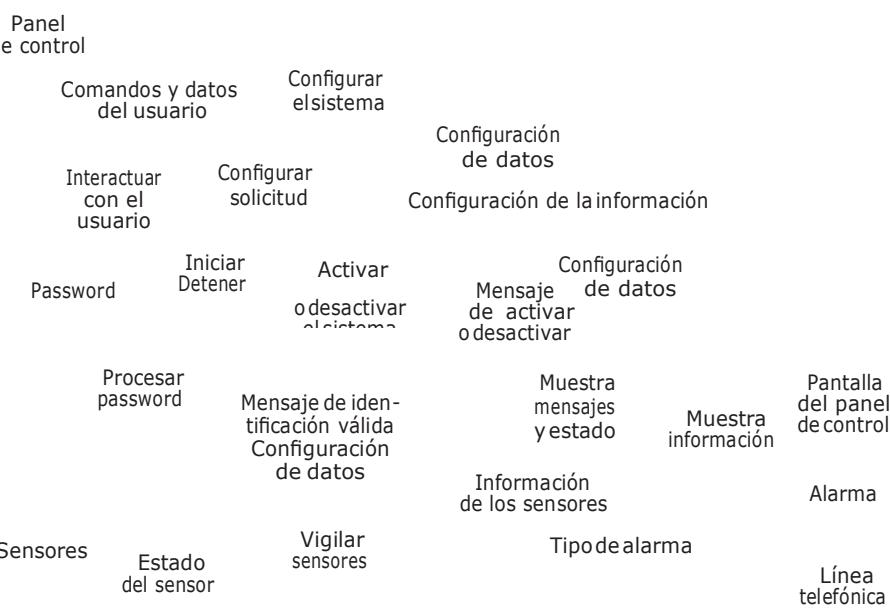
En relación con el análisis gramatical, los verbos son los procesos de *CasaSegura* y se representarán como burbujas en un DFD posterior. Los sustantivos son entidades externas (cuadros), datos u objetos de control (flechas) o almacenamiento de datos (líneas dobles). De lo estudiado en el capítulo 6 se recuerda que los sustantivos y verbos se asocian entre sí (por ejemplo, a cada sensor se asigna un número y tipo; entonces, número y tipo son atributos del objeto de datos **sensor**). De modo que al realizar un análisis gramatical de la narración de procesamiento en cualquier nivel del DFD, se genera mucha información útil sobre la manera de proceder para la mejora del nivel siguiente. En la figura 7.2 se presenta un DFD de nivel 1 con el empleo de esta información. El proceso en el nivel de contexto que se ilustra en la figura 7.1 ha sido expandido

<sup>2</sup> Es decir, los objetos de datos que entran al sistema o a cualquier transformación en cierto nivel deben ser los mismos objetos de datos (o sus partes constitutivas) que entran a la transformación en un nivel mejorado.



FIGURA 7.2

**DFD de nivel 1 para la función de seguridad de CasaSegura**



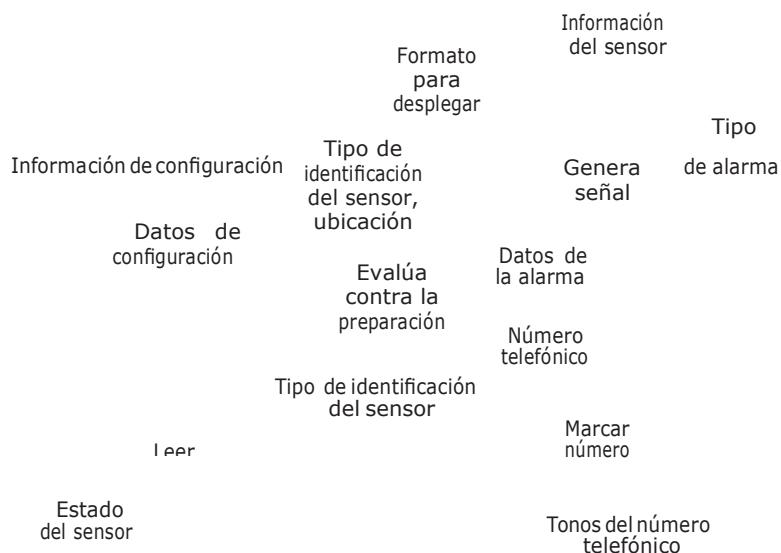
a seis procesos derivados del estudio del análisis gramatical. De manera similar, el flujo de información entre procesos del nivel 1 ha surgido de dicho análisis. Además, entre los niveles 0 y 1 se mantiene la continuidad del flujo de información.

Los procesos representados en el nivel 1 del DFD pueden mejorarse más hacia niveles inferiores. Por ejemplo, el proceso *vigilar sensores* se mejora en el DFD de nivel 2, como se aprecia en la figura 7.3. De nuevo, observe que entre los niveles se ha mantenido la continuidad del flujo de información.

La mejoría de los DFD continúa hasta que cada burbuja realiza una función simple. Es decir, hasta que el proceso representado por la burbuja ejecuta una función que se implementaría fácilmente como componente de un programa. En el capítulo 8 se estudia un concepto llamado

FIGURA 7.3

**DFD de nivel 2 que mejora el proceso *vigilar sensores***



*cohesión*, que se utiliza para evaluar el objeto del procesamiento de una función dada. Por ahora, se trata de mejorar los DFD hasta que cada burbuja tenga “un solo pensamiento”.

### Creación de un modelo de flujo de control

Para ciertos tipos de aplicaciones, el modelo de datos y el diagrama de flujo de datos es todo lo que se necesita para obtener una visión significativa de los requerimientos del software. Sin embargo, como ya se dijo, un gran número de aplicaciones son “motivadas” por eventos y no por datos, producen información de control en lugar de reportes o pantallas, y procesan información con mucha atención en el tiempo y el desempeño. Tales aplicaciones requieren el uso del *modelado del flujo de control*, además de modelar el flujo de datos.

Se dijo que un evento o aspecto del control se implementa como valor booleano (por ejemplo, verdadero o falso, encendido o apagado, 1 o 0) o como una lista discreta de condiciones (vacío, bloqueado, lleno, etc.). Se sugieren los lineamientos siguientes para seleccionar eventos candidatos potenciales:



¿Cómo seleccionar los eventos potenciales para un diagrama de flujo de control, de estado o CSPEC?

- Enlistar todos los sensores que son “leídos” por el software.
- Enlistar todas las condiciones de interrupción.
- Enlistar todos los “interruptores” que son activados por un operador.
- Enlistar todas las condiciones de los datos.
- Revisar todos los “aspectos de control” como posibles entradas o salidas de especificación del control, según el análisis gramatical de sustantivos y verbos que se aplicó a la narración del procesamiento.
- Describir el comportamiento de un sistema con la identificación de sus estados, identificar cómo se llega a cada estado y definir las transiciones entre estados.
- Centrarse en las posibles omisiones, error muy común al especificar el control; por ejemplo, se debe preguntar: “¿hay otro modo de llegar a este estado o de salir de él?”

Entre los muchos eventos y aspectos del control que forman parte del software de *CasaSegura*, se encuentran **evento de sensor** (por ejemplo, un sensor se descompone), **bandera de cambio** (señal para que la pantalla cambie) e **interruptor iniciar/detener** (señal para encender o apagar el sistema).

### La especificación de control

Una *especificación de control* (CSPEC) representa de dos maneras distintas el comportamiento del sistema (en el nivel desde el que se hizo referencia a él).<sup>3</sup> La CSPEC contiene un diagrama de estado que es una especificación secuencial del comportamiento. También puede contener una tabla de activación del programa, especificación combinatoria del comportamiento.

La figura 7.4 ilustra un diagrama de estado preliminar<sup>4</sup> para el nivel 1 del modelo de flujo de control para *CasaSegura*. El diagrama indica cómo responde el sistema a eventos conforme pasa por los cuatro estados definidos en este nivel. Con la revisión del diagrama de estado se determina el comportamiento del sistema, y, lo que es más importante, se investiga si existen “agujeros” en el comportamiento especificado.

Por ejemplo, el diagrama de estado (véase la figura 7.4) indica que las transiciones del estado **Ocioso** ocurren si el sistema se reinicia, se activa o se apaga. Si el sistema se activa (por ejem-

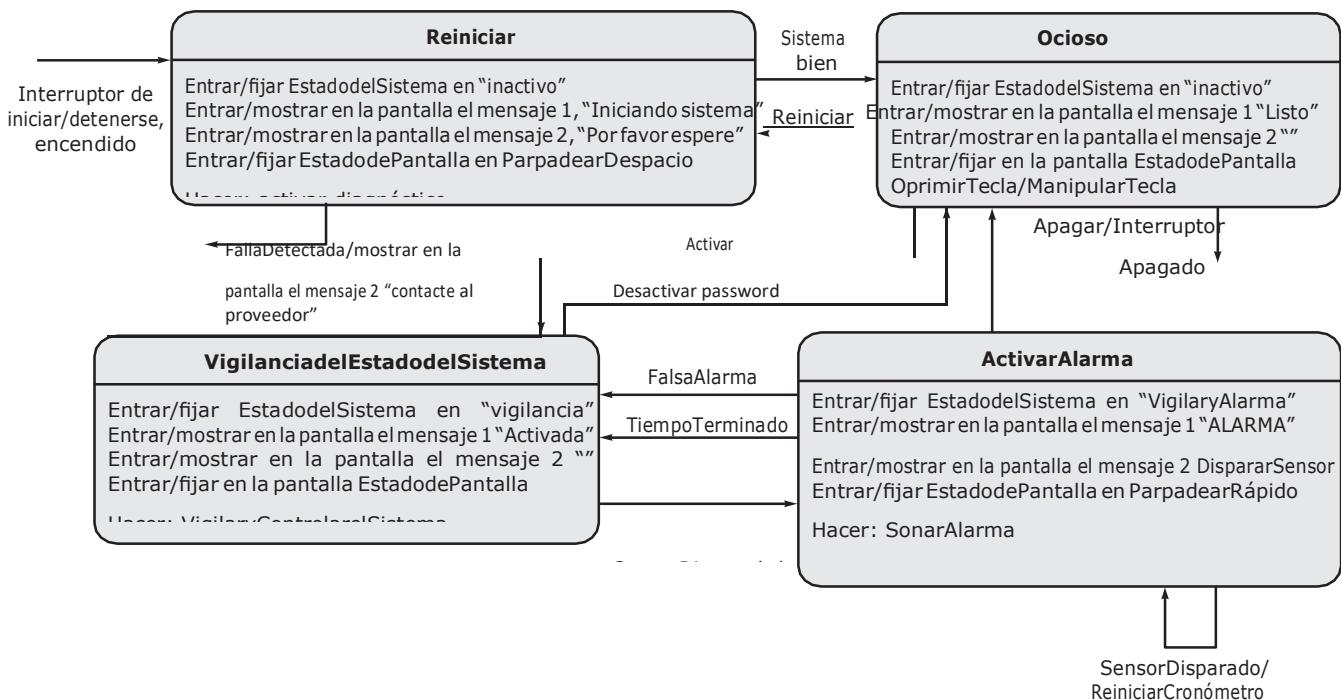
3 En la sección 7.3 se presenta notación adicional de modelado por comportamiento.

4 La notación del diagrama de estado que se emplea aquí sigue la del UML. En el análisis estructurado se dispone de un “diagrama de transición de estado”, pero el formato UML es mejor en contenido y representación de la información.



FIGURA 7.4

Diagrama de estado para la función de seguridad de CasaSegura



plo, se enciende el sistema de alarma), ocurre una transición al estado **Vigilancia del Estado del Sistema**, los mensajes en la pantalla cambian como se muestra y se invoca el proceso *Sistema de Vigilancia y Control*. Fuera del estado **Sistema de Vigilancia y Control** ocurren dos transiciones: 1) cuando se desactiva el sistema hay una transición de regreso al estado **Ocioso**; 2) cuando se dispara un sensor en el estado **Activar Alarma**. Durante la revisión se consideran todas las transiciones y el contenido de todos los estados.

La tabla de activación del proceso (TAP) es un modo algo distinto de representar el comportamiento. La TAP representa la información contenida en el diagrama de estado en el contexto de los procesos, no de los estados. Es decir, la TAP indica cuáles procesos (burbujas) serán invocados en el modelo del flujo cuando ocurra un evento. La TAP se usa como guía para un diseñador que debe construir una ejecución que controle los procesos representados en este nivel. En la figura 7.5 se aprecia una TAP para el nivel 1 del modelo de flujo del software de *CasaSegura*.

La CSPEC describe el comportamiento del sistema, pero no da información acerca del funcionamiento interno de los procesos que se activan como resultado de dicho comportamiento. En la sección 7.2.4 se estudia la notación de modelación que da esta información.

### La especificación del proceso

La *especificación del proceso* (PSPEC) se utiliza para describir todos los procesos del modelo del flujo que aparecen en el nivel final de la mejora. El contenido de la especificación del proceso incluye el texto narrativo, una descripción del lenguaje de diseño del programa<sup>5</sup> del algoritmo del proceso, ecuaciones matemáticas, tablas o diagramas de actividad UML. Si se da una PSPEC

<sup>5</sup> El lenguaje de diseño del programa (LDP) mezcla la sintaxis del lenguaje de programación con el texto narrativo a fin de dar detalles del diseño del procedimiento. En el capítulo 10 se analiza el LDP.

FIGURA 7.5

**Tabla de activación del proceso para la función de seguridad de CasaSegura**



## CLAVE

La PSPEC es una “miniespecificación” de cada transformación en el nivel más bajo de mejora de un DFD.

que acompañe a cada burbuja del modelo del flujo, se crea una “miniespecificación” que sirve como guía para diseñar la componente del software que implementará la burbuja.

Para ilustrar el uso de la PSPEC, considere la transformación *procesar password* representada en el modelo de flujo de la figura 7.2. La PSPEC de esta función adopta la forma siguiente:

**PSPEC: procesar password (en el panel de control).** La transformación *procesar password* realiza la validación en el panel de control para la función de seguridad de *CasaSegura*. *Procesar password* recibe un password de cuatro dígitos de la función *interactuar con usuario*. Primero, el password se compara con el password maestro almacenado dentro del sistema. Si el password maestro coincide, se pasa <mensaje de identificación válida = verdadero> a la función *mostrar mensaje y estado*. Si el

### CASA SEGURA Modelado del flujo de datos



**La escena:** Cubículo de Jamie, después de que terminó la reunión para recabar los requerimientos.

**Ed:** Parece que pudiéramos convertir cada burbuja en un componente ejecutable... al menos en el nivel más bajo del DFD.

**Participantes:** Jamie, Vinod y Ed, miembros del equipo de ingeniería de software de *CasaSegura*.

**Jamie:** Ésa es la mejor parte, sí se puede. En realidad, hay una forma de traducir los DFD a una arquitectura de diseño.

#### La conversación:

(Jamie presenta a Ed y a Vinod los dibujos que hizo de los modelos que se muestran en las figuras 7.1 a 7.5.)

**Ed:** ¿En verdad?

**Jamie:** Sí, pero primero tenemos que desarrollar un modelo completo de los requerimientos, y esto no lo es.

**Jamie:** En la universidad tomé un curso de ingeniería de software y aprendí esto. El profesor dijo que es un poco anticuado, pero, saben, me ayuda a aclarar las cosas.

**Vinod:** Bueno, es un primer paso, pero vamos a tener que enfrentar los elementos basados en clases y también los aspectos de comportamiento, aunque el diagrama de estado y la TAP hacen algo de eso.

**Ed:** Está muy bien. Pero no veo ninguna clase de objetos ahí.

**Ed:** Tenemos mucho trabajo por hacer y poco tiempo.

**Jamie:** No... Esto sólo es un modelo del flujo con un poco de comportamiento ilustrado.

(Entra al cubículo Doug, el gerente de ingeniería de software.)

**Vinod:** Así que estos DFD representan la E-P-S del software, ¿o no?

**Doug:** Así que dedicaremos los siguientes días a desarrollar el modelo de los requerimientos, ¿eh?

**Ed:** ¿E-P-S?

**Jamie (con orgullo):** Ya comenzamos.

**Vinod:** Entrada-proceso-salida. En realidad, los DFD son muy intuitivos... si se observan un rato, indican cómo fluyen los objetos de datos por el sistema y cómo se transforman mientras lo hacen.

**Doug:** Qué bueno, tenemos mucho trabajo por hacer y poco tiempo.  
(Los tres ingenieros de software se miran entre sí y sonríen.)



password maestro no coincide, los cuatro dígitos se comparan con una tabla de passwords secundarios (que deben asignarse para recibir invitados o trabajadores que necesiten entrar a la casa cuando el propietario no esté presente). Si el password coincide con una entrada de la tabla, se pasa <mensaje de identificación válida = verdadero> a la función *mostrar mensaje y estado*. Si no coinciden, se pasa

<mensaje de identificación válida = falso> a la función de mostrar mensaje y estado.

Si en esta etapa se desean detalles algorítmicos adicionales, también puede incluirse una representación del lenguaje de diseño del programa (LDP) como parte de la PSPEC. Sin embargo, muchos profesionales piensan que la versión LDP debe posponerse hasta comenzar el diseño de los componentes.

## HERRAMIENTAS DE SOFTWARE



### Análisis estructurado

**Objetivo:** Las herramientas de análisis estructurado permiten que un ingeniero de software cree modelos de datos,

de flujo y de comportamiento en una forma que permite la consistencia y continuidad con facilidad para hacer la revisión, edición y ampliación. Los modelos creados con estas herramientas dan al ingeniero de software la perspectiva de la representación del análisis y lo ayudan a eliminar errores antes de que éstos se propaguen al diseño o, lo que sería peor, a la implementación.

**Mecánica:** Las herramientas de esta categoría son un "diccionario de datos", como la base de datos central para describir todos los objetos de datos. Una vez definidas las entradas del diccionario, se crean diagramas entidad-relación y se desarrollan las jerarquías de los objetos. Las características de los diagramas de flujo de datos permiten que sea fácil crear este modelo gráfico y también proveen de características para generar PSPEC y CSPEC. Asimismo, las herra-

mientas de análisis permiten que el ingeniero de software produzca modelos de comportamiento con el empleo del diagrama de estado como notación operativa.

#### Herramientas representativas.<sup>6</sup>

*MacA&D, WinA&D*, desarrolladas por software Excel ([www.excelsoftware.com](http://www.excelsoftware.com)), brinda un conjunto de herramientas de análisis y diseño sencillas y baratas para computadoras Mac y Windows.

*MetaCASE Workbench*, desarrollada por MetaCase Consulting ([www.metacase.com](http://www.metacase.com)), es una metaherramienta utilizada para definir un método de análisis o diseño (incluso análisis estructurado) y sus conceptos, reglas, notaciones y generadores.

*System Architect*, desarrollado por Popkin Software ([www.popkin.com](http://www.popkin.com)), da una amplia variedad de herramientas de análisis y diseño, incluso para modelar datos y hacer análisis estructurado.

## 7.3 CREACIÓN DE UN MODELO DE COMPORTAMIENTO



¿Cómo se modela la reacción del software ante algún evento externo?

La notación de modelado que hemos estudiado hasta el momento representa elementos estáticos del modelo de requerimientos. Es hora de hacer la transición al comportamiento dinámico del sistema o producto. Para hacerlo, dicho comportamiento se representa como función de eventos y tiempo específicos.

El *modelo de comportamiento* indica la forma en la que responderá el software a eventos o estímulos externos. Para generar el modelo deben seguirse los pasos siguientes:

1. Evaluar todos los casos de uso para entender por completo la secuencia de interacción dentro del sistema.
2. Identificar los eventos que conducen la secuencia de interacción y que entienden el modo en el que éstos se relacionan con objetos específicos.
3. Crear una secuencia para cada caso de uso.
4. Construir un diagrama de estado para el sistema.
5. Revisar el modelo de comportamiento para verificar la exactitud y consistencia.

En las secciones siguientes se estudia cada uno de estos pasos.



### Identificar los eventos con el caso de uso

En el capítulo 6 se aprendió que el caso de uso representa una secuencia de actividades que involucra a los actores y al sistema. En general, un evento ocurre siempre que el sistema y un actor intercambian información. En la sección 7.2.3 se dijo que un evento *no* es la información que se intercambia, sino el hecho de que se intercambió la información.

Un caso de uso se estudia para efectos del intercambio de información. Para ilustrarlo, volvamos al caso de uso de una parte de la función de seguridad de *CasaSegura*.

El propietario utiliza el teclado para escribir un password de cuatro dígitos. El password se compara con el password válido guardado en el sistema. Si el password es incorrecto, el panel de control emitirá un sonido una vez y se reiniciará para recibir entradas adicionales. Si el password es correcto, el panel de control queda en espera de otras acciones.

Las partes subrayadas del escenario del caso de uso indican eventos. Debe identificarse un actor para cada evento, anotarse la información que se intercambia y enlistarse cualesquiera condiciones o restricciones.

Como ejemplo de evento común considere la frase subrayada en el caso de uso “el propietario utiliza el teclado para escribir un password de cuatro dígitos”. En el contexto del modelo de los requerimientos, el objeto **PropietariodeCasa**<sup>7</sup> transmite un evento al objeto **PaneldeControl**. El evento tal vez se llame *password introducido*. La información que se transfiere son los cuatro dígitos que constituyen el password, pero ésta no es una parte esencial del modelo de comportamiento. Es importante observar que ciertos eventos tienen un efecto explícito en el flujo del control del caso de uso, mientras que otros *no* lo tienen. Por ejemplo, el evento *password introducido* no cambia explícitamente el flujo del control del caso de uso, pero los resultados del evento *password comparado* (derivado de la interacción el “password se compara con el password válido guardado en el sistema”) tendrán un efecto explícito en el flujo de información y control del software *CasaSegura*.

Una vez identificados todos los eventos, se asignan a los objetos involucrados. Los objetos son responsables de la generación de eventos (por ejemplo, **Propietario** genera el evento *password introducido*) o de reconocer los eventos que hayan ocurrido en cualquier lugar (**PaneldeControl** reconoce el resultado binario del evento *password comparado*).

### Representaciones de estado

En el contexto del modelado del comportamiento deben considerarse dos caracterizaciones diferentes de los estados: 1) el estado de cada clase cuando el sistema ejecuta su función y 2) el estado del sistema según se observa desde el exterior cuando realiza su función.<sup>8</sup>

El estado de una clase tiene características tanto pasivas como activas [Cha93]. Un *estado pasivo* es sencillamente el estado actual de todos los atributos de un objeto. Por ejemplo, el estado pasivo de la clase **Jugador** (en la aplicación de juego de video que se vio en el capítulo 6) incluiría los atributos actuales **posición** y **orientación** de **Jugador**, así como otras características de éste que sean relevantes para el juego (por ejemplo, un atributo que incluya **deseos mágicos restantes**). El *estado activo* de un objeto indica el estado actual del objeto conforme pasa por una transformación o procesamiento continuos. La clase **Jugador** tal vez tenga los siguientes estados activos: *moverse, en descanso, herido, en curación, atrapado, perdido, etc.* Debe ocurrir un evento (en ocasiones llamado *disparador o trigger*) para forzar a un objeto a hacer una transición de un estado activo a otro.

<sup>7</sup> En este ejemplo se supone que cada usuario (propietario) que interactúe con *CasaSegura* tiene un password de identificación, por lo que es un objeto legítimo.

<sup>8</sup> Los diagramas de estado presentados en el capítulo 6 y en la sección 7.3.2 ilustran el estado del sistema. En esta sección, nuestro análisis se centrará en el estado de cada clase dentro del modelo del análisis.

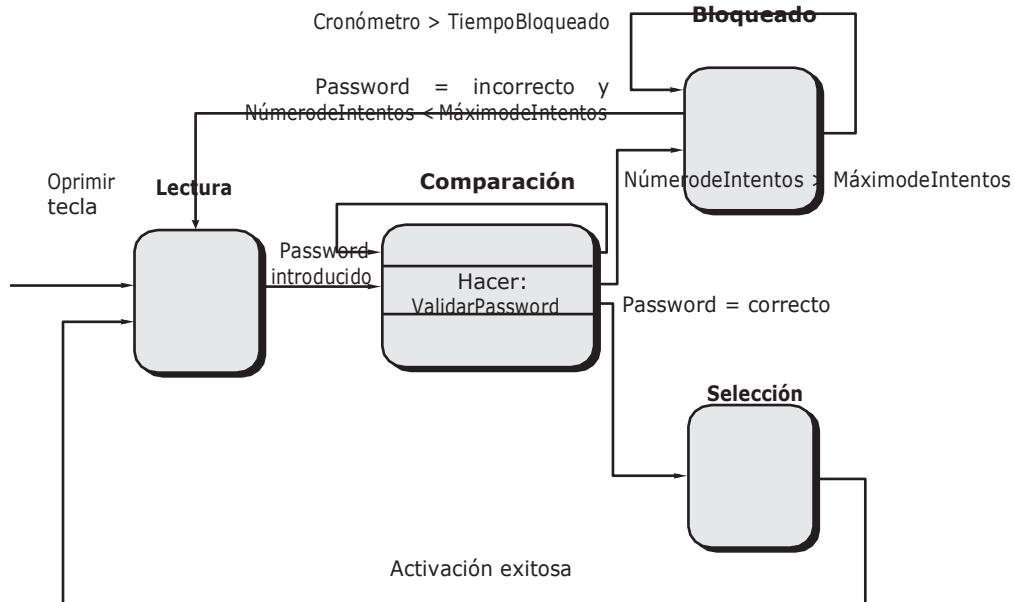
CLAVE

El sistema tiene estados que representan un comportamiento específico observable desde el exterior; una clase tiene estados que representan su comportamiento cuando el sistema realiza sus funciones.

**FIGURA 7.6**  
Diagrama  
de estado  
para la clase

**PaneldeControl**

Cronómetro ≤ TiempoBloqueado



En los párrafos que siguen se analizan dos representaciones distintas del comportamiento. La primera indica la manera en la que una clase individual cambia su estado con base en eventos externos, y la segunda muestra el comportamiento del software como una función del tiempo.

**Diagramas de estado para clases de análisis.** Un componente de modelo de comportamiento es un diagrama de estado UML<sup>9</sup> que representa estados activos para cada clase y los eventos (disparadores) que causan cambios en dichos estados activos. La figura 7.6 ilustra un diagrama de estado para el objeto **PaneldeControl** en la función de seguridad de *CasaSegura*. Cada flecha que aparece en la figura 7.6 representa una transición de un estado activo de un objeto a otro. Las leyendas en cada flecha representan el evento que dispara la transición. Aunque el modelo de estado activo da una perspectiva útil de la “historia de la vida” de un objeto, es posible especificar información adicional para llegar a más profundidad en la comprensión del comportamiento de un objeto. Además de especificar el evento que hace que la transición ocurra, puede especificarse una guardia y una acción [Cha93]. Una *guardia* es una condición booleana que debe satisfacerse para que tenga lugar una transición. Por ejemplo, la guardia para la transición del estado de “lectura” al de “comparación” en la figura 7.6 se determina con el análisis del caso de uso:

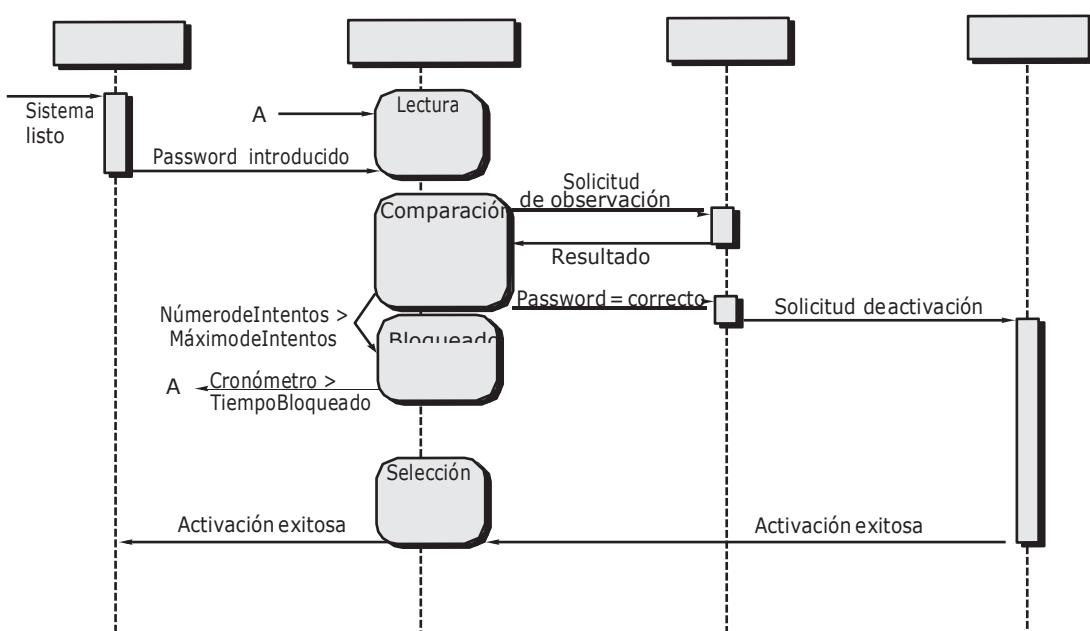
Si (password de entrada = 4 dígitos) entonces comparar con el password guardado

En general, la guardia para una transición depende del valor de uno o más atributos de un objeto. En otras palabras, depende del estado pasivo del objeto.

Una *acción* sucede en forma concurrente con la transición de estado o como consecuencia de ella, y por lo general involucra una o más operaciones (responsabilidades) del objeto. Por



**FIGURA 7.7**  
**Diagrama de secuencia (parcial) para la función CasaSegura**



ejemplo, la acción conectada con el evento *password introducido* (véase la figura 7.6) es una operación llamada *ValidarPassword()* que accede a un objeto **password** y realiza una comparación dígito por dígito para validar el password introducido.

**Diagramas de secuencia.** El segundo tipo de representación del comportamiento, llamado *diagrama de secuencia* en UML, indica la forma en la que los eventos provocan transiciones de un objeto a otro. Una vez identificados los objetos por medio del análisis del caso de uso, el modelador crea un diagrama de secuencia: representación del modo en el que los eventos causan el flujo de uno a otro como función del tiempo. En esencia, el diagrama de secuencia es una versión taquigráfica del caso de uso. Representa las clases **password** y los eventos que hacen que el comportamiento avance de una clase a otra.

La figura 7.7 ilustra un diagrama parcial de secuencia para la función de seguridad de *CasaSegura*. Cada flecha representa un evento (derivado de un caso de uso) e indica la forma en la que éste canaliza el comportamiento entre los objetos de *CasaSegura*. El tiempo se mide en la dirección vertical (hacia abajo) y los rectángulos verticales angostos representan el que toma el procesamiento de una actividad. Los estados se presentan a lo largo de una línea de tiempo vertical.

El primer evento, *sistema listo*, se deriva del ambiente externo y canaliza el comportamiento al objeto **Propietario**. El propietario introduce un password. El evento *solicitud de observación* pasa al **Sistema**, que observa el password en una base de datos sencilla y devuelve un *resultado* (*encontrada o no encontrada*) a **PaneldeControl** (ahora en el estado de *comparación*). Un password válido da como resultado el evento *password = correcto* hacia **Sistema**, lo que activa a **Sensores** con un evento de *solicitud de activación*. Por último, el control pasa de nuevo al propietario con el evento *activación exitosa*.

Una vez que se ha desarrollado un diagrama de secuencia completo, todos los eventos que causan transiciones entre objetos del sistema se recopilan en un conjunto de eventos de entrada y de salida (desde un objeto). Esta información es útil en la generación de un diseño eficaz para el sistema que se va a construir.



## CLAVE

A diferencia de un diagrama de estado que representa el comportamiento sin fijarse en las clases involucradas, un diagrama de secuencia representa el comportamiento, describiendo la forma en la que las clases pasan de un estado a otro.

**HERRAMIENTAS DE SOFTWARE****Modelación de análisis generalizado en UML**

**Objetivo:** Las herramientas de modelado del análisis dan la capacidad de desarrollar modelos basados en el escenario, en la clase y en el comportamiento con el uso de notación UML.

**Mecánica:** Las herramientas en esta categoría dan apoyo a toda la variedad de diagramas UML requeridos para construir un modelo del análisis (estas herramientas también apoyan el modelado del diseño). Además de los diagramas, las herramientas en esta categoría 1) hacen revisiones respecto de la consistencia y corrección para todos los diagramas UML, 2) proveen vínculos para producir el diseño y generar el código, y 3) construyen una base de datos que permite administrar y evaluar modelos UML grandes requeridos en sistemas complejos.

**Herramientas representativas:<sup>10</sup>**

Las herramientas siguientes apoyan toda la variedad de diagramas UML que se requieren para modelar el análisis:

*ArgoUML* es una herramienta de fuente abierta disponible en [argouml.tigris.org](http://argouml.tigris.org).

*Enterprise Architect*, desarrollada por Sparx Systems ([www.sparxsystems.com.au](http://www.sparxsystems.com.au)).

*PowerDesigner*, desarrollada por Sybase ([www.sybase.com](http://www.sybase.com)).  
*Rational Rose*, desarrollada por IBM (Rational) ([www01.ibm.com/software/rational/](http://www01.ibm.com/software/rational/)).

*System Architect*, desarrollada por Popkin Software ([www.popkin.com](http://www.popkin.com)).

*UML Studio*, desarrollada por Pragsoft Corporation ([www.pragsoft.com](http://www.pragsoft.com)).

*Visio*, desarrollada por Microsoft ([www.microsoft.com](http://www.microsoft.com)).

*Visual UML*, desarrollada por Visual Object Modelers ([www.visualuml.com](http://www.visualuml.com)).

**7.4 PATRONES PARA EL MODELADO DE REQUERIMIENTOS**

Los patrones de software son un mecanismo para capturar conocimiento del dominio, en forma que permita que vuelva a aplicarse cuando se encuentre un problema nuevo. En ciertos casos, el conocimiento del dominio se aplica a un nuevo problema dentro del mismo dominio de la aplicación. En otros, el conocimiento del dominio capturado por un patrón puede aplicarse por analogía a otro dominio de una aplicación diferente por completo.

El autor original de un patrón de análisis no “crea” el patrón, sino que lo descubre a medida que se realiza el trabajo de ingeniería de requerimientos. Una vez descubierto el patrón, se documenta describiendo “explícitamente el problema general al que es aplicable el patrón, la solución prescrita, las suposiciones y restricciones del uso del patrón en la práctica y, con frecuencia, alguna otra información sobre éste, como la motivación y las fuerzas que impulsan el empleo del patrón, el análisis de las ventajas y desventajas del mismo y referencias a algunos ejemplos conocidos de su empleo en aplicaciones prácticas” [Dev01].

En el capítulo 5 se presentó el concepto de patrones de análisis y se indicó que éstos representan una solución que con frecuencia incorpora una clase, función o comportamiento dentro del dominio de aplicación. El patrón vuelve a utilizarse cuando se hace el modelado de los requerimientos para una aplicación dentro del dominio.<sup>11</sup> Los patrones de análisis se guardan en un depósito para que los miembros del equipo de software usen herramientas de búsqueda para encontrarlos y volverlos a emplear. Una vez seleccionado un patrón apropiado, se integra en el modelo de requerimientos, haciendo referencia a su nombre.

**Descubrimiento de patrones de análisis**

El modelo de requerimientos está formado por una amplia variedad de elementos: basados en el escenario (casos de uso), orientados a datos (el modelo de datos), basados en clases, orientados al flujo y del comportamiento. Cada uno de estos elementos estudia el problema desde





una perspectiva diferente y da la oportunidad de descubrir patrones que tal vez suceden en un dominio de aplicación o por analogía en distintos dominios de aplicación.

El elemento más fundamental en la descripción de un modelo de requerimientos es el caso de uso. En el contexto de este análisis, un conjunto coherente de casos de uso sirve como base para descubrir uno o más patrones de análisis. Un *patrón de análisis semántico* (PAS) “es un patrón que describe un conjunto pequeño de casos de uso coherentes que describen a su vez una aplicación general” [Fer00].

Considere el siguiente caso de uso preliminar para el software que se requiere a fin de controlar y vigilar una cámara de visión real y un sensor de proximidad para un automóvil:

#### **Caso de uso: Vigilar el movimiento en reversa**

**Descripción:** Cuando se coloca el vehículo en *reversa*, el software de control permite que se transmita un video a una pantalla que está en el tablero, desde una cámara colocada en la parte posterior. El software superpone varias líneas de orientación y distancia en la pantalla a fin de que el operador del auto mantenga la orientación cuando éste se mueve en reversa. El software de control también vigila un sensor de proximidad con el fin de determinar si un objeto se encuentra dentro de una distancia de 10 pies desde la parte trasera del carro. Esto frenará al vehículo de manera automática si el sensor de proximidad indica que hay un objeto a  $x$  pies de la defensa trasera, donde  $x$  se determina con base en la velocidad del automóvil.

Este caso de uso implica varias funciones que se mejorarían y elaborarían (en un conjunto coherente de casos de uso) durante la reunión para recabar y modelar los requerimientos. Sin importar cuánta elaboración se logre, los casos de uso sugieren un PAS sencillo pero con amplias aplicaciones (la vigilancia y control de sensores y actuadores de un sistema físico con base en software). En este caso, los “sensores” dan información en video sobre la proximidad. El “actuador” es el sistema de frenado del vehículo (que se invoca si hay un objeto muy cerca de éste). Pero en un caso más general, se descubre un patrón de aplicación muy amplio.

En muchos dominios distintos de aplicación, se requiere software para vigilar sensores y controlar actuadores físicos. Se concluye que podría usarse mucho un patrón de análisis que describa los requerimientos generales para esta capacidad. El patrón, llamado **Actuador-Sensor**, se aplicaría como parte del modelo de requerimientos para *CasaSegura* y se analiza en la sección 7.4.2, a continuación.

#### **Ejemplo de patrón de requerimientos: Actuador-Sensor<sup>12</sup>**

Uno de los requerimientos de la función de seguridad de *CasaSegura* es la capacidad de vigilar sensores de seguridad (por ejemplo, sensores de frenado, de incendio, de humo o contenido de CO, de agua, etc.). Las extensiones basadas en internet para *CasaSegura* requerirán la capacidad de controlar el movimiento (por ejemplo, apertura, acercamiento, etc.) de una cámara de seguridad dentro de una residencia. La implicación es que el software de *CasaSegura* debe manejar varios sensores y “actuadores” (como los mecanismos de control de las cámaras).

Konrad y Cheng [Kon02] sugieren un patrón llamado **Actuador-Sensor** que da una guía útil para modelar este requerimiento dentro del software de *CasaSegura*. A continuación se presenta una versión abreviada del patrón **Actuador-Sensor**, desarrollada originalmente para aplicaciones automotrices.

#### **Nombre del patrón. Actuador-Sensor**

**Objetivo.** Especifica distintas clases de sensores y actuadores en un sistema incrustado.

**Motivación.** Por lo general, los sistemas incrustados tienen varias clases de sensores y actuadores, conectados en forma directa o indirecta con una unidad de control. Aunque muchos de

12 Esta sección se adaptó de [Kon02] con permiso de los autores.



los sensores y actuadores se ven muy distintos, su comportamiento es lo bastante similar como para estructurarlos en un patrón. Éste ilustra la forma de especificar los sensores y actuadores para un sistema, incluso los atributos y operaciones. El patrón **Actuador-Sensor** usa un mecanismo para *jalar* (solicitud explícita de información) **SensoresPasivos** y otro mecanismo para *empujar* (emisión de información) los **SensoresActivos**.

### Restricciones

- Cada sensor pasivo debe tener algún método para leer la entrada de un sensor y los atributos que representan al valor del sensor.
- Cada sensor activo debe tener capacidades para emitir mensajes actualizados cuando su valor cambie.
- Cada sensor activo debe enviar un *latido de vida*, mensaje de estado que se emite cada cierto tiempo para detectar fallas.
- Cada actuador debe tener un método para invocar la respuesta apropiada determinada por el **Cálculo de Componente**.
- Cada sensor y actuador deben tener una función implementada para revisar su propio estado de operación.
- Cada sensor y actuador debe ser capaz de someter a prueba la validez de los valores recibidos o enviados y fijar su estado de operación si los valores se encuentran fuera de las especificaciones.

**Aplicabilidad.** Es útil en cualquier sistema en el que haya varios sensores y actuadores.

**Estructura.** En la figura 7.8 se presenta un diagrama de clase UML para el patrón **Actuador-Sensor**. **Actuador**, **SensorPasivo** y **SensorActivos** son clases abstractas y están escritas con letra cursiva. En este patrón hay cuatro tipos diferentes de sensores y actuadores. Las clases **Booleano**, **Entero** y **Real** representan los tipos más comunes de sensores y actuadores. Las clases complejas de éstos son aquellas que usan valores que no se representan con facilidad en términos de tipos de datos primitivos, tales como los de un radar. No obstante, estos equipos

FIGURA 7.8

Diagrama de secuencia UML para el patrón Actuador-Sensor.

Fuente: Adaptado de [Kon02], con permiso.

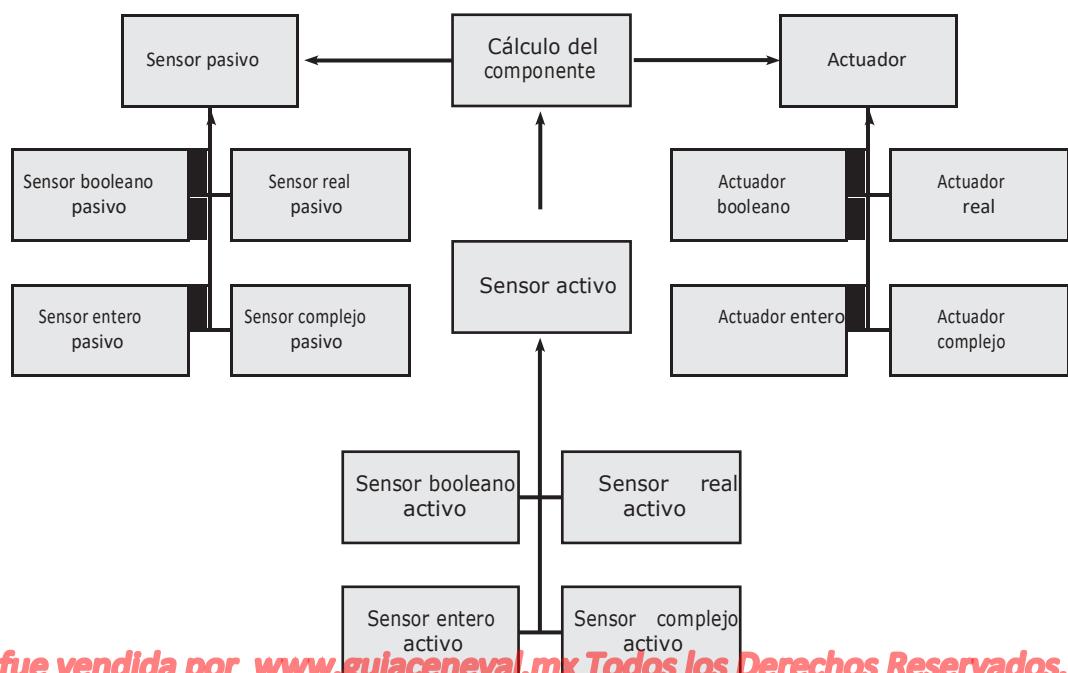
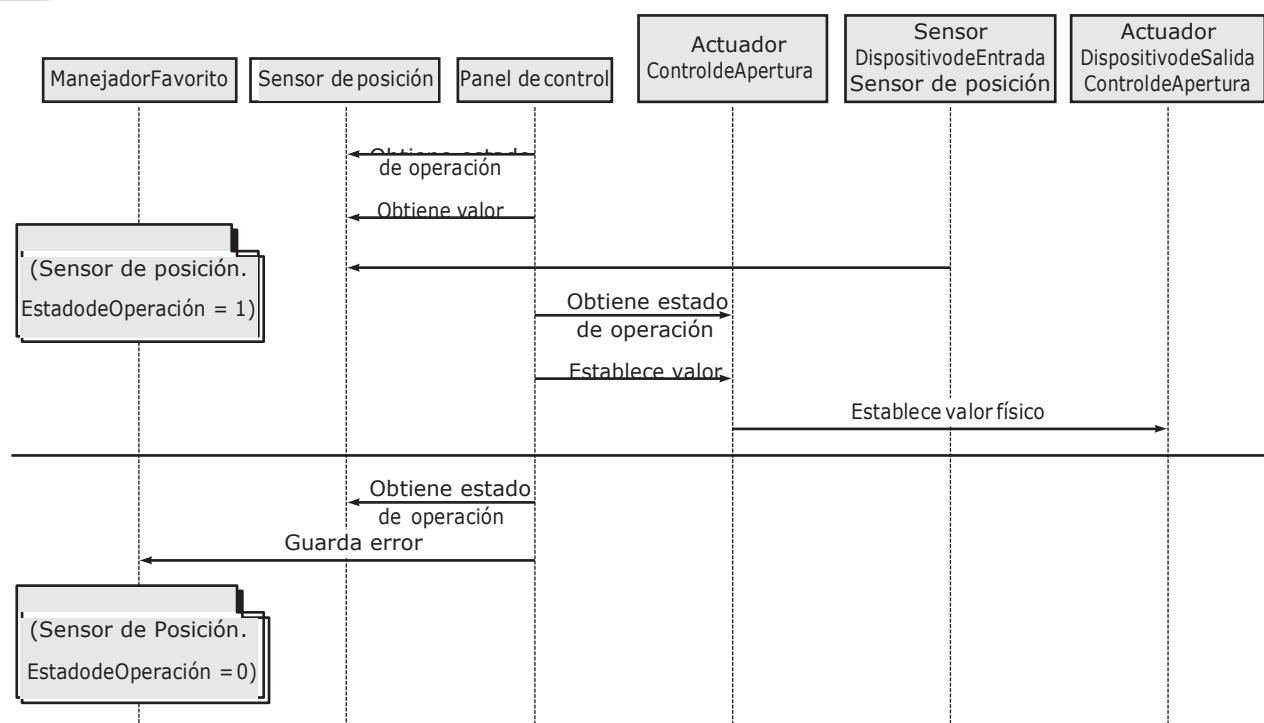


FIGURA 7.9 Diagrama de clase UML para el patrón Actuador-Sensor. Fuente: Reimpreso de [Kon02], con permiso.



deben heredar la interfaz de las clases abstractas, ya que deben tener funciones básicas, tales como consultar los estados de operación.

**Comportamiento:** La figura 7.9 presenta un diagrama de secuencia UML para un ejemplo de patrón **Actuador-Sensor** según podría aplicarse a la función de *CasaSegura* que controla el posicionamiento (como la apertura y el acercamiento) de una cámara de seguridad. Aquí, el **PaneldeControl**<sup>13</sup> consulta un sensor (uno de posición pasiva) y un actuador (control de apertura) para comprobar el estado de operación con fines de diagnóstico antes de leer o establecer un valor. Los mensajes *Establecer un Valor Físico* y *Obtener un Valor Físico* no son mensajes entre objetos. En vez de ello, describen la interacción entre los dispositivos físicos del sistema y sus contrapartes de software. En la parte inferior del diagrama, bajo la línea horizontal, el **SensordePosición** reporta que el estado de operación es igual a cero. Entonces, el **Cálculo- lodeComponente** (representado como **PaneldeControl**) envía el código de error para una falla de posición de un sensor al **ManejadordeFallas**, que decidirá cómo afecta este error al sistema y qué acciones se requieren. Obtiene los datos de los sensores y calcula la respuesta requerida por parte de los actuadores.

**Participantes.** Esta sección de la descripción de patrones “clasifica las clases u objetos incluidos en el patrón de requerimientos” [Kon02] y describe las responsabilidades de cada clase u objeto (véase la figura 7.8). A continuación se presenta una lista abreviada:

- **Resumen de SensorPasivo:** Define una interfaz para los sensores pasivos.
- **SensorBooleanoPasivo:** Define los sensores booleanos pasivos.
- **SensorEnteroPasivo:** Define los sensores enteros pasivos.

13 El patrón original usa la frase general **Cálculo de Componente**.



- **SensorRealPasivo:** Define los sensores reales pasivos.
- **Resumen de SensorActivo:** Define una interfaz para los sensores activos.
- **SensorBooleanoActivo:** Define los sensores booleanos activos.
- **SensorEnteroActivo:** Define los sensores enteros activos.
- **SensorRealActivo:** Define los sensores reales activos.
- **Resumen de actuador:** Define una interfaz para los actuadores.
- **ActuadorBooleano:** Define los actuadores booleanos.
- **ActuadorEntero:** Define los actuadores enteros.
- **ActuadorReal:** Define los actuadores reales.
- **CálculodeComponente:** Parte central del controlador; obtiene los datos de los sensores y calcula la respuesta requerida para los actuadores.
- **SensorComplejoActivo:** Los sensores complejos activos tienen la funcionalidad básica de la clase **SensorActivo**, pero es necesario especificar métodos y atributos adicionales más elaborados.
- **SensorComplejoPasivo:** Los sensores complejos pasivos tienen la funcionalidad básica de la clase abstracta **SensorPasivo**, pero se necesita especificar métodos y atributos adicionales más elaborados.
- **ActuadorComplejo:** Los actuadores complejos también tienen la funcionalidad básica de la clase abstracta **Actuador**, pero se requiere especificar métodos y atributos adicionales más elaborados.

**Colaboraciones.** Esta sección describe cómo interactúan los objetos y clases entre sí, y cómo efectúa cada uno sus responsabilidades.

- Cuando **CálculodeComponente** necesita actualizar el valor de un **SensorPasivo**, consulta a los sensores y solicita el valor enviando el mensaje apropiado.
- Los **SensoresActivos** no son consultados. Inician la transmisión de los valores del sensor a la unidad de cálculo, con el uso del método apropiado para establecer el valor en **CálculodeComponente**. Durante un tiempo especificado, envían un latido de vida al menos una vez con el fin de actualizar sus parámetros de tiempo con el reloj del sistema.
- Cuando **CálculodeComponente** necesita establecer el valor de un actuador, envía el valor a éste.
- **CálculodeComponente** consulta y establece el estado de operación de los sensores y actuadores por medio de los métodos apropiados. Si un estado de operación es cero, entonces se envía el error al **ManejadordeFallas**, clase que contiene métodos para manejar mensajes de error, tales como reiniciar un mecanismo más elaborado de recuperación o un dispositivo de respaldo. Si no es posible la recuperación, entonces el sistema sólo usa el último valor conocido para el sensor o uno preestablecido.
- Los **SensoresActivos** ofrecen métodos para agregar o retirar los evaluadores o evalúan rangos de los componentes que quieren que reciban los mensajes en caso de un cambio de valor.

#### Consecuencias

1. Las clases sensor y actuador tienen una interfaz común.
2. Sólo puede accederse a los atributos de clase a través de mensajes y la clase decide si se aceptan o no. Por ejemplo, si se establece el valor de un actuador por arriba del

máximo, entonces la clase actuador tal vez no acepte el mensaje, o quizás emplee un valor máximo preestablecido.

3. La complejidad del sistema es potencialmente reducida debido a la uniformidad de las interfaces para los actuadores y sensores.

La descripción del patrón de requerimientos también da referencias acerca de otros requerimientos y patrones de diseño relacionados.

## 7.5 MODELADO DE REQUERIMIENTOS PARA WEBAPPS <sup>14</sup>

Es frecuente que los desarrolladores de web manifiesten escepticismo cuando se plantea la idea del análisis de los requerimientos para webapps. Acostumbran decir: “después de todo, el proceso de desarrollo en web debe ser ágil y el análisis toma tiempo. Nos hará ser lentos justo cuando necesitemos diseñar y construir la *webapp*”.

El análisis de los requerimientos lleva tiempo, pero resolver el problema equivocado toma aún más tiempo. La pregunta que debe responder todo desarrollador en web es sencilla: ¿estás seguro de que entiendes los requerimientos del problema? Si la respuesta es un “sí” inequívoco, entonces tal vez sea posible omitir el modelado de los requerimientos, pero si la respuesta es “no”, entonces ésta debe llevarse a cabo.

### ¿Cuánto análisis es suficiente?

El grado en el que se profundice en el modelado de los requerimientos para las *webapps* depende de los factores siguientes:

- Tamaño y complejidad del incremento de la *webapp*.
- Número de participantes (el análisis ayuda a identificar los requerimientos conflictivos que provienen de distintas fuentes).
- Tamaño del equipo de la *webapp*.
- Grado en el que los miembros del equipo han trabajado juntos antes (el análisis ayuda a desarrollar una comprensión común del proyecto).
- Medida en la que el éxito de la organización depende directamente del éxito de la *webapp*.

El inverso de los puntos anteriores es que a medida que el proyecto se hace más chico, que el número de participantes disminuye, que el equipo de desarrollo es más cohesivo y que la aplicación es menos crítica, es razonable aplicar un enfoque más ligero para el análisis.

Aunque es una buena idea analizar el problema *antes* de que comience el diseño, no es verdad que *todo* el análisis deba preceder a *todo* el diseño. En realidad, el diseño de una parte específica de la *webapp* sólo demanda un análisis de los requerimientos que afectan a sólo esa parte de la *webapp*. Como un ejemplo proveniente de *CasaSegura*, podría diseñarse con validez la estética general del sitio web (formatos, colores, etc.) sin tener que analizar los requerimientos funcionales de las capacidades de comercio electrónico. Sólo se necesita analizar aquella parte del problema que sea relevante para el trabajo de diseño del incremento que se va a entregar.

### Entrada del modelado de los requerimientos

En el capítulo 2 se analizó una versión ágil del proceso de software general que puede aplicarse cuando se hace la ingeniería de las *webapps*. El proceso incorpora una actividad de comunica-



ción que identifica a los participantes y las categorías de usuario, el contexto del negocio, las metas definidas de información y aplicación, requerimientos generales de *webapps* y los escenarios de uso, información que se convierte en la entrada del modelado de los requerimientos. Esta información se representa en forma de descripciones hechas en lenguaje natural, a grandes rasgos, en bosquejos y otras representaciones no formales.

El análisis toma esta información, la estructura con el empleo de un esquema de representación definido formalmente (donde sea apropiado) y luego produce como salida modelos más rigurosos. El modelo de requerimientos brinda una indicación detallada de la verdadera estructura del problema y da una perspectiva de la forma de la solución.

En el capítulo 6 se introdujo la función **AVC-MVC** (vigilancia con cámaras). En ese momento, esta función parecía relativamente clara y se describió con cierto detalle como parte del caso de uso (véase la sección 6.2.1). Sin embargo, la revisión del caso de uso quizás revele información oculta, ambigua o poco clara.

Algunos aspectos de esta información faltante emergerían de manera natural durante el diseño. Los ejemplos quizás incluyan el formato específico de los botones de función, su aspecto y percepción estética, el tamaño de las vistas instantáneas, la colocación del ángulo de las cámaras y el plano de la casa, o incluso minucias tales como las longitudes máxima y mínima de las claves. Algunos de estos aspectos son decisiones de diseño (como el aspecto de los botones) y otros son requerimientos (como la longitud de las claves) que no influyen de manera fundamental en los primeros trabajos de diseño.

Pero cierta información faltante sí podría influir en el diseño general y se relaciona más con la comprensión real de los requerimientos. Por ejemplo:

P<sub>1</sub>: ¿Cuál es la resolución del video de salida que dan las cámaras de *CasaSegura*?

P<sub>2</sub>: ¿Qué ocurre si se encuentra una condición de alarma mientras la cámara está siendo vigilada?

P<sub>3</sub>: ¿Cómo maneja el sistema las cámaras con vistas panorámicas y de acercamiento?

P<sub>4</sub>: ¿Qué información debe darse junto con la vista de la cámara (por ejemplo, ubicación, fecha y hora, último acceso, etcétera)?

Ninguna de estas preguntas fue identificada o considerada en el desarrollo inicial del caso de uso; no obstante, las respuestas podrían tener un efecto significativo en los diferentes aspectos del diseño.

Por tanto, es razonable concluir que aunque la actividad de comunicación provea un buen fundamento para entender, el análisis de los requerimientos mejora este entendimiento al dar una interpretación adicional. Como la estructura del problema se delineea como parte del modelo de requerimientos, invariablemente surgen preguntas. Son éstas las que llenan los huecos y, en ciertos casos, en realidad ayudan a encontrarlos.

En resumen, la información obtenida durante la actividad de comunicación será la entrada del modelo de los requerimientos, cualquiera que sea, desde un correo electrónico informal hasta un proyecto detallado con escenarios de uso exhaustivos y especificaciones del producto.

### Salida del modelado de los requerimientos

El análisis de los requerimientos provee un mecanismo disciplinado para representar y evaluar el contenido y funcionamiento de las *webapp*, los modos de interacción que hallarán los usuarios y el ambiente e infraestructura en las que reside la *webapp*.

Cada una de estas características se representa como un conjunto de modelos que permiten que los requerimientos de la *webapp* sean analizados en forma estructurada. Si bien los modelos específicos dependen en gran medida de la naturaleza de la *webapp*, hay cinco clases principales de ellos:

- **Modelo de contenido:** identifica el espectro completo de contenido que dará la *webapp*. El contenido incluye datos de texto, gráficos e imágenes, video y sonido.
- **Modelo de interacción:** describe la manera en que los usuarios interactúan con la *webapp*.
- **Modelo funcional:** define las operaciones que se aplicarán al contenido de la *webapp* y describe otras funciones de procesamiento que son independientes del contenido pero necesarias para el usuario final.
- **Modelo de navegación:** define la estrategia general de navegación para la *webapp*.
- **Modelo de configuración:** describe el ambiente e infraestructura en la que reside la *webapp*.

Es posible desarrollar cada uno de estos modelos con el empleo de un esquema de representación (llamado con frecuencia “lenguaje”) que permite que su objetivo y estructura se comuniquen y evalúen con facilidad entre los miembros del equipo de ingeniería de web y otros participantes. En consecuencia, se identifica una lista de aspectos clave (como errores, omisiones, inconsistencias, sugerencias de mejora o modificaciones, puntos de aclaración, etc.) para tratar sobre ellos.

### **Modelo del contenido de las webapps**

El modelo de contenido incluye elementos estructurales que dan un punto de vista importante de los requerimientos del contenido de una *webapp*. Estos elementos estructurales agrupan los objetos del contenido y todas las clases de análisis, entidades visibles para el usuario que se crean o manipulan cuando éste interactúa con la *webapp*.<sup>15</sup>

El contenido puede desarrollarse antes de la implementación de la *webapp*, mientras ésta se construye o cuando ya opera. En cualquier caso, se incorpora por referencia de navegación en la estructura general de la *webapp*. Un *objeto de contenido* es una descripción de un producto en forma de texto, un artículo que describe un evento deportivo, una fotografía tomada en éste, la respuesta de un usuario en un foro de análisis, una representación animada de un logotipo corporativo, una película corta de un discurso o una grabación en audio para una presentación con diapositivas. Los objetos de contenido pueden almacenarse como archivos separados, integrarse directamente en páginas web u obtenerse en forma dinámica de una base de datos. En otras palabras, un objeto de contenido es cualquier aspecto de información cohesiva que se presente al usuario final.

Los objetos de contenido se determinan directamente a partir de casos de uso, estudiando la descripción del escenario respecto de referencias directas e indirectas al contenido. Por ejemplo, se establece en **CasaSeguraAsegurada.com** una *webapp* que da apoyo a *CasaSegura*. Un caso de uso, *Comprar componentes seleccionados de CasaSegura*, describe el escenario que se requiere para comprar un componente de *CasaSegura* y que contiene la siguiente oración:

Podré obtener información descriptiva y de precios de cada componente del producto.

El modelo de contenido debe ser capaz de describir el objeto de contenido **Componente**. En muchas circunstancias, para definir los requerimientos para el contenido que debe diseñarse e implementarse, es suficiente una lista sencilla de los objetos de contenido, junto con la descripción breve de cada uno. Sin embargo, en ciertos casos, el modelo de contenido se beneficia de un análisis más rico que ilustre en forma gráfica las relaciones entre los objetos de contenido y la jerarquía que mantiene una *webapp*.

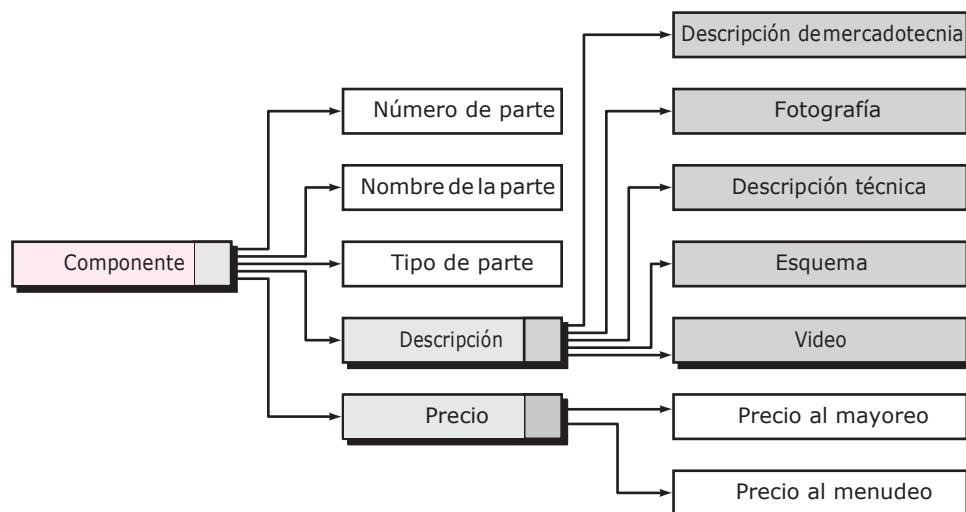
Por ejemplo, tome en cuenta el *árbol de datos* [Sri01] creado por el componente **CasaSeguraAsegurada.com** que aparece en la figura 7.10. El árbol representa una jerarquía de informa-

15 Las clases de análisis se estudiaron en el capítulo 6.



FIGURA 7.10

**Árbol de datos para el componente Casa-SeguraAsegurada.com**



ción que se utiliza para describir un componente. Los aspectos de datos simples o compuestos (uno o más valores de los datos) se representan con rectángulos sin sombra. Los objetos de contenido se representan con rectángulos con sombra. En la figura, **descripción** está definida por cinco objetos (los rectángulos sombreados). En ciertos casos, uno o más de estos objetos se mejorará más conforme se expanda el árbol de datos.

Es posible crear un árbol de datos para cualquier contenido que se componga de múltiples objetos de contenido y aspectos de datos. El árbol de datos se desarrolla como un esfuerzo para definir relaciones jerárquicas entre los objetos de contenido y para dar un medio de revisión del contenido a fin de que se descubran las omisiones e inconsistencias antes de que comience el diseño. Además, el árbol de datos sirve como base para diseñar el contenido.

### Modelo de la interacción para webapps

La gran mayoría de *webapps* permiten una “conversación” entre un usuario final y funcionalidad, contenido y comportamiento de la aplicación. Esta conversación se describe con el uso de un modelo de *interacción* que se compone de uno o más de los elementos siguientes: 1) casos de uso, 2) diagramas de secuencia, 3) diagramas de estado<sup>16</sup> y 4) prototipos de la interfaz de usuario.

En muchas instancias, basta un conjunto de casos de uso para describir la interacción en el nivel del análisis (durante el diseño se introducirán más mejoras y detalles). Sin embargo, cuando la secuencia de interacción es compleja e involucra múltiples clases de análisis o muchas tareas, es conveniente ilustrarla de forma más rigurosa mediante un diagrama.

El formato de la interfaz de usuario, el contenido que presenta, los mecanismos de interacción que implementa y la estética general de las conexiones entre el usuario y la *webapp* tienen mucho que ver con la satisfacción de éste y con el éxito conjunto del software. Aunque se afirme que la creación de un prototipo de interfaz de usuario es una actividad de diseño, es una buena idea llevarla a cabo durante la creación del modelo de análisis. Entre más pronto se revise la representación física de la interfaz de usuario, más probable es que los consumidores finales obtengan lo que desean. En el capítulo 11 se estudia con detalle el diseño de interfaces de usuario.

<sup>16</sup> Los diagramas de secuencia y los de estado se modelan con el empleo de notación UML. Los diagramas de estado se describen en la sección 7.3. Para mayores detalles, consulte el apéndice 1.

Como hay muchas herramientas para construir *webapps* baratas y poderosas en sus funciones, es mejor crear el prototipo de la interfaz con el empleo de ellas. El prototipo debe implementar los vínculos de navegación principales y representar la pantalla general en forma muy parecida a la que se construirá. Por ejemplo, si van a ponerse a disposición del usuario final cinco funciones principales del sistema, el prototipo debe representarlas tal como las verá cuando entre por primera vez a la *webapp*. ¿Se darán vínculos gráficos? ¿Dónde se desplegará el menú de navegación? ¿Qué otra información verá el usuario? Preguntas como éstas son las que debe responder el prototipo.

### Modelo funcional para las *webapps*

Muchas *webapps* proporcionan una amplia variedad de funciones de computación y manipulación que se asocian directamente con el contenido (porque lo utilizan o porque lo producen) y es frecuente que sean un objetivo importante de la interacción entre el usuario y la *webapp*. Por esta razón, deben analizarse los requerimientos funcionales y modelarlos cuando sea necesario.

El *modelo funcional* enfrenta dos elementos de procesamiento de la *webapp*, cada uno de los cuales representa un nivel distinto de abstracción del procedimiento: 1) funciones observables por los usuarios que entrega la *webapp* a éstos y 2) las operaciones contenidas en las clases de análisis que implementan comportamientos asociados con la clase.

La funcionalidad observable por el usuario agrupa cualesquiera funciones de procesamiento que inicie directamente el usuario. Por ejemplo, una *webapp* financiera tal vez implemente varias funciones de finanzas (como una calculadora de ahorros para una colegiatura universitaria o un fondo para el retiro). Estas funciones en realidad se implementan con el uso de operaciones dentro de clases de análisis, pero desde el punto de vista del usuario final; el resultado visible es la función (más correctamente, los datos que provee la función).

En un nivel más bajo de abstracción del procedimiento, el modelo de requerimientos describe el procesamiento que se realizará por medio de operaciones de clase de análisis. Estas operaciones manipulan los atributos de clase y se involucran como clases que colaboran entre sí para lograr algún comportamiento que se desea.

Sin que importe el nivel de abstracción del procedimiento, el diagrama de actividades UML se utiliza para representar detalles de éste. En el nivel de análisis, los diagramas de actividades deben usarse sólo donde la funcionalidad sea relativamente compleja. Gran parte de la complejidad de muchas *webapps* ocurre no en las funciones que proveen, sino en la naturaleza de la información a que se accede y en las formas en las que se manipula.

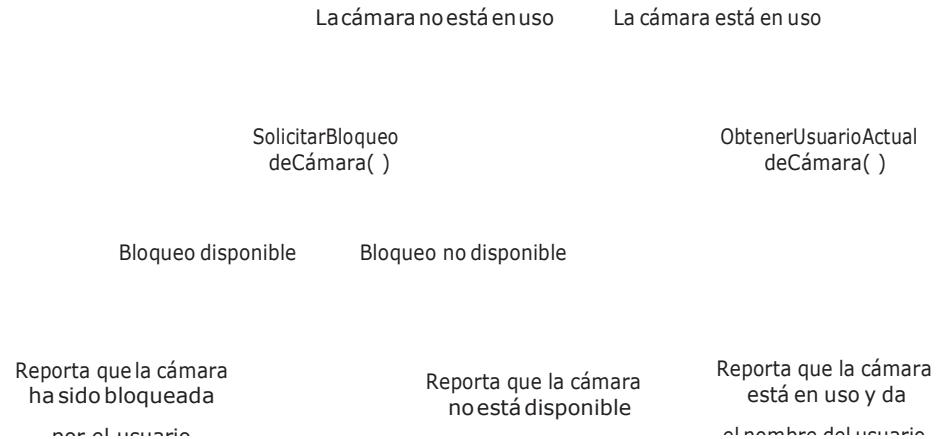
Un ejemplo de complejidad relativa de la funcionalidad para **CasaSeguraAsegurada.com** se aborda en un caso de uso llamado *Obtener recomendaciones para la distribución de sensores en mi espacio*. El usuario ya ha desarrollado la distribución del espacio que se vigilará y en este caso de uso selecciona dicha distribución y solicita ubicaciones recomendables para los sensores dentro de ella. **CasaSeguraAsegurada.com** responde con la representación gráfica de la distribución por medio de información adicional acerca de la ubicación recomendable para los sensores. La interacción es muy sencilla, el contenido es algo más complejo, pero la funcionalidad subyacente es muy sofisticada. El sistema debe realizar un análisis relativamente complejo de la planta del piso para determinar el conjunto óptimo de sensores. Debe examinar las dimensiones de la habitación, la ubicación de puertas y ventanas, y coordinar éstas con la capacidad y especificaciones de los sensores. ¡No es una tarea fácil! Para describir el procesamiento de este caso de uso se utiliza un conjunto de diagramas de actividades.

El segundo ejemplo es el caso de uso *Controlar cámaras*. En éste, la interacción es relativamente sencilla, pero existe el potencial de una funcionalidad compleja, dado que dicha operación “sencilla” requiere una comunicación compleja con dispositivos ubicados en posiciones remotas y a los que se accede por internet. Una complicación adicional se relaciona con la ne-



**FIGURA 7.11**  
**Diagrama**

**de actividades para la operación**  
*TomarControl deCámara( )*



gociación del control cuando varias personas autorizadas tratan de vigilar o controlar un mismo sensor al mismo tiempo.

La figura 7.11 ilustra el diagrama de actividades para la operación *TomarControl deCámara* que forma parte de la clase de análisis **Cámara** usada dentro del caso de uso *Controlar cámaras*. Debe observarse que con el flujo de procedimiento se invocan dos operaciones adicionales: *SolicitarBloqueodeCámera ()*, que trata de bloquear la cámara para este usuario, y *ObtenerUsuarioActualdeCámera ()*, que recupera el nombre del usuario que controla en ese momento la cámara. Los detalles de construcción indican cómo se invocan estas operaciones, y los de la interfaz para cada operación no se señalan hasta que comienza el diseño de la *webapp*.

### Modelos de configuración para las webapps

En ciertos casos, el modelo de configuración no es sino una lista de atributos del lado del servidor y del lado del cliente. Sin embargo, para *webapps* más complejas, son varias las dificultades de configuración (por ejemplo, distribuir la carga entre servidores múltiples, arquitecturas caché, bases de datos remotas, distintos servidores que atienden a varios objetos en la misma página web, etc.) que afectan el análisis y diseño. El *diagrama de despliegue UML* se utiliza en situaciones en las que deben considerarse arquitecturas de configuración compleja.

Para **CasaSeguraAsegurada.com**, deben especificarse el contenido y funcionalidad públicos a fin de que sean accesibles a través de todos los clientes principales de web (como aquéllos con 1 por ciento o más de participación en el mercado).<sup>17</sup> A la inversa, es aceptable restringir las funciones más complejas de control y vigilancia (que sólo es accesible para los usuarios tipo **Propietario**) a un conjunto más pequeño de clientes. El modelo de configuración para **CasaSeguraAsegurada.com** también especificará la operación cruzada con las bases de datos de productos y aplicaciones de vigilancia.

<sup>17</sup> La determinación de la participación en el mercado para los navegadores es notoriamente problemática y varía en función de cuál fuente se utilice. No obstante, en el momento de escribir este libro, Internet Explorer y Firefox eran los únicos que superaban 30 por ciento, y Mozilla, Opera y Safari los únicos que superaban de manera consistente 1 por ciento.

### Modelado de la navegación

Para modelar la navegación se considera cómo navegará cada categoría de usuario de un elemento de la *webapp* (como un objeto de contenido) a otro. La mecánica de navegación se define como parte del diseño. En esa etapa debe centrarse la atención en los requerimientos generales de navegación. Deben considerarse las preguntas siguientes:

- ¿Ciertos elementos deben ser más fáciles de alcanzar (requieren menos pasos de navegación) que otros? ¿Cuál es la prioridad de presentación?
- ¿Debe ponerse el énfasis en ciertos elementos para forzar a los usuarios a navegar en esa dirección?
- ¿Cómo deben manejarse los errores en la navegación?
- ¿Debe darse prioridad a la navegación hacia grupos de elementos relacionados y no hacia un elemento específico?
- ¿La navegación debe hacerse por medio de vínculos, acceso basado en búsquedas o por otros medios?
- ¿Debe presentarse a los usuarios ciertos elementos con base en el contexto de acciones de navegación previas?
- ¿Debe mantenerse un registro de usuarios de la navegación?
- ¿Debe estar disponible un mapa completo de la navegación (en oposición a un solo vínculo para “regresar” o un apuntador dirigido) en cada punto de la interacción del usuario?
- ¿El diseño de la navegación debe estar motivado por los comportamientos del usuario más comunes y esperados o por la importancia percibida de los elementos definidos de la *webapp*?
- ¿Un usuario puede “guardar” su navegación previa a través de la *webapp* para hacer expedito el uso futuro?
- ¿Para qué categoría de usuario debe diseñarse la navegación óptima?
- ¿Cómo deben manejarse los vínculos externos hacia la *webapp*? ¿Con la superposición de la ventana del navegador existente? ¿Como nueva ventana del navegador? ¿En un marco separado?

Estas preguntas y muchas otras deben plantearse y responderse como parte del análisis de la navegación.

Usted y otros participantes también deben determinar los requerimientos generales para la navegación. Por ejemplo, ¿se dará a los usuarios un “mapa del sitio” y un panorama de toda la estructura de la *webapp*? ¿Un usuario puede hacer una “visita guiada” que resalte los elementos más importantes (objetos y funciones de contenido) con que se disponga? ¿Podrá acceder un usuario a los objetos o funciones de contenido con base en atributos definidos de dichos elementos (por ejemplo, un usuario tal vez desee acceder a todas las fotografías de un edificio específico o a todas las funciones que permiten calcular el peso)?

## 7.6 RESUMEN

Los modelos orientados al flujo se centran en el flujo de objetos de datos a medida que son transformados por las funciones de procesamiento. Derivados del análisis estructurado, los modelos orientados al flujo usan el diagrama de flujo de datos, notación de modelación que ilustra la manera en la que se transforma la entrada en salida cuando los objetos de datos se mueven a través del sistema. Cada función del software que transforme datos es descrita por la



especificación o narrativa de un proceso. Además del flujo de datos, este elemento de modelado también muestra el flujo del control, representación que ilustra cómo afectan los eventos al comportamiento de un sistema.

El modelado del comportamiento ilustra el comportamiento dinámico. El modelo de comportamiento utiliza una entrada basada en el escenario, orientada al flujo y elementos basados en clases para representar los estados de las clases de análisis y al sistema como un todo. Para lograr esto, se identifican los estados y se definen los eventos que hacen que una clase (o el sistema) haga una transición de un estado a otro, así como las acciones que ocurren cuando se efectúa dicha transición. Los diagramas de estado y de secuencia son la notación que se emplea para modelar el comportamiento.

Los patrones de análisis permiten a un ingeniero de software utilizar el conocimiento del dominio existente para facilitar la creación de un modelo de requerimientos. Un patrón de análisis describe una característica o función específica del software que puede describirse con un conjunto coherente de casos de uso. Especifica el objetivo del patrón, la motivación para su uso, las restricciones que limitan éste, su aplicabilidad en distintos dominios de problemas, la estructura general del patrón, su comportamiento y colaboraciones, así como información suplementaria.

El modelado de los requerimientos para las *webapps* utiliza la mayoría, si no es que todos, los elementos de modelado que se estudian en el libro. Sin embargo, dichos elementos se aplican dentro de un conjunto de modelos especializados que se abocan al contenido, interacción, función, navegación y configuración cliente-servidor en la que reside la *webapp*.

## PROBLEMAS Y PUNTOS POR EVALUAR

¿Cuál es la diferencia fundamental entre el análisis estructurado y las estrategias orientadas a objetos para hacer el análisis de los requerimientos?

En un diagrama de flujo de datos, ¿una flecha representa un flujo del control u otra cosa?

¿Qué es la “continuidad del flujo de información” y cómo se aplica cuando se mejora el diagrama de flujo de datos?

¿Cómo se utiliza el análisis gramatical en la creación de un DFD?

¿Qué es una especificación del control?

¿Son lo mismo una PSPEC y un caso de uso? Si no es así, explique las diferencias.

Hay dos tipos diferentes de “estados” que los modelos del comportamiento pueden representar. ¿Cuáles son?

¿En qué difiere un diagrama de secuencia de un diagrama de estado? ¿En qué se parecen?

Sugiera tres patrones de requerimientos para un teléfono inalámbrico moderno y escriba una descripción breve de cada uno. ¿Estos patrones podrían usarse para otros equipos? Dé un ejemplo.

Seleccione uno de los patrones desarrollados en el problema 7.9 y desarrolle una descripción del patrón razonablemente completa, similar en contenido y estilo a la que se presentó en la sección 7.4.2.

¿Cuánto modelado del análisis piensa que se requeriría para **CasaSeguraAsegurada.com**? ¿Se necesitaría cada uno de los tipos de modelo descritos en la sección 7.5.3?

¿Cuál es el propósito del modelo de interacción para una *webapp*?

Un modelo funcional de *webapp* debe retrasarse hasta el diseño. Diga los pros y contras de este argumento.

¿Cuál es el propósito de un modelo de configuración?

¿En qué difiere el modelo de navegación del modelo de interacción?

## LECTURAS ADICIONALES Y FUENTES DE INFORMACIÓN

Se han publicado decenas de libros sobre el análisis estructurado. Todos cubren el tema de manera adecuada, pero algunos son excelentes. DeMarco y Plauger escribieron un clásico (*Structured Analysis and System Specification*, Pearson, 1985) que sigue siendo una buena introducción a la notación básica. Los libros escritos por Kendall y Kendall (*Systems Analysis and Design*, 5a. ed., Prentice-Hall, 2002), Hoffer et al. (*Modern Systems Analysis and Design*, Addison-Wesley, 3a. ed., 2001), Davis y Yen (*The Information System Consultant's Handbook: Systems Analysis and Design*, CRC Press, 1998) y Modell (*A Professional's Guide to Systems Analysis*, 2a. ed., McGraw-Hill, 1996) son buenas referencias. El escrito por Yourdon (*Modern Structured Analysis*, Yourdon-Press, 1989) sobre el tema está entre las fuentes más exhaustivas publicadas hasta la fecha.

El modelado del comportamiento presenta un punto de vista dinámico e importante del comportamiento de un sistema. Los libros de Wagner et al. (*Modeling Software with Finite State Machines: A Practical Approach*, Auerbach, 2006) y Boerger y Staerk (*Abstract State Machines*, Springer, 2003) presentan un análisis completo de los diagramas de estado y de otras representaciones del comportamiento.

La mayoría de textos escritos sobre patrones de software se centran en el diseño de éste. Sin embargo, los libros de Evans (*Domain-Driven Design*, Addison-Wesley, 2003) y Fowler ([Fow03] y [Fow97]) abordan específicamente los patrones de análisis.

Pressman y Lowe presentan un tratamiento profundo del modelado del análisis para webapps [Pre08]. Los artículos contenidos dentro de una antología editada por Murugesan y Desphande (*Web Engineering: Managing Diversity and Complexity of Web Application Development*, Springer, 2001) analizan distintos aspectos de los requerimientos para las webapps. Además, la edición anual de *Proceedings of the International Conference on Web Engineering* analiza en forma regular aspectos del modelado de los requerimientos.

En internet hay una amplia variedad de fuentes de información sobre modelado de los requerimientos. En el sitio web del libro, [www.mhhe.com/engcs/compsciprofessional/olc/ser.htm](http://www.mhhe.com/engcs/compsciprofessional/olc/ser.htm), se encuentra una lista actualizada de referencias que hay en la red mundial, relevantes para el modelado del análisis.



## **Lectura 4. Modelado de los requerimientos: escenarios**

CAPÍTULO

6

## MODELADO DE LOS REQUERIMIENTOS: ESCENARIOS, INFORMACIÓN Y CLASES DE ANÁLISIS



**SEP**

SECRETARÍA DE  
EDUCACIÓN PÚBLICA



CENTRO NACIONAL  
DE EVALUACIÓN PARA  
LA EDUCACIÓN SUPERIOR, A.C.



**EDITORIAL:**  
[guiaceneval.mx](http://guiaceneval.mx)



**E**n el nivel técnico, la ingeniería de software comienza con una serie de tareas de modelado que conducen a la especificación de los requerimientos y a la representación de un diseño del software que se va a elaborar. El modelo de requerimientos<sup>1</sup> –un conjunto de modelos, en realidad– es la primera representación técnica de un sistema.

En un libro fundamental sobre métodos para modelar los requerimientos, Tom DeMarco [DeM79] describe el proceso de la manera siguiente:

Al mirar retrospectivamente los problemas y las fallas detectados en la fase de análisis, concluyo que es necesario agregar lo siguiente al conjunto de objetivos de dicha fase. Debe ser muy fácil dar mantenimiento a los productos del análisis. Esto se aplica en particular al Documento de Objetivos [especificación de los requerimientos del software]. Los problemas grandes deben ser enfrentados con el empleo de un método eficaz para dividirlos. La especificación victoriana original resulta caduca. Deben usarse gráficas, siempre que sea posible. Es necesario diferenciar las consideraciones lógicas [esenciales] y las físicas [implementación]... Finalmente, se necesita... algo que ayude a dividir los requerimientos y a documentar dicha partición antes de elaborar la especificación... algunos medios para dar seguimiento a las interfaces y evaluarlas... nuevas herramientas para describir la lógica y la política, algo mejor que un texto narrativo.

## UNA MIRADA RÁPIDA

**¿Qué es?** La palabra escrita es un vehículo maravilloso para la comunicación, pero no necesariamente es la mejor forma de representar los requerimientos de computadora.

El modelado de los requerimientos utiliza una combinación de texto y diagramas para ilustrarlos en forma que sea relativamente fácil de entender y, más importante, de revisar para corregir, completar y hacer congruente.

**¿Quién lo hace?** Un ingeniero de software (a veces llamado “analista”) construye el modelo con el uso de los requerimientos recabados del cliente.

**¿Por qué es importante?** Para validar los requerimientos del software se necesita estudiarlos desde varios puntos de vista diferentes. En este capítulo se considerará el modelado de los requerimientos desde tres perspectivas distintas: modelos basados en el escenario, modelos de datos (información) y modelos basados en la clase. Cada una representa a los requerimientos en una “dimensión” diferente, con lo que aumenta la probabilidad de detectar errores, de que afloren las inconsistencias y de que se revelen las omisiones.

**¿Cuáles son los pasos?** El modelado basado en escenarios es una representación del sistema desde el punto de

vista del usuario. El modelado basado en datos recrea el espacio de información e ilustra los objetos de datos que manipulará el software y las relaciones entre ellos. El modelado orientado a clases define objetos, atributos y

relaciones. Una vez que se crean los modelos preliminares, se mejoran y analizan para evaluar si están claros y completos, y si son consistentes. En el capítulo 7 se amplían con representaciones adicionales las dimensiones del modelado descritas aquí, lo que da un punto de vista más sólido de los requerimientos.

**¿Cuál es el producto final?** Para construir el modelo de requerimientos, se escoge una amplia variedad de representaciones basadas en texto y en diagramas. Cada una de dichas representaciones da una perspectiva de uno o más de los elementos del modelo.

**¿Cómo me aseguro de que lo hice bien?** Los productos del trabajo para modelar los requerimientos deben revisarse para saber si son correctos, completos y consistentes. Deben reflejar las necesidades de todos los participantes y establecer el fundamento desde el que se realizará el diseño.

<sup>1</sup> En ediciones anteriores de este libro, se usó el término *modelo de análisis*, en lugar de *modelo de requerimientos*. En esta edición, el autor decidió usar ambas expresiones para designar la actividad que define distintos aspectos del problema por resolver. *Análisis* es lo que ocurre cuando se obtienen los *requerimientos*.

Aunque DeMarco escribió hace más de un cuarto de siglo acerca de los atributos del modelado del análisis, sus comentarios aún son aplicables a los métodos y notación modernos del modelado de los requerimientos.

## 6.1 ANÁLISIS DE LOS REQUERIMIENTOS

El análisis de los requerimientos da como resultado la especificación de las características operativas del software, indica la interfaz de éste y otros elementos del sistema, y establece las restricciones que limitan al software. El análisis de los requerimientos permite al profesional (sin importar si se llama *ingeniero de software, analista o modelista*) construir sobre los requerimientos básicos establecidos durante las tareas de concepción, indagación y negociación, que son parte de la ingeniería de los requerimientos (véase el capítulo 5).

La acción de modelar los requerimientos da como resultado uno o más de los siguientes tipos de modelo:

- *Modelos basados en el escenario* de los requerimientos desde el punto de vista de distintos “actores” del sistema.
- *Modelos de datos*, que ilustran el dominio de información del problema.
- *Modelos orientados a clases*, que representan clases orientadas a objetos (atributos y operaciones) y la manera en la que las clases colaboran para cumplir con los requerimientos del sistema.
- *Modelos orientados al flujo*, que representan los elementos funcionales del sistema y la manera como transforman los datos a medida que se avanza a través del sistema.
- *Modelos de comportamiento*, que ilustran el modo en el que se comparte el software como consecuencia de “eventos” externos.



Cita:

“Cualquier ‘vista’ de los requerimientos es insuficiente para entender o describir el comportamiento deseado de un sistema complejo.”

Alan M. Davis



### CLAVE

El modelo de análisis y la especificación de requerimientos proporcionan un medio para evaluar la calidad una vez construido el software.

Estos modelos dan al diseñador del software la información que se traduce en diseños de arquitectura, interfaz y componentes. Por último, el modelo de requerimientos (y la especificación de requerimientos de software) brinda al desarrollador y al cliente los medios para evaluar la calidad una vez construido el software.

Este capítulo se centra en el *modelado basado en escenarios*, técnica que cada vez es más popular entre la comunidad de la ingeniería de software; el *modelado basado en datos*, más especializado, apropiado en particular cuando debe crearse una aplicación o bien manipular un espacio complejo de información; y el *modelado orientado a clases*, representación de las clases orientada a objetos y a las colaboraciones resultantes que permiten que funcione el sistema. En

FIGURA 6.1

**El modelo de requerimientos como puente entre la descripción del sistema y el modelo del diseño**

Descripción del sistema

Modelo del análisis

Modelo del diseño



el capítulo 7 se analizan los modelos orientados al flujo, al comportamiento, basados en el patrón y en *webapps*.

### Objetivos y filosofía general



Cita:

"Los requerimientos no son arquitectura. No son diseño ni la interfaz de usuario. Los requerimientos son las necesidades."

Andrew Hunt y David Thomas

Durante el modelado de los requerimientos, la atención se centra en *qué*, no en *cómo*. ¿Qué interacción del usuario ocurre en una circunstancia particular?, ¿qué objetos manipula el sistema?, ¿qué funciones debe realizar el sistema?, ¿qué comportamientos tiene el sistema?, ¿qué interfaces se definen? y ¿qué restricciones son aplicables?<sup>2</sup>

En los capítulos anteriores se dijo que en esta etapa tal vez no fuera posible tener la especificación completa de los requerimientos. El cliente quizás no esté seguro de qué es lo que quiere con precisión para ciertos aspectos del sistema. Puede ser que el desarrollador esté inseguro de que algún enfoque específico cumpla de manera apropiada la función y el desempeño. Estas realidades hablan a favor de un enfoque iterativo para el análisis y el modelado de los requerimientos. El analista debe modelar lo que se sabe y usar el modelo como base para el diseño del incremento del software.<sup>3</sup>

El modelo de requerimientos debe lograr tres objetivos principales: 1) describir lo que quiere el cliente, 2) establecer una base para la creación de un diseño de software y 3) definir un conjunto de requerimientos que puedan validarse una vez construido el software. El modelo de análisis es un puente entre la descripción en el nivel del sistema que se centra en éste en lo general o en la funcionalidad del negocio que se logra con la aplicación de software, hardware, datos, personas y otros elementos del sistema y un diseño de software (véanse los capítulos 8 a 13) que describa la arquitectura de la aplicación del software, la interfaz del usuario y la estructura en el nivel del componente. Esta relación se ilustra en la figura 6.1.

Es importante observar que todos los elementos del modelo de requerimientos pueden trasladarse directamente hasta las partes del modelo del diseño. No siempre es posible la división clara entre las tareas del análisis y las del diseño en estas dos importantes actividades del modelado. Invariablemente, ocurre algo de diseño como parte del análisis y algo de análisis se lleva a cabo durante el diseño.

### Reglas prácticas del análisis

Arlow y Neustadt [Arl02] sugieren cierto número de reglas prácticas útiles que deben seguirse cuando se crea el modelo del análisis:

- *El modelo debe centrarse en los requerimientos que sean visibles dentro del problema o dentro del dominio del negocio. El nivel de abstracción debe ser relativamente elevado. "No se empantane en los detalles"* [Arl02] que traten de explicar cómo funciona el sistema.
- *Cada elemento del modelo de requerimientos debe agregarse al entendimiento general de los requerimientos del software y dar una visión del dominio de la información, de la función y del comportamiento del sistema.*
- *Hay que retrasar las consideraciones de la infraestructura y otros modelos no funcionales hasta llegar a la etapa del diseño.* Es decir, quizás se requiera una base de datos, pero las clases necesarias para implementarla, las funciones requeridas para acceder a ella y el comportamiento que tendrá cuando se use sólo deben considerarse después de que se haya terminado el análisis del dominio del problema.

2 Debe notarse que, a medida que los clientes tienen más conocimientos tecnológicos, hay una tendencia hacia la especificación del *cómo* tanto como del *qué*. Sin embargo, la atención debe centrarse en el *qué*.

3 En un esfuerzo por entender mejor los requerimientos para el sistema, el equipo del software tiene la alternativa de escoger la creación de un prototipo (véase el capítulo 2).



Cita:

"Los problemas que es benéfico atacar demuestran su beneficio con un contragolpe."

Piet Hein

- *Debe minimizarse el acoplamiento a través del sistema.* Es importante representar las relaciones entre las clases y funciones. Sin embargo, si el nivel de "interconectividad" es extremadamente alto, deben hacerse esfuerzos para reducirlo.
- *Es seguro que el modelo de requerimientos agrega valor para todos los participantes.* Cada actor tiene su propio uso para el modelo. Por ejemplo, los participantes de negocios deben usar el modelo para validar los requerimientos; los diseñadores deben usarlo como pase para el diseño; el personal de aseguramiento de la calidad lo debe emplear como ayuda para planear las pruebas de aceptación.
- *Mantener el modelo tan sencillo como se pueda.* No genere diagramas adicionales si no agregan nueva información. No utilice notación compleja si basta una sencilla lista.

## Análisis del dominio

### WebRef

En la dirección [www.iturls.com/English/SoftwareEngineering/SE\\_mod5.asp](http://www.iturls.com/English/SoftwareEngineering/SE_mod5.asp), existen muchos recursos útiles para el análisis del dominio.

### PONTO CLAVE

El análisis del dominio no busca en una aplicación específica, sino en el dominio en el que reside la aplicación. El objetivo es identificar elementos comunes para la solución de problemas, que sean útiles en todas las aplicaciones dentro del dominio.

Al estudiar la ingeniería de requerimientos (en el capítulo 5), se dijo que es frecuente que haya patrones de análisis que se repiten en muchas aplicaciones dentro de un dominio de negocio específico. Si éstos se definen y clasifican en forma tal que puedan reconocerse y aplicarse para resolver problemas comunes, la creación del modelo del análisis es más expedita. Más importante aún es que la probabilidad de aplicar patrones de diseño y componentes de software reutilizable se incrementa mucho. Esto mejora el tiempo para llegar al mercado y reduce los costos de desarrollo.

Pero, ¿cómo se reconocen por primera vez los patrones de análisis y clases? ¿Quién los define, clasifica y prepara para usarlos en los proyectos posteriores? La respuesta a estas preguntas está en el *análisis del dominio*. Firesmith [Fir93] lo describe del siguiente modo:

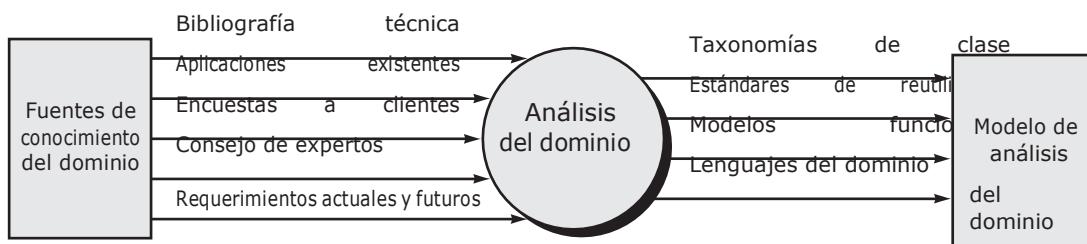
El análisis del dominio del software es la identificación, análisis y especificación de los requerimientos comunes, a partir de un dominio de aplicación específica, normalmente para usarlo varias veces en múltiples proyectos dentro del dominio de la aplicación [...] [El análisis del dominio orientado a objetos es] la identificación, análisis y especificación de capacidades comunes y reutilizables dentro de un dominio de aplicación específica en términos de objetos, clases, subensambles y estructuras comunes.

El "dominio de aplicación específica" se extiende desde el control electrónico de aviones hasta la banca, de los juegos de video en multimedios al software incrustado en equipos médicos. La meta del análisis del dominio es clara: encontrar o crear aquellas clases o patrones de análisis que sean aplicables en lo general, de modo que puedan volverse a usar.<sup>4</sup>

Con el empleo de la terminología que se introdujo antes en este libro, el análisis del dominio puede considerarse como una actividad sombrilla para el proceso del software. Esto significa que el análisis del dominio es una actividad de la ingeniería de software que no está conectada

FIGURA 6.2

Entradas y salidas para el análisis del dominio



4 Un punto de vista complementario del análisis del dominio "involucra el modelado de éste, de manera que los ingenieros del software y otros participantes aprendan más al respecto [...] no todas las clases de dominio necesariamente dan como resultado el desarrollo de clases reutilizables [...]" [Let03a].



## CASASEGURA



### Análisis del dominio

**La escena:** Oficina de Doug Miller, después de una reunión con personal de mercadotecnia.

**Participantes:** Doug Miller, gerente de ingeniería de software, y Vinod Raman, miembro del equipo de ingeniería de software.

#### La conversación:

**Doug:** Tenemos que hacer un proyecto especial, Vinod. Voy a retirarte de las reuniones para recabar los requerimientos.

**Vinod (con el ceño fruncido):** Muy mal. Ese formato en verdad funciona... Estaba sacando algo de ahí. ¿Qué pasa?

**Doug:** Jamie y Ed te cubrirán. De cualquier manera, el departamento de mercadotecnia insiste en que en la primera entrega de *CasaSegura* dispongamos de la capacidad de acceso por internet junto con la función de seguridad para el hogar. Estamos bajo fuego en esto... sin tiempo ni personal suficiente, así que tenemos que resolver ambos problemas a la vez: la interfaz de PC y la interfaz de web.

**Vinod (confundido):** No sabía que el plan era entregar... ni siquiera hemos terminado de recabar los requerimientos.

**Doug (con una sonrisa tenua):** Lo sé, pero los plazos son tan breves que decidí comenzar ya la estrategia con mercadotecnia... de cualquier modo, revisaremos cualquier plan tentativo una vez que tengamos la información de todas las juntas que se efectuarán para recabar los requerimientos.

**Vinod:** Está bien, entonces? ¿Qué quieres que haga?

**Doug:** ¿Sabes qué es el "análisis del dominio"?

**Vinod:** Algo sé. Buscas patrones similares en aplicaciones que hagan lo mismo que la que estás elaborando. Entonces, si es posible, calcas los patrones y los reutilizas en tu trabajo.

**Doug:** No estoy seguro de que la palabra sea *calcar*, pero básicamente tienes razón. Lo que me gustaría que hicieras es que comienzas a buscar interfaces de usuario ya existentes para sistemas que controlen algo como *CasaSegura*. Quiero que propongas un conjunto de patrones y clases de análisis que sean comunes tanto a la interfaz basada en PC que estará en el hogar como a la basada en un navegador al que se accederá por internet.

**Vinod:** Ahorraríamos tiempo si las hicieramos iguales... ¿por qué no las hacemos así?

**Doug:** Ah... es grato tener gente que piense como lo haces tú. Ése es el meollo del asunto: ahorraremos tiempo y esfuerzo si las dos interfaces son casi idénticas; las implementamos con el mismo código y acabamos con la insistencia de mercadotecnia.

**Vinod:** ¿Entonces, qué quieres?, ¿clases, patrones de análisis, patrones de diseño?

**Doug:** Todo eso. Nada formal en este momento. Sólo quiero que comencemos despacio con nuestros trabajos de análisis interno y de diseño.

**Vinod:** Iré a nuestra biblioteca de clases y veré qué tenemos. También usaré un formato de patrones que vi en un libro que leí hace unos meses.

**Doug:** Bien. Manos a la obra.

con ningún proyecto de software. En cierta forma, el papel del analista del dominio es similar al de un maestro herrero en un ambiente de manufactura pesada. El trabajo del herrero es diseñar y fabricar herramientas que utilicen muchas personas que hacen trabajos similares pero no necesariamente iguales. El papel del analista de dominio<sup>5</sup> es descubrir y definir patrones de análisis, clases de análisis e información relacionada que pueda ser utilizada por mucha gente que trabaje en aplicaciones similares, pero que no son necesariamente las mismas.

La figura 6.2 [Ara89] ilustra entradas y salidas clave para el proceso de análisis del dominio. Las fuentes de conocimiento del dominio se mapean con el fin de identificar los objetos que pueden reutilizarse a través del dominio.

### Enfoques del modelado de requerimientos

Un enfoque del modelado de requerimientos, llamado *análisis estructurado*, considera que los datos y los procesos que los transforman son entidades separadas. Los objetos de datos se modelan de modo que se definen sus atributos y relaciones. Los procesos que manipulan a los objetos de datos se modelan en forma que se muestre cómo transforman a los datos a medida que los objetos que se corresponden con ellos fluyen por el sistema.

#### Cita:

“... el análisis es frustrante, está lleno de relaciones interpersonales complejas, indefinidas y difíciles. En una palabra, es fascinante. Una vez atrapado, los antiguos y fáciles placeres de la construcción de sistemas nunca más volverán a satisfacerte.”

Tom DeMarco

- 5 No suponga que el ingeniero de software **no** necesita entender el dominio de la aplicación tan sólo porque hay un analista del dominio trabajando. Todo miembro del equipo del software debe entender algo del dominio en el que se va a colocar el software.

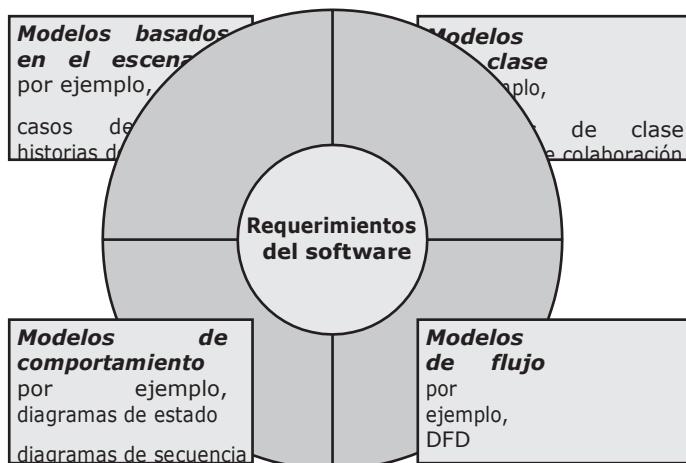


FIGURA 6.3

Elementos  
del modelo de  
análisis

?

¿Cuáles son los diferentes puntos de vista que se usan para describir el modelo de requerimientos?



Un segundo enfoque del modelado del análisis, llamado *análisis orientado a objetos*, se centra en la definición de las clases y en la manera en la que colaboran uno con el otro para cumplir los requerimientos. El UML y el proceso unificado (véase el capítulo 2) están orientados a objetos, sobre todo.

Aunque el modelo de requerimientos propuesto en este libro combina características de ambos enfoques, los equipos de software escogen con frecuencia uno y excluyen todas las representaciones del otro. La pregunta no es cuál es mejor, sino qué combinación de representaciones proporcionará a los participantes el mejor modelo de requerimientos del software y el puente más eficaz para el diseño del mismo.

Cada elemento del modelo de requerimientos (véase la figura 6.3) presenta el problema desde diferentes puntos de vista. Los elementos basados en el escenario ilustran cómo interactúa el usuario con el sistema y la secuencia específica de actividades que ocurren cuando se utiliza el software. Los elementos basados en la clase modelan los objetos que el sistema manipulará, las operaciones que se aplicarán a ellos para realizar dicha manipulación, las relaciones (algunas jerárquicas) entre los objetos y las colaboraciones que ocurrirán entre las clases que se definan. Los elementos del comportamiento ilustran la forma en la que los eventos externos cambian el estado del sistema o las clases que residen dentro de éste. Por último, los elementos orientados al flujo representan al sistema como una transformación de la información e ilustran la forma en la que se transforman los objetos de datos cuando fluyen a través de las distintas funciones del sistema.

El modelado del análisis lleva a la obtención de cada uno de estos elementos de modelado. Sin embargo, el contenido específico de cada elemento (por ejemplo, los diagramas que se emplean para construir el elemento y el modelo) tal vez difiera de un proyecto a otro. Como se ha dicho varias veces en este libro, el equipo del software debe trabajar para mantenerlo sencillo. Sólo deben usarse elementos de modelado que agreguen valor al modelo.

## 6.2 MODELADO BASADO EN ESCENARIOS

Aunque el éxito de un sistema o producto basado en computadora se mide de muchas maneras, la satisfacción del usuario ocupa el primer lugar de la lista. Si se entiende cómo desean interactuar los usuarios finales (y otros actores) con un sistema, el equipo del software estará mejor preparado para caracterizar adecuadamente los requerimientos y hacer análisis significativos y

modelos del diseño. Entonces, el modelado de los requerimientos con UML<sup>6</sup> comienza con la creación de escenarios en forma de casos de uso, diagramas de actividades y diagramas tipo carril de natación.

### Creación de un caso preliminar de uso

Cita:

"[Los casos de uso] simplemente son una ayuda para definir lo que existe fuera del sistema (actores) y lo que debe realizar el sistema (casos de uso)."

Ivar Jacobson



*En ciertas situaciones, los casos de uso se convierten en el mecanismo dominante de la ingeniería de requerimientos. Sin embargo, esto no significa que deban descartarse otros métodos de modelado cuando resulten apropiados.*

Alistair Cockburn caracteriza un caso de uso como un "contrato para el comportamiento" [Coc01b]. Como se dijo en el capítulo 5, el "contrato" define la forma en la que un actor<sup>7</sup> utiliza un sistema basado en computadora para alcanzar algún objetivo. En esencia, un caso de uso capta las interacciones que ocurren entre los productores y consumidores de la información y el sistema en sí. En esta sección se estudiará la forma en la que se desarrollan los casos de uso como parte de los requerimientos de la actividad de modelado.<sup>8</sup>

En el capítulo 5 se dijo que un caso de uso describe en lenguaje claro un escenario específico desde el punto de vista de un actor definido. Pero, ¿cómo se sabe sobre qué escribir, cuánto escribir sobre ello, cuán detallada hacer la descripción y cómo organizarla? Son preguntas que deben responderse si los casos de uso han de tener algún valor como herramienta para modelar los requerimientos.

**¿Sobre qué escribir?** Las dos primeras tareas de la ingeniería de requerimientos –concepción e indagación– dan la información que se necesita para comenzar a escribir casos de uso. Las reuniones para recabar los requerimientos, el DEC, y otros mecanismos para obtenerlos se utilizan para identificar a los participantes, definir el alcance del problema, especificar los objetivos operativos generales, establecer prioridades, delineando todos los requerimientos funcionales conocidos y describir las cosas (objetos) que serán manipuladas por el sistema.

Para comenzar a desarrollar un conjunto de casos de uso, se enlistan las funciones o actividades realizadas por un actor específico. Éstas se obtienen de una lista de las funciones requeridas del sistema, por medio de conversaciones con los participantes o con la evaluación de los diagramas de actividades (véase la sección 6.3.1) desarrollados como parte del modelado de los requerimientos.

## CASASEGURA



### Desarrollo de otro escenario preliminar de uso

**La escena:** Sala de juntas, durante la segunda reunión para recabar los requerimientos.

**Participantes:** Jamie Lazar, miembro del equipo del software; Ed Robbins, integrante del equipo del software; Doug Miller, gerente de ingeniería de software; tres miembros de mercadotecnia; un representante de ingeniería del producto, y un facilitador.

#### La conversación:

**Facilitador:** Es hora de que hablamos sobre la función de vigilancia de CasaSegura. Vamos a desarrollar un escenario de usuario que accede a la función de vigilancia.

**Jamie:** ¿Quién juega el papel del actor aquí?

**Facilitador:** Creo que Meredith (persona de mercadotecnia) ha estado trabajando en dicha funcionalidad. ¿Por qué no adoptas tú ese papel?

**Meredith:** Quieres que lo hagamos de la misma forma que la vez pasada, ¿verdad?

**Facilitador:** Sí... en cierto modo.

**Meredith:** Bueno, es obvio que la razón de la vigilancia es permitir que el propietario de la casa la revise cuando se encuentre fuera, así como poder grabar y reproducir el video que se grabe... esa clase de cosas.

**Ed:** ¿Usaremos compresión para guardar el video?

<sup>6</sup> En todo el libro se usará UML como notación para elaborar modelos. En el apéndice 1 se ofrece un método breve de enseñanza para aquellos lectores que no estén familiarizados con lo más básico de dicha notación.

<sup>7</sup> Un actor no es una persona específica sino el rol que desempeña ésta (o un dispositivo) en un contexto específico. Un actor "llama al sistema para que entregue uno de sus servicios" [Coc01b].

<sup>8</sup> Los casos de uso son una parte del modelado del análisis de importancia especial para las interfaces. El análisis de la interfaz se estudia en detalle en el capítulo 11.



**Facilitador:** Buena pregunta, Ed, pero por ahora pospondremos los aspectos de la implementación. ¿Meredith?

**Meredith:** Bien, básicamente hay dos partes en la función de vigilancia... la primera configura el sistema, incluso un plano de la planta —tiene que haber herramientas que ayuden al propietario a hacer esto—, y la segunda parte es la función real de vigilancia. Como el plano es parte de la actividad de configuración, me centraré en la función de vigilancia.

**Facilitador (sonríe):** Me quitaste las palabras de la boca.

**Meredith:** Mmm... quiero tener acceso a la función de vigilancia, ya sea por PC o por internet. Tengo la sensación de que el acceso por internet se usaría con más frecuencia. De cualquier manera, quisiéra poder mostrar vistas de la cámara en una PC y controlar el ángulo y acercamiento de una cámara en particular. Especificaría la

cámara seleccionándola en el plano de la casa. También quiero poder bloquear el acceso a una o más cámaras con una clave determinada. Además, desearía tener la opción de ver pequeñas ventanas con vistas de todas las cámaras y luego escoger una que desee agrandar.

**Jamie:** Ésas se llaman vistas reducidas.

**Meredith:** Bien, entonces quiero vistas reducidas de todas las cámaras. También quisiera que la interfaz de la función de vigilancia tuviera el mismo aspecto y sensación que todas las demás del sistema *CasaSegura*. Quiero que sea intuitiva, lo que significa que no tenga que leer un manual para usarla.

**Facilitador:** Buen trabajo. Ahora, veamos esta función con un poco más de detalle...

La función (subsistema) de vigilancia de *CasaSegura* estudiada en el recuadro identifica las funciones siguientes (lista abreviada) que va a realizar el actor **propietario**:

- Seleccionar cámara para ver.
- Pedir vistas reducidas de todas las cámaras.
- Mostrar vistas de las cámaras en una ventana de PC.
- Controlar el ángulo y acercamiento de una cámara específica.
- Grabar la salida de cada cámara en forma selectiva.
- Reproducir la salida de una cámara.
- Acceder por internet a la vigilancia con cámaras.

A medida que avanzan las conversaciones con el participante (quien juega el papel de propietario), el equipo que recaba los requerimientos desarrolla casos de uso para cada una de las funciones estudiadas. En general, los casos de uso se escriben primero en forma de narración informal. Si se requiere más formalidad, se reescribe el mismo caso con el empleo de un formato estructurado, similar al propuesto en el capítulo y que se reproduce en un recuadro más adelante, en esta sección.

Para ilustrar esto, considere la función *acceder a la vigilancia con cámaras por internet-mostrar vistas de cámaras (AVC-MVC)*. El participante que tenga el papel del actor llamado **propietario** escribiría una narración como la siguiente:

**Caso de uso: acceder a la vigilancia con cámaras por internet, mostrar vistas de cámaras (AVC-MVC)**

**Actor: propietario**

Si estoy en una localidad alejada, puedo usar cualquier PC con un software de navegación apropiado para entrar al sitio web de *Productos CasaSegura*. Introduzco mi identificación de usuario y dos niveles de claves; una vez validadas, tengo acceso a toda la funcionalidad de mi sistema instalado. Para acceder a la vista de una cámara específica, selecciono "vigilancia" de los botones mostrados para las funciones principales. Luego selecciono "escoger una cámara" y aparece el plano de la casa. Despues elijo la cámara que me interesa. Alternativamente, puedo ver la vista de todas las cámaras simultáneamente si selecciono "todas las cámaras". Una vez que escojo una, selecciono "vista" y en la ventana que cubre la cámara aparece una vista con velocidad de un cuadro por segundo. Si quiero cambiar entre las cámaras, selecciono "escoger una cámara" y desaparece la vista original y de nuevo se muestra el plano de la casa. Despues, selecciono la cámara que me interesa. Aparece una nueva ventana de vistas.

Una variación de la narrativa del caso de uso presenta la interacción como una secuencia ordenada de acciones del usuario. Cada acción está representada como enunciado declarativo. Al visitar la función **ACS-DCV**, se escribiría lo siguiente:

**Caso de uso: acceder a la vigilancia con cámaras por internet, mostrar vistas de cámaras (AVC-MVC)**

**Actor: propietario**

Cita:

"Los casos de uso se emplean en muchos procesos [de software]. Nuestro favorito es el que es iterativo y guiado por el riesgo."

Gerl Schneider y Jason Winters

1. El propietario accede al sitio web *Productos CasaSegura*.
2. El propietario introduce su identificación de usuario.
3. El propietario escribe dos claves (cada una de al menos ocho caracteres de longitud).
4. El sistema muestra los botones de todas las funciones principales.
5. El propietario selecciona "vigilancia" de los botones de las funciones principales.
6. El propietario elige "seleccionar una cámara".
7. El sistema presenta el plano de la casa.
8. El propietario escoge el ícono de una cámara en el plano de la casa.
9. El propietario selecciona el botón "vista".
10. El sistema presenta la ventana de vista identificada con la elección de la cámara.
11. El sistema muestra un video dentro de la ventana a velocidad de un cuadro por segundo.

Es importante observar que esta presentación en secuencia no considera interacciones alternativas (la narración fluye con más libertad y representa varias alternativas). Los casos de este tipo en ocasiones se denominan *escenarios primarios* [Sch98a].

### Mejora de un caso de uso preliminar

Para entender por completo la función que describe un caso de uso, es esencial describir interacciones alternativas. Después se evalúa cada paso en el escenario primario, planteando las preguntas siguientes [Sch98a]:

¿Cuando desarrollo un caso de uso, ¿cómo examino los cursos alternativos de acción?

- *¿El actor puede emprender otra acción en este punto?*
- *¿Es posible que el actor encuentre alguna condición de error en este punto? Si así fuera, ¿cuál podría ser?*
- *En este punto, ¿es posible que el actor encuentre otro comportamiento (por ejemplo, alguno que sea invocado por cierto evento fuera del control del actor)? En ese caso, ¿cuál sería?*

Las respuestas a estas preguntas dan como resultado la creación de un conjunto de *escenarios secundarios* que forman parte del caso de uso original, pero que representan comportamientos alternativos. Por ejemplo, considere los pasos 6 y 7 del escenario primario ya descrito:

6. El propietario elige "seleccionar una cámara".
7. El sistema presenta el plano de la casa.

*¿El actor puede emprender otra acción en este punto?* La respuesta es "sí". Al analizar la narración de flujo libre, el actor puede escoger mirar vistas de todas las cámaras simultáneamente. Entonces, un escenario secundario sería "observar vistas instantáneas de todas las cámaras".

*¿Es posible que el actor encuentre alguna condición de error en este punto?* Cualquier número de condiciones de error puede ocurrir cuando opera un sistema basado en computadora. En este contexto, sólo se consideran las condiciones que sean probables como resultado directo de la acción descrita en los pasos 6 o 7. De nuevo, la respuesta es "sí". Tal vez nunca se haya configurado un plano con íconos de cámara. Entonces, elegir "seleccionar una cámara" da como



resultado una condición de error: “No hay plano configurado para esta casa.”<sup>9</sup> Esta condición de error se convierte en un escenario secundario.

En este punto, ¿es posible que el actor encuentre otro comportamiento (por ejemplo, alguno que sea invocado por cierto evento fuera del control del actor)? Otra vez, la respuesta es “sí”. A medida que ocurran los pasos 6 y 7, el sistema puede hallar una condición de alarma. Esto dará como resultado que el sistema desplegará una notificación especial de alarma (tipo, ubicación, acción del sistema) y proporcionará al actor varias opciones relevantes según la naturaleza de la alarma. Como este escenario secundario puede ocurrir en cualquier momento para prácticamente todas las interacciones, no se vuelve parte del caso de uso **AVC-MVC**. En vez de ello, se desarrollará un caso de uso diferente —**Condición de alarma encontrada**— al que se hará referencia desde otros casos según se requiera.

Cada una de las situaciones descritas en los párrafos precedentes se caracteriza como una excepción al caso de uso. Una excepción describe una situación (ya sea condición de falla o alternativa elegida por el actor) que ocasiona que el sistema presente un comportamiento algo distinto.

Cockburn [Coc01b] recomienda el uso de una sesión de “lluvia de ideas” para obtener un conjunto razonablemente complejo de excepciones para cada caso de uso. Además de las tres preguntas generales ya sugeridas en esta sección, también deben explorarse los siguientes aspectos:

- ¿Existen casos en los que ocurra alguna “función de validación” durante este caso de uso? Esto implica que la función de validación es invocada y podría ocurrir una potencial condición de error.
- ¿Hay casos en los que una función (o actor) de soporte falle en responder de manera apropiada? Por ejemplo, una acción de usuario espera una respuesta pero la función que ha de responder se cae.
- ¿El mal desempeño del sistema da como resultado acciones inesperadas o impropias? Por ejemplo, una interfaz con base en web responde con demasiada lentitud, lo que da como resultado que un usuario haga selecciones múltiples en un botón de procesamiento. Estas selecciones se forman de modo equivocado y, en última instancia, generan un error.

La lista de extensiones desarrollada como consecuencia de preguntar y responder estas preguntas debe “racionalizarse” [Coc01b] con el uso de los siguientes criterios: una excepción debe describirse dentro del caso de uso si el software la puede detectar y debe manejarla una vez detectada. En ciertos casos, una excepción precipitará el desarrollo de otro caso de uso (el de manejar la condición descrita).

### Escritura de un caso de uso formal

En ocasiones, para modelar los requerimientos es suficiente con los casos de uso informales presentados en la sección 6.2.1. Sin embargo, cuando un caso de uso involucra una actividad crítica o cuando describe un conjunto complejo de etapas con un número significativo de excepciones, es deseable un enfoque más formal.

El caso de uso **AVC-MVC** mostrado en el recuadro de la página 136 sigue el guión común para los casos de uso formales. El *objetivo en contexto* identifica el alcance general del caso de

<sup>9</sup> En este caso, otro actor, **administrador del sistema**, tendría que configurar el plano de la casa, instalar e inicializar todas las cámaras (por ejemplo, asignar una identificación a los equipos) y probar cada una para garantizar que se encuentren accesibles por el sistema y a través del plano de la casa.

uso. La *precondición* describe lo que se sabe que es verdadero antes de que inicie el caso de uso. El *disparador* (o *trigger*) identifica el evento o condición que “hace que comience el caso de uso” [Coc01b]. El *escenario* enlista las acciones específicas que requiere el actor, y las respuestas apropiadas del sistema. Las *excepciones* identifican las situaciones detectadas cuando se mejora el caso de uso preliminar (véase la sección 6.2.2). Pueden incluirse o no encabezados adicionales y se explican por sí mismos en forma razonable.

## CASASEGURA



### Formato de caso de uso para vigilancia

**Caso de uso:** Acceder a la vigilancia con cámaras por internet, mostrar vistas de cámaras (AVC-MVC).

**Iteración:** 2, última modificación: 14 de enero por V. Raman.

**Actor principal:** Propietario.

**Objetivo en contexto:** Ver la salida de las cámaras colocadas en la casa desde cualquier ubicación remota por medio de internet.

**Precondiciones:** El sistema debe estar configurado por completo; deben obtenerse las identificaciones y claves de usuario apropiadas.

**Disparador:** El propietario decide ver dentro de la casa mientras está fuera.

#### Escenario:

- El propietario se registra en el sitio web *Productos CasaSegura*.
- El propietario introduce su identificación de usuario.
- El propietario proporciona dos claves (cada una con longitud de al menos ocho caracteres).
- El sistema despliega todos los botones de las funciones principales.
- El propietario selecciona “vigilancia” entre los botones de funciones principales.
- El propietario escoge “seleccionar una cámara”.
- El sistema muestra el plano de la casa.
- El propietario selecciona un ícono de cámara en el plano de la casa.
- El propietario pulsa el botón “vista”.
- El sistema muestra la ventana de la vista de la cámara identificada.
- El sistema presenta una salida de video dentro de la ventana de vistas, con una velocidad de un cuadro por segundo.

#### Excepciones:

- La identificación o las claves son incorrectas o no se reconocen (véase el caso de uso **Validar identificación y claves**).
- La función de vigilancia no está configurada para este sistema (el sistema muestra el mensaje de error apropiado; véase el caso de uso **Configurar la función de vigilancia**).
- El propietario selecciona “Mirar vistas reducidas de todas las cámaras” (véase el caso de uso **Mirar vistas reducidas de todas las cámaras**).
- No se dispone o no se ha configurado el plano de la casa (se muestra el mensaje de error apropiado y véase el caso de uso **Configurar plano de la casa**).
- Se encuentra una condición de alarma (véase el caso de uso **Condición de alarma encontrada**).

**Prioridad:** Moderada, por implementarse después de las funciones básicas.

**Cuándo estará disponible:** En el tercer incremento.

**Frecuencia de uso:** Frecuencia moderada.

**Canal al actor:** A través de un navegador con base en PC y conexión a internet.

**Actores secundarios:** Administrador del sistema, cámaras.

#### Canales a los actores secundarios:

- Administrador del sistema: sistema basado en PC.
- Cámaras: conectividad inalámbrica.

#### Asuntos pendientes:

- ¿Qué mecanismos protegen el uso no autorizado de esta capacidad por parte de los empleados de *Productos CasaSegura*?
- Es suficiente la seguridad? El acceso ilegal a esta característica representaría una invasión grave de la privacidad.
- ¿Será aceptable la respuesta del sistema por internet dado el ancho de banda que requieren las vistas de las cámaras?
- ¿Desarrollaremos una capacidad que provea el video a una velocidad más alta en cuadros por segundo cuando se disponga de conexiones con un ancho de banda mayor?

#### WebRef

¿Cuándo se ha terminado de escribir casos de uso? Para un análisis benéfico de esto, consulte la dirección [otips.org/use-cases-done.html](http://otips.org/use-cases-done.html).

**SEP**

SECRETARÍA DE  
EDUCACIÓN PÚBLICA

En muchos casos, no hay necesidad de crear una representación gráfica de un escenario de uso. Sin embargo, la representación con diagramas facilita la comprensión, en particular cuando el escenario es complejo. Como ya se dijo en este libro, UML cuenta con la capacidad de hacer diagramas de casos de uso. La figura 6.4 ilustra un diagrama de caso de uso preliminar para el producto *CasaSegura*. Cada caso de uso está representado por un óvalo. En esta sección sólo se estudia el caso de uso **AVC-MVC**.



CENTRO NACIONAL  
DE FORMACIÓN  
PROFECCIONAL  
LA EDUCACIÓN SUPERIOR, A.C.

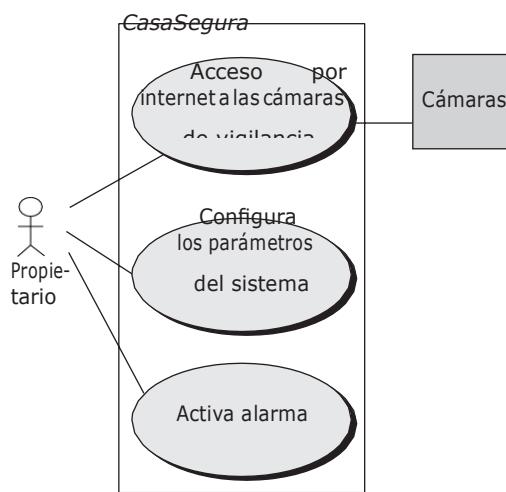


EDITORIAL  
[guiaceneval.mx](http://guiaceneval.mx)



FIGURA 6.4

Diagrama de caso de uso preliminar para el sistema  
*CasaSegura*



Toda notación de modelado tiene sus limitaciones, y la del caso de uso no es la excepción. Como cualquier otra forma de descripción escrita, un caso de uso es tan bueno como lo sea(n) su(s) autor(es). Si la descripción es poco clara, el caso de uso será confuso o ambiguo. Un caso de uso se centra en los requerimientos funcionales y de comportamiento, y por lo general es inapropiado para requerimientos disfuncionales. Para situaciones en las que el modelo de requerimientos deba tener detalle y precisión significativos (por ejemplo, sistemas críticos de seguridad), tal vez no sea suficiente un caso de uso.

Sin embargo, el modelado basado en escenarios es apropiado para la gran mayoría de todas las situaciones que encontrará un ingeniero de software. Si se desarrolla bien, el caso de uso proporciona un beneficio sustancial como herramienta de modelado.

### 6.3 MODELOS UML QUE PROPORCIONAN EL CASO DE USO

Hay muchas situaciones de modelado de requerimientos en las que un modelo basado en texto – incluso uno tan sencillo como un caso de uso – tal vez no brinde información en forma clara y concisa. En tales casos, es posible elegir de entre una amplia variedad de modelos UML gráficos.

#### Desarrollo de un diagrama de actividades

El diagrama de actividad UML enriquece el caso de uso al proporcionar una representación gráfica del flujo de interacción dentro de un escenario específico. Un diagrama de actividades es similar a uno de flujo, y utiliza rectángulos redondeados para denotar una función específica del sistema, flechas para representar flujo a través de éste, rombos de decisión para ilustrar una ramificación de las decisiones (cada flecha que salga del rombo se etiqueta) y líneas continuas para indicar que están ocurriendo actividades en paralelo. En la figura 6.5 se presenta un diagrama de actividades para el caso de uso **AVC-MVC**. Debe observarse que el diagrama de actividades agrega detalles adicionales que no se mencionan directamente (pero que están implícitos) en el caso de uso.

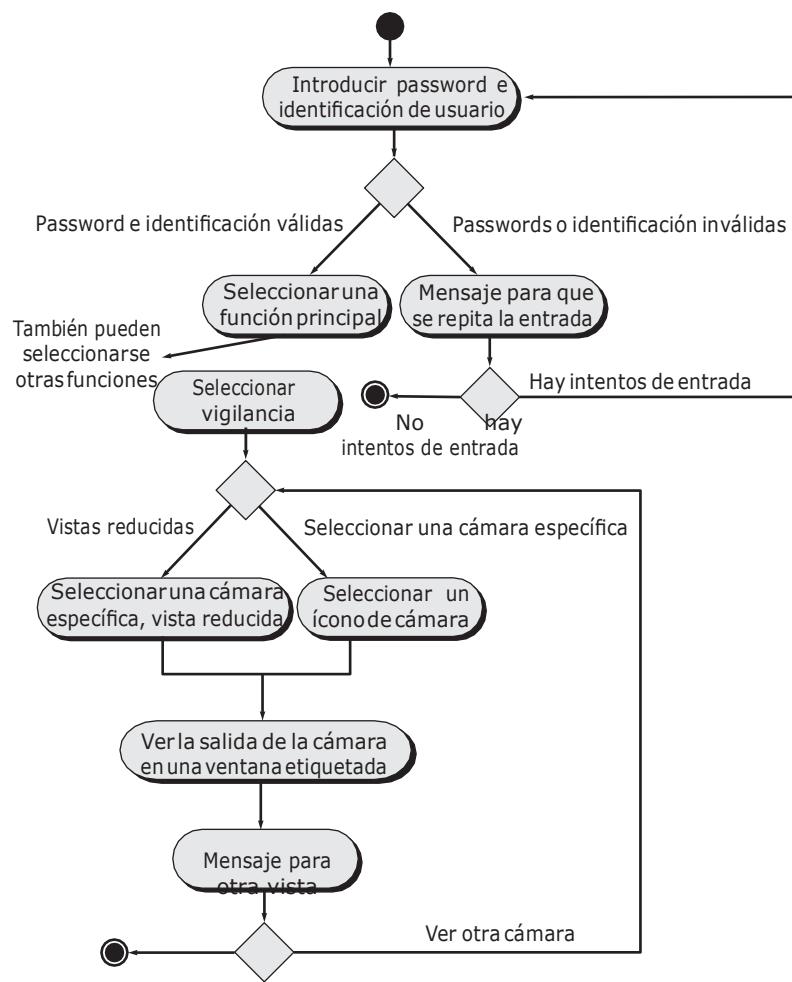
Por ejemplo, un usuario quizás sólo haga algunos intentos de introducir su **identificación** y **password**. Esto se representa por el rombo de decisión debajo de “Mensaje para que se repita la entrada”.

#### PONTO CLAVE

Un diagrama de actividades UML representa las acciones y decisiones que ocurren cuando se realiza cierta función.

FIGURA 6.5

Diagrama de actividades para la función Acceder a la vigilancia con cámaras por internet, mostrar vistas de cámaras.



### Diagramas de canal (swimlane)

El *diagrama de canal* de UML es una variación útil del diagrama de actividades y permite representar el flujo de actividades descritas por el caso de uso; al mismo tiempo, indica qué actor (si hubiera muchos involucrados en un caso específico de uso) o clase de análisis (se estudia más adelante, en este capítulo) es responsable de la acción descrita por un rectángulo de actividad. Las responsabilidades se representan con segmentos paralelos que dividen el diagrama en forma vertical, como los canales o carriles de una alberca.

Son tres las clases de análisis: **Propietario**, **Cámara** e **Interfaz**, que tienen responsabilidad directa o indirecta en el contexto del diagrama de actividades representado en la figura 6.5. En relación con la figura 6.6, el diagrama de actividades se reacomodó para que las actividades asociadas con una clase de análisis particular queden dentro del canal de dicha clase. Por ejemplo, la clase **Interfaz** representa la interfaz de usuario como la ve el propietario. El diagrama de actividades tiene dos mensajes que son responsabilidad de la interfaz: "mensaje para que se repita la entrada" y "mensaje para otra vista". Estos mensajes y las decisiones asociadas con ellos caen dentro del canal **Interfaz**. Sin embargo, las flechas van de ese canal de regreso al de **Propietario**, donde ocurren las acciones de éste.

Los casos de uso, junto con los diagramas de actividades y de canal, están orientados al procedimiento. Representan la manera en la que los distintos actores invocan funciones específicas (u otros pasos del procedimiento) para satisfacer los requerimientos del sistema. Pero

**CLAVE**  
Un diagrama de canal (*swimlane*) representa el flujo de acciones y decisiones e indica qué actores efectúan cada una de ellas.

Cita:

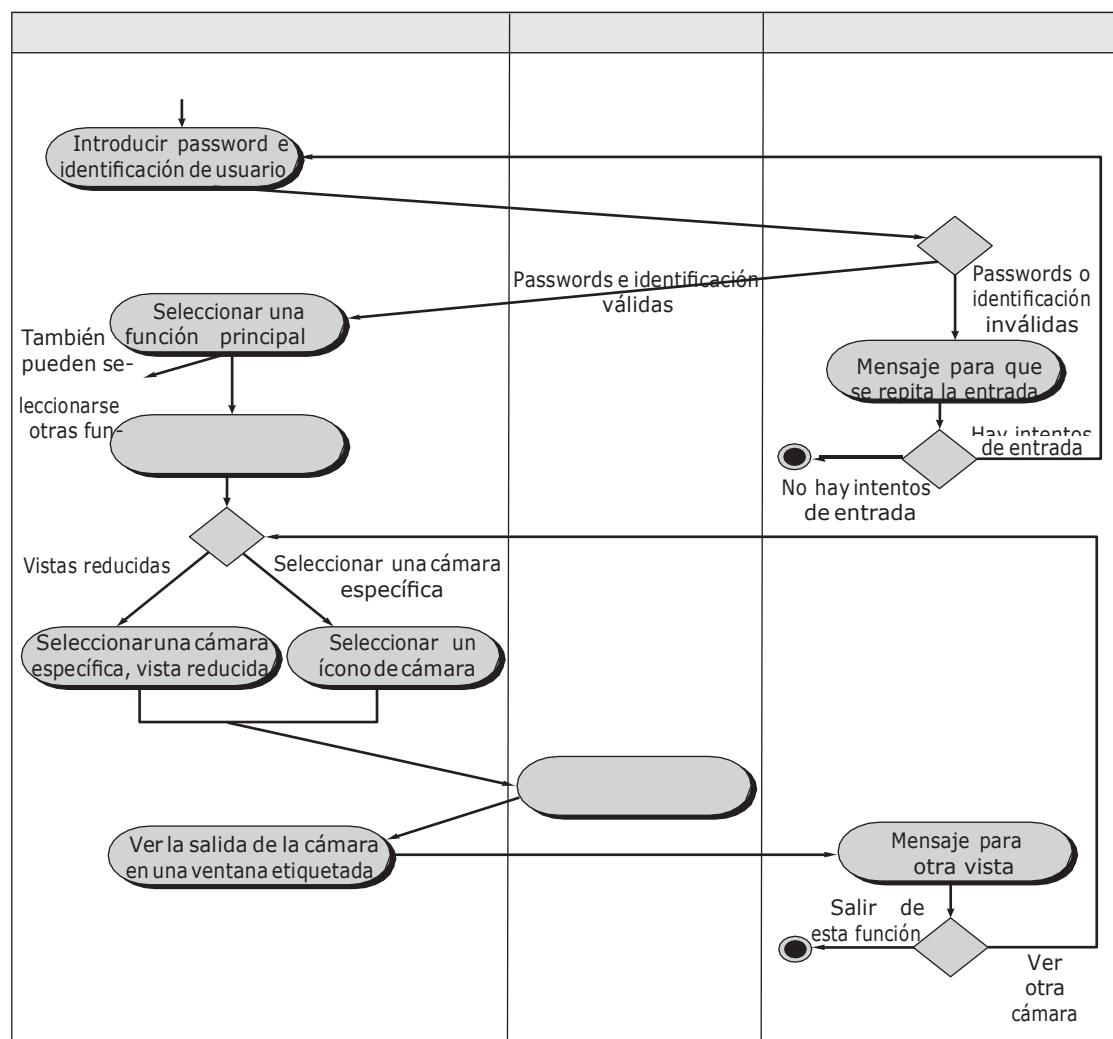
"Un buen modelo guía el pensamiento; uno malo lo desvía."

Brian Marick



FIGURA 6.6

Diagrama de canal para la función Acceder a la vigilancia con cámaras por internet, mostrar vistas de cámaras



una vista del procedimiento de los requerimientos representa una sola dimensión del sistema. En la sección 6.4 se estudia el espacio de información y la forma en la que se representan los datos de requerimientos.

## 6.4 CONCEPTOS DE MODELADO DE DATOS

### WebRef

En la dirección [www.datamodel.org](http://www.datamodel.org), hay información útil sobre el modelado de datos.

Si los requerimientos del software incluyen la necesidad de crear, ampliar o hacer interfaz con una base de datos, o si deben construirse y manipularse estructuras de datos complejas, el equipo del software tal vez elija crear un *modelo de datos* como parte del modelado general de los requerimientos. Un ingeniero o analista de software define todos los objetos de datos que se procesan dentro del sistema, la relación entre ellos y otro tipo de información que sea pertinente para las relaciones. El *diagrama entidad-relación* (DER) aborda dichos aspectos y representa todos los datos que se introducen, almacenan, transforman y generan dentro de una aplicación.



¿Cómo se manifiesta un objeto de datos en el contexto de una aplicación?

### Objetos de datos

Un *objeto de datos* es una representación de información compuesta que debe ser entendida por el software. *Información compuesta* quiere decir algo que tiene varias propiedades o atributos



## CLAVE

Un objeto de datos es una representación de cualquier información compuesta que se procese en el software.



Los atributos nombran a un objeto de **WebRef** describen sus para aquellos que intentan hacer modelado de datos, es importante un concepto llamado "normalización". En la dirección [www.datamodel.org](http://www.datamodel.org) se encuentra una introducción útil.

diferentes. Por tanto, el ancho (un solo valor) no sería un objeto de datos válido, pero las **dimensiones** (que incorporan altura, ancho y profundidad) sí podrían definirse como un objeto.

Un objeto de datos puede ser una entidad externa (por ejemplo, cualquier cosa que produzca o consuma información), una cosa (por ejemplo, un informe o pantalla), una ocurrencia (como una llamada telefónica) o evento (por ejemplo, una alarma), un rol (un vendedor), una unidad organizacional (por ejemplo, el departamento de contabilidad), un lugar (como una bodega) o estructura (como un archivo). Por ejemplo, una **persona** o un **auto** pueden considerarse como objetos de datos en tanto cada uno se define en términos de un conjunto de atributos. La descripción del objeto de datos incorpora a ésta todos sus atributos.

Un objeto de datos contiene sólo datos –dentro de él no hay referencia a las operaciones que se apliquen sobre los datos.<sup>10</sup> Entonces, el objeto de datos puede representarse en forma de tabla, como la que se muestra en la figura 6.7. Los encabezados de la tabla reflejan atributos del objeto. En este caso, un **auto** se define en términos de fabricante, modelo, número de serie, tipo de carrocería, color y propietario. El cuerpo de la tabla representa instancias específicas del objeto de datos. Por ejemplo, un Chevy Corvette es una instancia del objeto de datos **auto**.

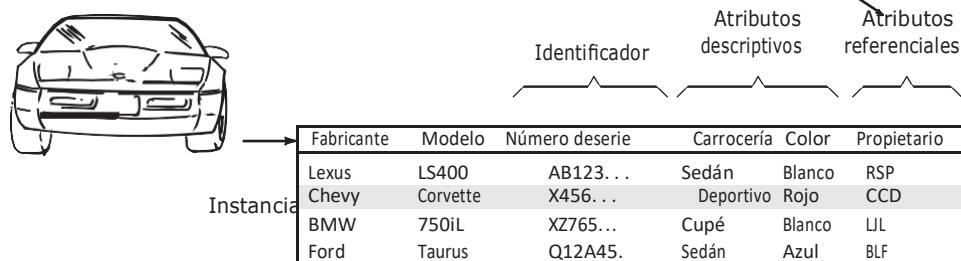
### Atributos de los datos

Los *atributos de los datos* definen las propiedades de un objeto de datos y tienen una de tres diferentes características. Se usan para 1) nombrar una instancia del objeto de datos, 2) describir la instancia o 3) hacer referencia a otra instancia en otra tabla. Además, debe definirse como identificador uno o más de los atributos –es decir, el atributo identificador se convierte en una “llave” cuando se desea encontrar una instancia del objeto de datos–. En ciertos casos, los valores para el (los) identificador(es) son únicos, aunque esto no es una exigencia. En relación con el objeto de datos **auto**, un identificador razonable sería el número de serie.

El conjunto de atributos que es apropiado para un objeto de datos determinado se define entendiendo el contexto del problema. Los atributos para **auto** podrían servir bien para una aplicación que usara un departamento de vehículos motorizados, pero serían inútiles para una compañía automotriz que necesitara hacer software de control de manufactura. En este último caso, los atributos para **auto** quizás también incluyan número de serie, tipo de carrocería y color, pero tendrían que agregarse muchos otros (por ejemplo, código interior, tipo de tracción, indicador de paquete de recorte, tipo de transmisión, etc.) para hacer de **auto** un objeto significativo en el contexto de control de manufactura.

FIGURA  
6.7

Representación tabular de objetos de datos





**INFORMACIÓN****Objetos de datos y clases orientadas a objetos: ¿son lo mismo?**

Al analizar objetos de datos es común que surja una pregunta: ¿un objeto de datos es lo mismo que una clase orientada<sup>11</sup> a objetos? La respuesta es "no".

Un objeto de datos define un aspecto de datos compuestos; es decir, incorpora un conjunto de características de datos individuales (atributos) y da al conjunto un nombre (el del objeto de datos).

Una clase orientada a objetos encierra atributos de datos, pero también incorpora las operaciones (métodos) que los manipulan y

que están determinadas por dichos atributos. Además, la definición de clases implica una infraestructura amplia que es parte del enfoque de la ingeniería de software orientada a objetos. Las clases se comunican entre sí por medio de mensajes, se organizan en jerarquías y tienen características hereditarias para los objetos que son una instancia de una clase.

**PONTO CLAVE**

Las relaciones indican la manera en la que los objetos de datos se conectan entre sí.

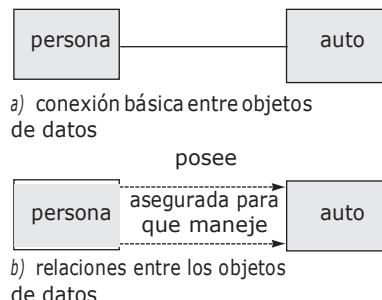
**Relaciones**

Los objetos de datos están conectados entre sí de diferentes maneras. Considere dos objetos de datos, **persona** y **auto**. Estos objetos se representan con la notación simple que se ilustra en la figura 6.8a). Se establece una conexión entre **persona** y **auto** porque ambos objetos están relacionados. Pero, ¿cuál es esa relación? Para determinarlo, debe entenderse el papel de las personas (propiedad, en este caso) y los autos dentro del contexto del software que se va a elaborar. Se establece un conjunto de parejas objeto/relación que definen las relaciones relevantes. Por ejemplo,

- Una persona *posee* un auto.
- Una persona *es asegurada para que maneje* un auto.

Las relaciones *posee* y *es asegurada para que maneje* definen las conexiones relevantes entre **persona** y **auto**. La figura 6.8b) ilustra estas parejas objeto-relación. Las flechas en esa figura dan información importante sobre la dirección de la relación y es frecuente que reduzcan las ambigüedades o interpretaciones erróneas.

**FIGURA 6.8**  
Relaciones entre  
objetos de datos



11 Los lectores que no estén familiarizados con los conceptos y terminología de la orientación a objetos deben consultar el breve instructivo que se presenta en el apéndice 2.  
CENTRO NACIONAL DE EVALUACIÓN PARA LA EDUCACIÓN SUPERIOR, A.C.



**INFORMACIÓN****Diagramas entidad-relación**

La pareja objeto-relación es la piedra angular del modelo de datos. Estas parejas se representan gráficamente con

el uso del diagrama entidad-relación (DER).<sup>12</sup> El DER fue propuesto por primera vez por Peter Chen [Che77] para diseñar sistemas de bases de datos relacionales y ha sido ampliado por otras personas. Se identifica un conjunto de componentes primarios para el DER: objetos de datos, atributos, relaciones y distintos indicadores de tipo. El propósito principal del DER es representar objetos de datos y sus relaciones.

Ya se presentó la notación DER básica. Los objetos de datos se representan con un rectángulo etiquetado. Las relaciones se indican con una línea etiquetada que conecta objetos. En ciertas variantes del DER, la línea de conexión contiene un rombo con la leyenda de la relación. Las conexiones entre los objetos de datos y las relaciones se establecen con el empleo de varios símbolos especiales que indican cardinalidad y modalidad.<sup>13</sup> Si el lector está interesado en obtener más información sobre el modelado de datos y el diagrama entidad-relación, consulte [Hob06] o [Sim05].

**Modelado de datos**

**Objetivo:** Las herramientas de modelado de datos dan a un ingeniero de software la capacidad de representar

objetos de datos, sus características y relaciones. Se usan sobre todo para aplicaciones de grandes bases de datos y otros proyectos de sistemas de información, y proveen medios automatizados para crear diagramas completos de entidad-relación, diccionarios de objetos de datos y modelos relacionados.

**Mecánica:** Las herramientas de esta categoría permiten que el usuario describa objetos de datos y sus relaciones. En ciertos casos, utilizan notación DER. En otros, modelan relaciones con el empleo de un mecanismo diferente. Es frecuente que las herramientas en esta categoría se usen como parte del diseño de una base de datos y que permitan la creación de su modelo con la generación de un esquema para sistemas comunes de administración de bases de datos comunes (DBMS).

**HERRAMIENTAS DE SOFTWARE****Herramientas representativas:**<sup>14</sup>

*AllFusion ERWin*, desarrollada por Computer Associates ([www3.ca.com](http://www3.ca.com)), ayuda en el diseño de objetos de datos, estructura apropiada y elementos clave para las bases de datos.

*ER/Studio*, desarrollada por Embarcadero Software ([www.embarcadero.com](http://www.embarcadero.com)), da apoyo al modelado entidad-relación.

*Oracle Designer*, desarrollada por Oracle Systems ([www.oracle.com](http://www.oracle.com)), "modela procesos de negocios, entidades y relaciones de datos [que] se transforman en diseños para los que se generan aplicaciones y bases de datos completas".

*Visible Analyst*, desarrollada por Visible Systems ([www.visible.com](http://www.visible.com)), da apoyo a varias funciones de modelado del análisis, incluso modelado de datos.

**6.5 MODELADO BASADO EN CLASES**

El modelado basado en clases representa los objetos que manipulará el sistema, las operaciones (también llamadas *métodos* o *servicios*) que se aplicarán a los objetos para efectuar la manipulación, las relaciones (algunas de ellas jerárquicas) entre los objetos y las colaboraciones que tienen lugar entre las clases definidas. Los elementos de un modelo basado en clases incluyen las clases y los objetos, atributos, operaciones, modelos clase-responsabilidad-colaborador (CRC), diagramas de colaboración y paquetes. En las secciones siguientes se presenta una serie de lineamientos informales que ayudarán a su identificación y representación.

12 Aunque algunas aplicaciones de diseño de bases de datos aún emplean el DER, ahora se utiliza la notación UML (véase el apéndice 1) para el diseño de datos.

13 La *cardinalidad* de una pareja objeto-relación especifica "el número de ocurrencias de uno [objeto] que se relaciona con el número de ocurrencias de otro [objeto]" [Til93]. La *modalidad* de una relación es 0 si no hay necesidad explícita para que ocurra la relación o si ésta es opcional. La modalidad es 1 si una ocurrencia de la relación es obligatoria.

14 Las herramientas mencionadas aquí no son obligatorias sino una muestra de las que hay en esta categoría. En la mayoría de casos, los nombres de las herramientas son marcas registradas por sus respectivos desarrolladores.

## Identificación de las clases de análisis



Cita:

"El problema realmente difícil es descubrir en primer lugar cuáles son los objetos correctos [clases]."

Carl Argila

Al mirar una habitación, se observa un conjunto de objetos físicos que se identifican, clasifican y definen fácilmente (en términos de atributos y operaciones). Pero cuando se “ve” el espacio del problema de una aplicación de software, las clases (y objetos) son más difíciles de concebir.

Se comienza por identificar las clases mediante el análisis de los escenarios de uso desarrollados como parte del modelo de requerimientos y la ejecución de un “análisis gramatical” [Abb83] sobre los casos de uso desarrollados para el sistema que se va a construir. Las clases se determinan subrayando cada sustantivo o frase que las incluya para introducirlo en una tabla simple. Deben anotarse los sinónimos. Si la clase (sustantivo) se requiere para implementar una solución, entonces forma parte del espacio de solución; de otro modo, si sólo es necesaria para describir la solución, es parte del espacio del problema.

Pero, ¿qué debe buscarse una vez identificados todos los sustantivos? Las *clases de análisis* se manifiestan en uno de los modos siguientes:



¿Cómo se manifiestan las clases en tantos elementos del espacio de solución?

- *Entidades externas* (por ejemplo, otros sistemas, dispositivos y personas) que producen o consumen la información que usará un sistema basado en computadora.
- *Cosas* (reportes, pantallas, cartas, señales, etc.) que forman parte del dominio de información para el problema.
- *Ocurrencias o eventos* (como una transferencia de propiedad o la ejecución de una serie de movimientos de un robot) que suceden dentro del contexto de la operación del sistema.
- *Roles* (gerente, ingeniero, vendedor, etc.) que desempeñan las personas que interactúan con el sistema.
- *Unidades organizacionales* (división, grupo, equipo, etc.) que son relevantes para una aplicación.
- *Lugares* (piso de manufactura o plataforma de carga) que establecen el contexto del problema y la función general del sistema.
- *Estructuras* (sensores, vehículos de cuatro ruedas, computadoras, etc.) que definen una clase de objetos o clases relacionadas de éstos.

Esta clasificación sólo es una de muchas propuestas en la bibliografía.<sup>15</sup> Por ejemplo, Budd [Bud96] sugiere una taxonomía de clases que incluye *productores* (fuentes) y *consumidores* (suministros) de datos, *administradores de datos*, *vista*, *clases de observador* y *clases de auxiliares*.

También es importante darse cuenta de lo que no son las clases u objetos. En general, una clase nunca debe tener un “nombre de procedimiento imperativo” [Cas89]. Por ejemplo, si los desarrolladores del software de un sistema de imágenes médicas definieron un objeto con el nombre **InvertirImagen** o incluso **InversióndeImagen**, cometían un error sutil. La **Imagen** obtenida del software podría ser, por supuesto, una clase (algo que es parte del dominio de la información). La inversión de la imagen es una operación que se aplica al objeto. Es probable que la inversión esté definida como una operación para el objeto **Imagen**, pero no lo estaría como clase separada con la connotación “inversión de imagen”. Como afirma Cashman [Cas89]: “el intento de la orientación a objetos es contener, pero mantener separados, los datos y las operaciones sobre ellos”.

Para ilustrar cómo podrían definirse las clases del análisis durante las primeras etapas del modelado, considere un análisis gramatical (los sustantivos están subrayados, los verbos apa-

15 En la sección 6.5.4 se estudia otra clasificación importante que define las clases *entidad*, *frontera* y *controladora*.



recen en cursivas) de una narración de procesamiento<sup>16</sup> para la función de seguridad de *Casa-Segura*.

La función de seguridad *CasaSegura* permite que el propietario configure el sistema de cuando se instala, vigila todos los sensores conectados al sistema de seguridad e interactúa con el propietario a través de internet, una PC o panel de control.

Durante la instalación, la PC de *CasaSegura* se utiliza para programar y configurar el sistema. Se asigna a cada sensor un número y tipo, se programa un password maestro para activar y desactivar el sistema y se introducen números telefónicos para marcar cuando ocurre un evento de sensor.



*El análisis gramatical no es una prueba de todo, pero da un impulso excelente para arrancar si se tienen dificultades para definir objetos de datos y las transformaciones que operan sobre ellos.*

Cuando se reconoce un evento de sensor, el software invoca una alarma audible instalada en el sistema. Despues de un tiempo de retraso que especifica el propietario durante las actividades de configuración del sistema, el software marca un número telefónico de un servicio de monitoreo, proporciona información acerca de la ubicación y reporta la naturaleza del evento detectado. El número telefónico se vuelve a marcar cada 20 segundos hasta que se obtiene la conexión telefónica.

El propietario recibe información de seguridad a través de un panel de control, la PC o un navegador, lo que en conjunto se llama interfaz. La interfaz despliega mensajes de aviso e información de estado del sistema en el panel de control, la PC o la ventana del navegador. La interacción del propietario tiene la siguiente forma...

Con los sustantivos se proponen varias clases potenciales:

Clase potencial	Clasificación general
propietario	rol de entidad externa
sensor	entidad externa
panel de control	entidad externa
instalación	ocurrencia
sistema (alias sistema de seguridad)	cosa
número, tipo	no objetos, atributos de sensor
password maestro	cosa
número telefónico	cosa
eventodesensor	ocurrencia
alarma audible	entidad externa
servicio de monitoreo	unidad organizacional o entidad externa

La lista continuará hasta que se hayan considerado todos los sustantivos en la narrativa de procesamiento. Observe que cada entrada en la lista se llama objeto potencial. El lector debe considerar cada una antes de tomar la decisión final.

Coad y Yourdon [Coa91] sugieren seis características de selección que deben usarse cuando se considere cada clase potencial para incluirla en el modelo de análisis:

1. *Información retenida.* La clase potencial será útil durante el análisis sólo si debe recordarse la información sobre ella para que el sistema pueda funcionar.
2. *Servicios necesarios.* La clase potencial debe tener un conjunto de operaciones identificables que cambien en cierta manera el valor de sus atributos.

?

¿Cómo determino si una clase potencial debe, en realidad, ser una clase de análisis?

<sup>16</sup> Una narración de procesamiento es similar al caso de uso en su estilo, pero algo distinto en su propósito. La narración de procesamiento hace una descripción general de la función que se va a desarrollar. No es un escenario escrito desde el punto de vista de un actor. No obstante, es importante observar que el análisis gramatical también puede emplearse para todo caso de uso desarrollado como parte de la obtención de requerimientos (indagación).

3. *Atributos múltiples.* Durante el análisis de los requerimientos, la atención debe estar en la información “principal”; en realidad, una clase con un solo atributo puede ser útil durante el diseño, pero es probable que durante la actividad de análisis se represente mejor como un atributo de otra clase.
4. *Atributos comunes.* Para la clase potencial se define un conjunto de atributos y se aplican éstos a todas las instancias de la clase.
5. *Operaciones comunes.* Se define un conjunto de operaciones para la clase potencial y éstas se aplican a todas las instancias de la clase.
6. *Requerimientos esenciales.* Las entidades externas que aparezcan en el espacio del problema y que produzcan o consuman información esencial para la operación de cualquier solución para el sistema casi siempre se definirán como clases en el modelo de requerimientos.

 Cita:  
 "Las clases luchan, algunas triunfan, otras son eliminadas."  
**Mao Tse Tung**

Para que se considere una clase legítima para su inclusión en el modelo de requerimientos, un objeto potencial debe satisfacer todas (o casi todas) las características anteriores. La decisión de incluir clases potenciales en el modelo de análisis es algo subjetiva, y una evaluación posterior tal vez haga que un objeto se descarte o se incluya de nuevo. Sin embargo, el primer paso del modelado basado en clases es la definición de éstas, y deben tomarse las medidas respectivas (aun las subjetivas). Con esto en mente, se aplicarán las características de selección a la lista de clases potenciales de *CasaSegura*:

Clase potencial	Número de característica que se aplica
propietario	rechazada: 1 y 2 fallan, aunque la 6 aplica
sensor	aceptada: se aplican todas
panel de control	aceptada: se aplican todas
instalación	rechazada
sistema (alias sistema de seguridad)	aceptada: se aplican todas
número, tipo	rechazada: 3 fallan, atributos de sensores
password maestro	rechazada: 3 falla
número telefónico	rechazada: 3 falla
eventodesensor	aceptada: se aplican todas
alarma audible	aceptada: se aplican 2, 3, 4, 5 y 6
servicio de monitoreo	rechazada: 1 y 2 fallan aunque la 6 aplica

Debe notarse que: 1) la lista anterior no es exhaustiva; para completar el modelo tendrían que agregarse clases adicionales; 2) algunas de las clases potenciales rechazadas se convertirán en atributos para otras que sí fueron aceptadas (por ejemplo, número y tipo son atributos de **Sensor**, y password maestro y número telefónico pueden convertirse en atributos de **Sistema**); 3) diferentes enunciados del problema harían que se tomaran decisiones distintas para “aceptar o rechazar” (por ejemplo, si cada propietario tuviera una clave individual o se identificara con reconocimiento de voz, la clase **Propietario** satisfaría las características 1 y 2, y se aceptaría).

## Especificación de atributos

Los *atributos* describen una clase que ha sido seleccionada para incluirse en el modelo de requerimientos. En esencia, son los atributos los que definen la clase (esto aclara lo que significa la clase en el contexto del espacio del problema). Por ejemplo, si se fuera a construir un sistema que analiza estadísticas de jugadores de béisbol profesionales, los atributos de la clase **Jugador** serían muy distintos de los que tendría la misma clase cuando se usara en el contexto del sis-

tema de pensiones de dicho deporte. En la primera, atributos tales como nombre, porcentaje de bateo, porcentaje de fildeo, años jugados y juegos jugados serían relevantes. Para la segunda, algunos de los anteriores sí serían significativos, pero otros se sustituirían (o se crearían) por atributos tales como salario promedio, crédito hacia el retiro completo, opciones del plan de pensiones elegido, dirección de correo, etcétera.

Para desarrollar un conjunto de atributos significativos para una clase de análisis, debe estudiarse cada caso de uso y seleccionar aquellas "cosas" que "pertenezcan" razonablemente a la clase. Además, debe responderse la pregunta siguiente para cada clase: "¿qué aspectos de los datos (compuestos o elementales) definen por completo esta clase en el contexto del problema en cuestión?"

Para ilustrarlo, se considera la clase **Sistema** definida para *CasaSegura*. El propietario configura la función de seguridad para que refleje la información de los sensores, la respuesta de la alarma, la activación o desactivación, la identificación, etc. Estos datos compuestos se representan del modo siguiente:

información de identificación = identificación del sistema + número telefónico de verificación + estado del sistema

información de respuesta de la alarma = tiempo de retraso + número telefónico

información de activación o desactivación = password maestro + número de intentos permisibles + password temporal

Cada uno de los datos a la derecha del signo igual podría definirse más, hasta un nivel elemental, pero para nuestros propósitos constituye una lista razonable de atributos para la clase **Sistema** (parte sombreada de la figura 6.9).

Los sensores forman parte del sistema general *CasaSegura*, pero no están enlistados como datos o atributos en la figura 6.9. **Sensor** ya se definió como clase, y se asociarán múltiples objetos **Sensor** con la clase **Sistema**. En general, se evita definir algo como atributo si más de uno va a asociarse con la clase.

### Definición de las operaciones



Cuando se definen operaciones para una clase de análisis, hay que centrarse en el comportamiento orientado al problema y no en los comportamientos requeridos para su implementación.

Las *operaciones* definen el comportamiento de un objeto. Aunque existen muchos tipos distintos de operaciones, por lo general se dividen en cuatro categorías principales: 1) operaciones que manipulan datos en cierta manera (por ejemplo, los agregan, eliminan, editan, seleccionan, etc.), 2) operaciones que realizan un cálculo, 3) operaciones que preguntan sobre el estado de un objeto y 4) operaciones que vigilan un objeto en cuanto a la ocurrencia de un evento de control. Estas funciones se llevan a cabo con operaciones sobre los atributos o sobre asociaciones de éstos (véase la sección 6.5.5). Por tanto, una operación debe tener "conocimiento" de la naturaleza de los atributos y de las asociaciones de la clase.

Como primera iteración al obtener un conjunto de operaciones para una clase de análisis, se estudia otra vez una narración del procesamiento (o caso de uso) y se eligen aquellas que pertenezcan razonablemente a la clase. Para lograr esto, de nuevo se efectúa el análisis gramatical y se aislan los verbos. Algunos de éstos serán operaciones legítimas y se conectarán con facilidad a una clase específica. Por ejemplo, de la narración del procesamiento de *CasaSegura* ya presentada en este capítulo, se observa que "se asigna a sensor un número y tipo" o "se programa un password maestro para activar y desactivar el sistema" indican cierto número de cosas:

- Que una operación *asignar()* es relevante para la clase **Sensor**.
- Que se aplicará una operación *programar()* a la clase **Sistema**.
- Que *activar()* y *desactivar()* son operaciones que se aplican a la clase **Sistema**.

**FIGURA 6.9**  
Diagrama de clase para la clase sistema

Sistema
Identificación del sistema Verificación del número telefónico Estado del sistema Tiempo de retraso Número telefónico Password maestro Password temporal Número de intentos
programa() pantalla() reiniciar() colar() activar() desactivar()

Hasta no hacer más investigaciones, es probable que la operación *programar()* se divida en cierto número de suboperaciones específicas adicionales que se requieren para configurar el sistema. Por ejemplo, *programar()* implica la especificación de números telefónicos, la configuración de las características del sistema (por ejemplo, crear la tabla de sensores, introducir las características de la alarma, etc.) y la introducción de la(s) clave(s). Pero, de momento, *programar()* se especifica como una sola operación.

Además del análisis gramatical, se obtiene más perspectiva sobre otras operaciones si se considera la comunicación que ocurre entre los objetos. Éstos se comunican con la transmisión de mensajes entre sí. Antes de continuar con la especificación de operaciones, se estudiará esto con más detalle.

## CASASEGURA



### Modelos de clase

**La escena:** Cubículo de Ed, cuando comienza el modelado de los requerimientos.

**Participantes:** Jamie, Vinod y Ed, todos ellos miembros del equipo de ingeniería de software para *CasaSegura*.

#### La conversación:

[Ed ha estado trabajando para obtener las clases a partir del formato del caso de uso para AVC-MVC (presentado en un recuadro anterior de este capítulo) y expone a sus colegas las que ha obtenido].

**Ed:** Entonces, cuando el propietario quiere escoger una cámara, la tiene que elegir del plano. Definí una clase llamada **Plano**. Éste es el diagrama.

(Observan la figura 6.10.)

**Jamie:** Entonces, **Plano** es un objeto que agrupa paredes, puertas, ventanas y cámaras. Eso significa esas líneas con leyendas, ¿verdad?

**Ed:** Sí, se llaman "asociaciones". Una clase se asocia con otra de acuerdo con las asociaciones que se ven (las asociaciones se estudian en la sección 6.5.5).

**Vinod:** Es decir, el plano real está constituido por paredes que contienen en su interior cámaras y sensores. ¿Cómo saber el plano dónde colocar estos objetos?

**Ed:** No lo sabe, pero las otras clases sí. Mira los atributos de, digamos, **SegmentodePared**, que se usa para construir una pared. El segmento de muro tiene coordenadas de inicio y final, y la operación *draw()* hace el resto.

**Jamie:** Y lo mismo vale para las ventanas y puertas. Parece como si cámara tuviera algunos atributos adicionales.

**Ed:** Sí. Los necesito para dar información del alcance y el acercamiento.

**Vinod:** Tengo una pregunta. ¿Por qué tiene la cámara una identificación pero las demás no? Veo que tienes un atributo llamado *ParedSiguiente*. ¿Cómo sabe **SegmentodePared** cuál será la pared siguiente?

**Ed:** Buena pregunta, pero, como dijimos, ésa es una decisión de diseño, por lo que la voy a retrasar hasta...

**Jamie:** Momento... Apuesto a que ya lo has imaginado.

**Ed (sonríe con timidez):** Es cierto, voy a usar una estructura de lista que modelaré cuando vayamos a diseñar. Si somos puristas en cuanto a separar el análisis y el diseño, el nivel de detalle podría parecer sospechoso.

**Jamie:** Me parece muy bien, pero tengo más preguntas.

**EDITORIAL:**

[guiaceneval.mx](http://guiaceneval.mx)



(Jamie hace preguntas que dan como resultado modificaciones menores.)

**Vinod:** ¿Tienes tarjetas CRC para cada uno de los objetos? Si así fuera, debemos actuar con ellas, sólo para estar seguros de que no hemos omitido nada.

**Ed:** No estoy seguro de cómo hacerlas.

**Vinod:** No es difícil y en verdad convienen. Te mostraré.

## Modelado clase-responsabilidad-colaborador (CRC)

Cita:

"Un propósito de las tarjetas CRC es que fallen pronto, con frecuencia y en forma barata. Es mucho más barato desechar tarjetas que reorganizar una gran cantidad de código fuente."

C. Horstman

El modelado clase-responsabilidad-colaborador (CRC) [Wir90] proporciona una manera sencilla de identificación y organización de las clases que son relevantes para los requerimientos de un sistema o producto. Ambler [Amb95] describe el modelado CRC en la siguiente forma:

Un modelo CRC en realidad es un conjunto de tarjetas índice estándar que representan clases. Las tarjetas se dividen en tres secciones. En la parte superior de la tarjeta se escribe el nombre de la clase, en la parte izquierda del cuerpo se enlistan las responsabilidades de la clase y en la derecha, los co-laboradores.

En realidad, el modelo CRC hace uso de tarjetas índice reales o virtuales. El objetivo es desarrollar una representación organizada de las clases. Las responsabilidades son los atributos y ope-

FIGURA 6.10

Diagrama de clase para Plano (véase el análisis en el recuadro)

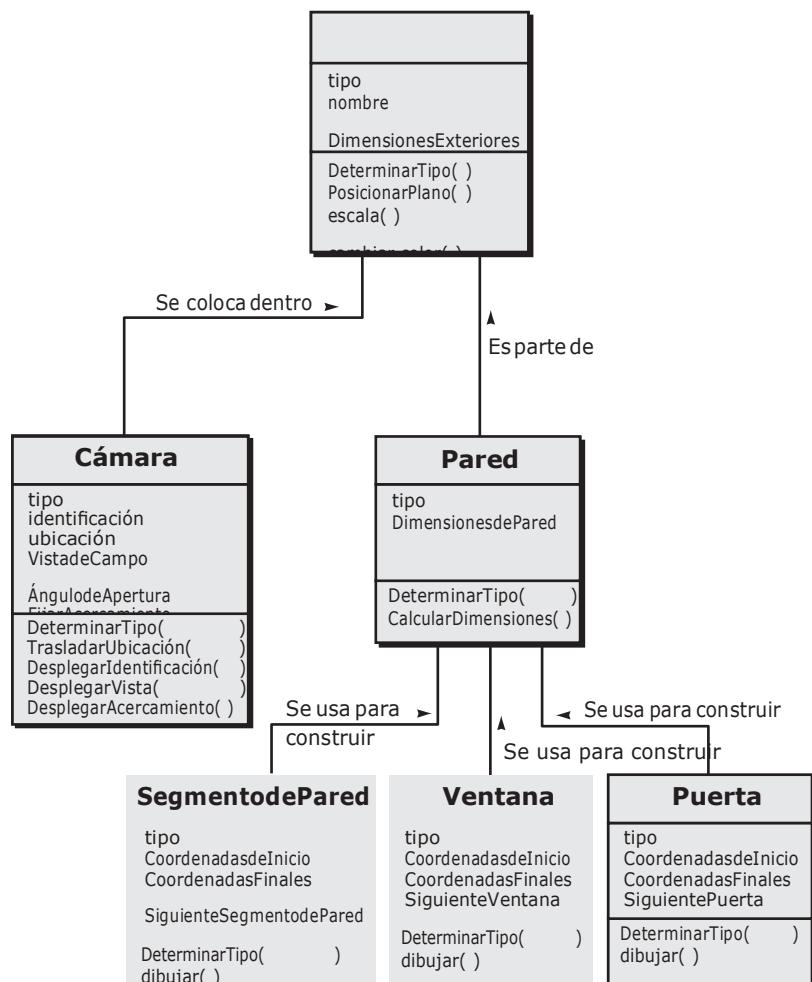


FIGURA 6.11

## **Modelo de tarjeta CRC índice**

raciones relevantes para la clase. En pocas palabras, una responsabilidad es “cualquier cosa que la clase sepa o haga” [Amb95]. Los *colaboradores* son aquellas clases que se requieren para dar a una clase la información necesaria a fin de completar una responsabilidad. En general, una *colaboración* implica una solicitud de información o de cierta acción.

En la figura 6.11 se ilustra una tarjeta CRC índice sencilla para la clase **Plano**: la lista de responsabilidades en la tarjeta CRC es preliminar y está sujeta a agregados o modificaciones. Las clases **Pared** y **Cámar**a se anotan frente a la responsabilidad que requerirá su colaboración.

WebRef

En la dirección [www.theumlcafe.com/a0079.htm](http://www.theumlcafe.com/a0079.htm) hay un análisis excelente de estos tipos de clase.



Cita:

"Pueden clasificarse científicamente los objetos en tres grandes categorías: los que no funcionan, los que se descomponen y los que se pierden."

Russell Baker

- *Clases de entidad*, también llamadas *clases modelo o de negocio*, se extraen directamente del enunciado del problema (por ejemplo, **Plano** y **Sensor**). Es común que estas clases representen cosas almacenadas en una base de datos y que persistan mientras dure la aplicación (a menos que se eliminen en forma específica).
  - *Clases de frontera* se utilizan para crear la interfaz (por ejemplo, pantallas interactivas o reportes impresos) que el usuario mira y con la que interactúa cuando utiliza el software. Los objetos de entidad contienen información que es importante para los usuarios, pero no se muestran por sí mismos. Las clases de frontera se diseñan con la responsabilidad de administrar la forma en la que se presentan a los usuarios los objetos de entidad. Por ejemplo, una clase de frontera llamada **VentanadeCámara** tendría la responsabilidad de desplegar la salida de una cámara de vigilancia para el sistema *CasaSegura*.
  - *Clases de controlador* administran una “unidad de trabajo” [UML03] de principio a fin. Es decir, las clases de controlador están diseñadas para administrar 1) la creación o actualización de objetos de entidad, 2) las instancias de los objetos de frontera en tanto obtienen información de los objetos de entidad, 3) la comunicación compleja entre conjuntos de objetos y 4) la validación de datos comunicados entre objetos o entre el usuario y la aplicación. En general, las clases de controlador no se consideran hasta haber comenzado la actividad de diseño.

**Responsabilidades.** En las secciones 6.5.2 y 6.5.3 se presentaron los lineamientos básicos para identificar responsabilidades (atributos y operaciones). Wirfs-Brock *et al.* [Wir90] sugieren cinco lineamientos para asignar responsabilidades a las clases:



**SEP**  
responsabilidades a las  
clases?



CENTRO NACIONAL  
DE EVALUACIÓN PARA  
LA EDUCACIÓN SUPERIOR, A.C.



## **EDITORIAL:** **guiaceneval.mx**



**1. La inteligencia del sistema debe estar distribuida entre las clases para enfrentar mejor las necesidades del problema.**

Toda aplicación contiene cierto grado de inteligencia, es decir, lo que el sistema sabe y lo que puede hacer. Esta inteligencia se distribuye entre las clases de diferentes maneras. Las clases “tontas” (aquellas que tienen pocas responsabilidades) pueden modelarse para que actúen como subordinadas de ciertas clases “inteligentes” (las que tienen muchas responsabilidades). Aunque este enfoque hace directo el flujo del control en un sistema, tiene algunas desventajas: centraliza toda la inteligencia en pocas clases, lo que hace que sea más difícil hacer cambios, y tiende a que se requieran más clases y por ello más trabajo de desarrollo.

Si la inteligencia del sistema tiene una distribución más pareja entre las clases de una aplicación, cada objeto sabe algo, sólo hace unas cuantas cosas (que por lo general están bien identificadas) y la cohesión del sistema mejora.<sup>17</sup> Esto facilita el mantenimiento del software y reduce el efecto de los resultados colaterales del cambio.

Para determinar si la inteligencia del sistema está distribuida en forma apropiada, deben evaluarse las responsabilidades anotadas en cada modelo de tarjeta CRC índice a fin de definir si alguna clase tiene una lista demasiado larga de responsabilidades. Esto indica una concentración de inteligencia.<sup>18</sup> Además, las responsabilidades de cada clase deben tener el mismo nivel de abstracción. Por ejemplo, entre las operaciones enlistadas para una clase agregada llamada **RevisarCuenta**, un revisor anota dos responsabilidades: *hacer el balance de la cuenta y eliminar comprobaciones concluidas*. La primera operación (responsabilidad) implica un procedimiento matemático complejo y lógico.

La segunda es una simple actividad de oficina. Como estas dos operaciones no están en el mismo nivel de abstracción, *eliminar comprobaciones concluidas* debe colocarse dentro de las responsabilidades de **RevisarEntrada**, clase que está incluida en la clase agregada **RevisarCuenta**.

**2. La responsabilidad debe enunciarse del modo más general posible.** Este lema implica que las responsabilidades generales (tanto atributos como operaciones) deben residir en un nivel elevado de la jerarquía de clases (porque son generales y se aplicarán a todas las subclases).

**3. La acción y el comportamiento relacionado con ella deben residir dentro de la misma clase.** Esto logra el principio orientado a objetos llamado *encapsulamiento*. Los datos y los procesos que los manipulan deben empacarse como una unidad cohesiva.

**4. La información sobre una cosa debe localizarse con una sola clase, y no distribuirse a través de muchas.** Una sola clase debe tener la responsabilidad de almacenar y manipular un tipo específico de información. En general, esta responsabilidad no debe ser compartida por varias clases. Si la información está distribuida, es más difícil dar mantenimiento al software y más complicado someterlo a prueba.

**5. Cuando sea apropiado, las responsabilidades deben compartirse entre clases relacionadas.** Hay muchos casos en los que varios objetos relacionados deben tener el mismo comportamiento al mismo tiempo. Por ejemplo, considere un juego de video que deba tener en la pantalla las clases siguientes: **Jugador**, **CuerpodelJugador**, **BrazosdelJugador**, **PiernasdelJugador** y **CabezadelJugador**. Cada una de estas clases tiene sus propios atributos (como posición, orientación, color y velocidad) y todas deben actualizarse y desplegarse a medida que el usuario manipula una palanca de juego. Las res-

17 La cohesión es un concepto de diseño que se estudia en el capítulo 8.

18 En tales casos, puede ser necesario dividir la clase en una multiplicidad de ellas o completar subsistemas con el objeto de distribuir la inteligencia de un modo más eficaz.

ponsibilidades *actualizar()* y *desplegar()* deben, por tanto, ser compartidas por cada uno de los objetos mencionados. **Jugador** sabe cuando algo ha cambiado y requiere *actualizarse()*. Colabora con los demás objetos para obtener una nueva posición u orientación, pero cada objeto controla su propio despliegue en la pantalla.

**Colaboraciones.** Una clase cumple sus responsabilidades en una de dos formas: 1) usa sus propias operaciones para manipular sus propios atributos, con lo que satisface una responsabilidad particular o 2) colabora con otras clases. Wirfs-Brock *et al.* [Wir90] definen las colaboraciones del modo siguiente:

Las colaboraciones representan solicitudes que hace un cliente a un servidor para cumplir con sus responsabilidades. Una colaboración es la materialización del contrato entre el cliente y el servidor [...] Decimos que un objeto colabora con otro si, para cumplir una responsabilidad, necesita enviar al otro objeto cualesquier mensajes. Una sola colaboración fluye en una dirección: representa una solicitud del cliente al servidor. Desde el punto de vista del cliente, cada una de sus colaboraciones está asociada con una responsabilidad particular implementada por el servidor.

Las colaboraciones se identifican determinando si una clase puede cumplir cada responsabilidad. Si no es así, entonces necesita interactuar con otra clase. Ésa es una colaboración.

Como ejemplo, considere la función de seguridad de *CasaSegura*. Como parte del procedimiento de activación, el objeto **PaneldeControl** debe determinar si están abiertos algunos sensores. Se define una responsabilidad llamada *determinar-estado-delsensor()*. Si los sensores están abiertos, **PaneldeControl** debe fijar el atributo estado como "no está listo". La información del sensor se adquiere de cada objeto **Sensor**. Por tanto, la responsabilidad *determinar-estado-delsensor()* se cumple sólo si **PaneldeControl** trabaja en colaboración con **Sensor**.

Para ayudar a identificar a los colaboradores, se estudian tres relaciones generales diferentes entre las clases [Wir90]: 1) la relación *es-parte-de*, 2) la relación *tiene-conocimiento-de* y 3) la relación *depende-de*. En los párrafos siguientes se analizan brevemente cada una de estas tres responsabilidades generales.

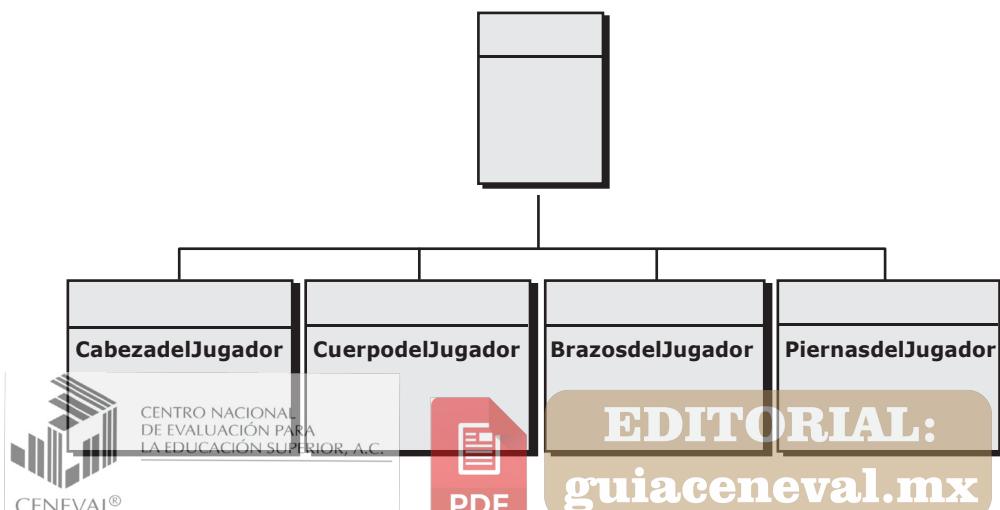
Todas las clases que forman parte de una clase agregada se conectan a ésta por medio de una relación *es-parte-de*. Considere las clases definidas por el juego mencionado antes, la clase **CuerpodelJugador** *es-parte-de* **Jugador**, igual que **BrazosdelJugador**, **PiernasdelJugador** y

**CabezadelJugador**. En UML, estas relaciones se representan como el agregado que se ilustra en la figura 6.12.

Cuando una clase debe adquirir información de otra, se establece la relación *tiene-conocimiento-de*. La responsabilidad *determinar-estado-delsensor()* ya mencionada es un ejemplo de ello.

FIGURA 6.12

Una clase agregada compuesta



La relación *depende-de* significa que dos clases tienen una dependencia que no se determina por *tiene-conocimiento-de* ni por *es-parte-de*. Por ejemplo, **CabezadelJugador** siempre debe estar conectada a **CuerpodelJugador** (a menos que el juego de video sea particularmente violento), pero cada objeto puede existir sin el conocimiento directo del otro. Un atributo del objeto **Cabeza del Jugador**, llamado posición-central, se determina a partir de la posición central de **Cuerpo del Jugador**. Esta información se obtiene por medio de un tercer objeto, **Jugador**, que la obtiene de **CuerpodelJugador**. Entonces, **CabezadelJugador** depende-de **CuerpodelJugador**.

En todos los casos, el nombre de la clase colaboradora se registra en el modelo de tarjeta CRC índice, junto a la responsabilidad que produce la colaboración. Por tanto, la tarjeta índice tiene una lista de responsabilidades y las colaboraciones correspondientes que hacen que se cumplan (véase la figura 6.11).

Cuando se ha desarrollado un modelo CRC completo, los participantes lo revisan con el empleo del enfoque siguiente [Amb95]:

1. Se da a todos los participantes que intervienen en la revisión (del modelo CRC) un subconjunto del modelo de tarjetas índice CRC. Deben separarse aquellas que colaboran (de modo que ningún revisor deba tener dos tarjetas que colaboren).
2. Todos los escenarios de casos de uso (y los diagramas correspondientes) deben organizarse en dos categorías.
3. El líder de la revisión lee el caso de uso en forma deliberada. Cuando llega a un objeto con nombre, entrega una ficha a la persona que tenga la tarjeta índice de la clase correspondiente. Por ejemplo, un caso de uso de *CasaSegura* contiene la narración siguiente:

El propietario observa el panel de control de *CasaSegura* para determinar si el sistema está listo para recibir una entrada. Si el sistema no está listo, el propietario debe cerrar físicamente las puertas y ventanas de modo que el indicador *listo* aparezca [un indicador *no está listo* implica que un sensor se encuentra abierto, es decir, que una puerta o ventana está abierta].

Cuando en la narración del caso de uso el líder de la revisión llega a “panel de control”, entrega la ficha a la persona que tiene la tarjeta índice **PaneldeControl**. La frase “implica que un sensor está abierto” requiere que la tarjeta índice contenga una responsabilidad que validará esta implicación (esto lo logra la responsabilidad *determinar-es-todo-delsensor()*). Junto a la responsabilidad, en la tarjeta índice se encuentra el **Sensor** colaborador. Entonces, la ficha pasa al objeto **Sensor**.

4. Cuando se pasa la ficha, se pide al poseedor de la tarjeta **Sensor** que describa las responsabilidades anotadas en la tarjeta. El grupo determina si una (o más) de las responsabilidades satisfacen el requerimiento del caso de uso.
5. Si las responsabilidades y colaboraciones anotadas en las tarjetas índice no se acoplan al caso de uso, éstas se modifican. Lo anterior tal vez incluya la definición de nuevas clases (y las tarjetas CRC índice correspondientes) o la especificación en las tarjetas existentes de responsabilidades o colaboraciones nuevas o revisadas.

PUNTO

## CLAVE

Una asociación define una relación entre clases. La multiplicidad define cuántas de una clase se relacionan con cuántas de otra clase.

Este modo de operar continúa hasta terminar el caso de uso. Cuando se han revisado todos los casos de uso, continúa el modelado de los requerimientos.

### Asociaciones y dependencias

En muchos casos, dos clases de análisis se relacionan de cierto modo con otra, en forma muy parecida a como dos objetos de datos se relacionan entre sí (véase la sección 6.4.3). En UML, estas relaciones se llaman asociaciones. A consultar la figura 6.10 la clase **Plano** se define con la identificación de un conjunto de asociaciones entre **Plano** y otras dos clases, **Cámara** y **Pa-**

## CASASEGURA



### Modelos CRC

**La escena:** Cubículo de Ed cuando comienza el modelado de los requerimientos.

**Participantes:** Vinod y Ed, miembros del equipo de ingeniería de software de *CasaSegura*.

#### La conversación:

[Vinod ha decidido enseñar a Ed con un ejemplo cómo desarrollar las tarjetas CRC.]

**Vinod:** Mientras tú trabajabas en la vigilancia y Jamie lo hacía con la seguridad, yo estaba en la función de administración del hogar.

**Ed:** ¿Cuál es el estado de eso? Mercadotecnia cambia lo que quiere a cada rato.

**Vinod:** Aquí está la primera versión de caso de uso para toda la función... la mejoramos un poco, pero debe darte el panorama general...

**Caso de uso:** Función de administración de *CasaSegura*.

**Narración:** Queremos usar la interfaz de administración del hogar en una PC o en una conexión de internet para controlar los dispositivos electrónicos que tengan controladores de interfaz inalámbrica. El sistema debe permitir encender y apagar focos específicos, controlar aparatos conectados a una interfaz inalámbrica y fijar el sistema de calefacción y aire acondicionado a la temperatura que desee. Para hacer esto, quiero seleccionar los aparatos en el plano de la casa. Cada equipo debe estar identificado en el plano. Como característica opcional, quiero controlar todos los equipos audiovisuales: sonido, televisión, DVD, grabadoras digitales, etcétera.

Con una sola selección, quiero preparar toda la casa para distintas situaciones. Una es *casa*, otra es *salir*, la tercera es *viaje nocturno* y la cuarta es *viaje largo*. Todas estas situaciones tienen especificaciones que se aplicarán a todos los equipos. En los estados de *viaje nocturno* y *viaje largo*, el sistema debe encender y apagar focos en momentos elegidos al azar (para que parezca que hay alguien en casa) y controlar el sistema de calefacción y aire acondicionado.

Debo poder hacer esta preparación por internet, con la protección de claves adecuadas...

**Ed:** ¿El personal de hardware ya tiene listas todas las interfaces inalámbricas?

**Vinod (sonríe):** Están trabajando en eso; dicen que no hay problema. De cualquier forma, obtuve muchas clases para la administración del hogar y podemos usar una como ejemplo. Tomemos la clase **InterfazdeAdministracióndelHogar**.

**Ed:** Bien... entonces, las responsabilidades son... los atributos y operaciones para la clase, y las colaboraciones son las clases que indican las responsabilidades.

**Vinod:** Pensé que no habías entendido el concepto CRC.

**Ed:** Un poco, quizás, pero continúa.

**Vinod:** Aquí está mi definición de la clase **InterfazdeAdministracióndelHogar**.

#### Atributos:

**PaneldeOpciones:** contiene información sobre los botones que permiten al usuario seleccionar funcionalidad.

**PaneldeSituación:** contiene información acerca de los botones que permiten que el usuario seleccione la situación.

**Plano:** igual que el objeto de vigilancia, pero éste muestra los equipos.

**ÍconosdeAparatos:** informa sobre los íconos que representan luces, aparatos, calefacción y aire acondicionado, etcétera.

**PaneldeAparatos:** simula el panel de control de un aparato o equipo; permite controlarlo.

#### Operaciones:

*DesplegarControl( ), SeleccionarControl( ), DesplegarSituación( ), SeleccionarSituación( ), AccederaPlano( ), SeleccionarÍcono deEquipo(), DesplegarPaneldeEquipo(), AccederaPaneldeEquipo(), ...*

**Clase:** InterfazdeAdministracióndelHogar

<b>Responsabilidad</b>	<b>Colaborador</b>
------------------------	--------------------

<i>DesplegarControl( )</i>	<b>PaneldeOpciones</b> (clase)
----------------------------	--------------------------------

<i>SeleccionarControl( )</i>	<b>PaneldeOpciones</b> (clase)
------------------------------	--------------------------------

<i>DesplegarSituación( )</i>	<b>PaneldeSituación</b> (clase)
------------------------------	---------------------------------

<i>SeleccionarSituación( )</i>	<b>PaneldeSituación</b> (clase)
--------------------------------	---------------------------------

<i>AccederaPlano( )</i>	<b>Plano</b> (clase) . . .
-------------------------	----------------------------

...

**Ed:** De modo que cuando se invoque a operación *AccederaPlano( )*, colabora con el objeto **Plano** de igual manera que el que desarrollamos para vigilancia. Espera, aquí tengo su descripción (ven la figura 6.10).

**Vinod:** Exactamente. Y si quisieramos revisar todo el modelo de la clase, podríamos comenzar con esta tarjeta índice, luego iríamos a la del colaborador y de ahí a una de los colaboradores del colaborador, y así sucesivamente.

**Ed:** Buena forma de encontrar omisiones o errores.

**Vinod:** Sí.



relación con la figura 6.10, un objeto **Pared** se construye a partir de uno o más objetos **SegmentodePared**. Además, el objeto **Pared** puede contener 0 o más objetos **Ventana** y 0 o más objetos **Puerta**. Estas restricciones de multiplicidad se ilustran en la figura 6.13, donde “uno o

FIGURA 6.13

## Multiplicidad

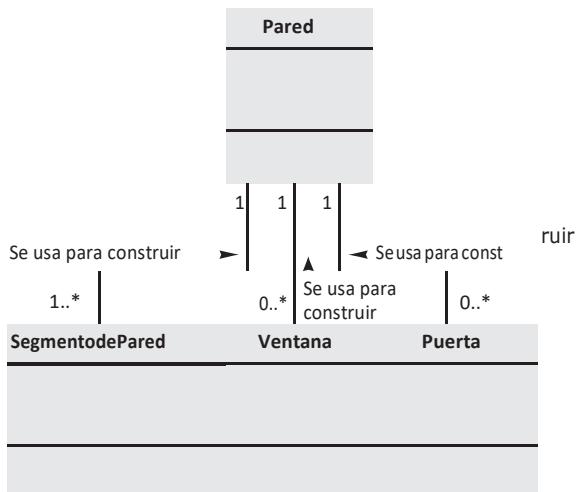
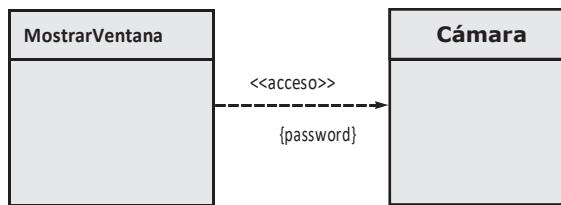


FIGURA 6.14

## Dependencias



más” se representa con 1...\*, y para “0 o más” se usa 0...\*. En LMU, el asterisco indica una frontera ilimitada en ese rango.<sup>19</sup>

Sucede con frecuencia que entre dos clases de análisis existe una relación cliente-servidor. En tales casos, una clase cliente depende de algún modo de la clase servidor, y se establece una *relación de dependencia*. Las dependencias están definidas por un estereotipo. Un *estereotipo* es un “mecanismo extensible” [Arl02] dentro del UML que permite definir un elemento especial de modelado con semántica y especialización determinadas. En UML, los estereotipos se representan entre paréntesis dobles angulares (por ejemplo, <<estereotipo>>).

Como ilustración de una dependencia simple dentro del sistema de vigilancia *CasaSegura*, un objeto **Cámara** (la clase servidora, en este caso) proporciona una imagen a un objeto **Mostrar- Ventana** (la clase cliente). La relación entre estos dos objetos no es una asociación simple sino de dependencia. En el caso de uso escrito para la vigilancia (que no se presenta aquí), debe darse una clave especial a fin de observar ubicaciones específicas de las cámaras. Una forma de lograr esto es hacer que **Cámara** pida un password y luego asegure el permiso a **MostrarVentana** para que presente el video. Esto se representa en la figura 6.14, donde <<acceso>> implica que el uso de la salida de cámara se controla con una clave especial.

¿Qué es un estereotipo?

### PUNTO CLAVE

Un paquete se utiliza para ensamblar un conjunto de clases relacionadas

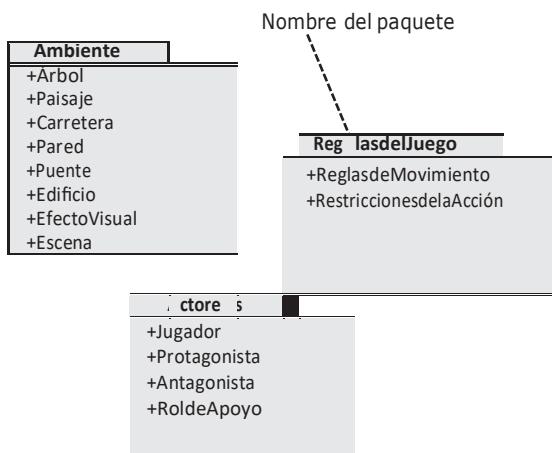
## Paquetes de análisis

Una parte importante del modelado del análisis es la categorización. Es decir, se clasifican distintos elementos del modelo de análisis (por ejemplo, casos de uso y clases de análisis) de ma-

19 Como parte de una asociación, pueden indicarse otras relaciones de multiplicidad: una a una, una a muchas, muchas a muchas, una a un rango específico con límites inferior y superior, y otras.



FIGURA  
Paquetes



nera que se agrupen en un paquete —llamado *paquete de análisis*— al que se da un nombre representativo.

Para ilustrar el uso de los paquetes de análisis, considere el juego de video que se mencionó antes. A medida que se desarrolla el modelo de análisis para el juego de video, se obtiene un gran número de clases. Algunas se centran en el ambiente del juego —las escenas visuales que el usuario ve cuando lo usa—. En esta categoría quedan clases tales como **Árbol**, **Paisaje**, **Ca-  
rretera**, **Pared**, **Puente**, **Edificio** y **EfectoVisual**. Otras se centran en los caracteres dentro del juego y describen sus características físicas, acciones y restricciones. Pueden definirse clases como **Jugador** (ya descrita), **Protagonista**, **Antagonista** y **RolesdeApoyo**. Otras más describen las reglas del juego —cómo se desplaza un jugador por el ambiente—. Candidatas para esto son clases como **ReglasdeMovimiento** y **RestriccionesdelaAcción**. Pueden existir muchas otras categorías. Estas clases se agrupan en los paquetes de análisis que se observan en la figura 6.15.

El signo *más* (suma) que precede al nombre de la clase de análisis en cada paquete, indica que las clases tienen visibilidad pública, por lo que son accesibles desde otros paquetes. Aunque no se aprecia en la figura, hay otros símbolos que preceden a un elemento dentro de un paquete. El signo *menos* (resta) indica que un elemento queda oculto desde todos los demás paquetes. Y el símbolo *#* señala que un elemento es accesible sólo para los paquetes contenidos dentro de un paquete dado.

## 6.6 RESUMEN

El objetivo del modelado de los requerimientos es crear varias representaciones que describan lo que necesita el cliente, establecer una base para generar un diseño de software y definir un conjunto de requerimientos que puedan ser validados una vez construido el software. El modelo de requerimientos cruza la brecha entre la representación del sistema que describe el sistema en su conjunto y la funcionalidad del negocio, y un diseño de software que describe la arquitectura de la aplicación del software, la interfaz de usuario y la estructura de componentes.

Los modelos basados en el escenario ilustran los requerimientos del software desde el punto de vista del usuario. El caso de uso —descripción, hecha con una narración o un formato, de una interacción entre un actor y el software— es el principal elemento del modelado. El caso de uso se obtiene durante la indagación de los requerimientos y define las etapas clave de una función o interacción específica. El grado de formalidad del caso de uso y su nivel de detalle varía, pero

el resultado final da las entradas necesarias a todas las demás actividades del modelado. Los escenarios también pueden ser descritos con el uso de un diagrama de actividades —representación gráfica parecida a un diagrama de flujo que ilustra el flujo del procesamiento dentro de un escenario específico—. Los diagramas de canal (swimlane) ilustran la forma en la que se asigna el flujo del procesamiento a distintos actores o clases.

El modelado de datos se utiliza para describir el espacio de información que será construido o manipulado por el software. El modelado de datos comienza con la representación de los objetos de datos —información compuesta que debe ser entendida por el software—. Se identifican los atributos de cada objeto de datos y se describen las relaciones entre estos objetos.

El modelado basado en clases utiliza información obtenida de los elementos del modelado basado en el escenario y en datos, para identificar las clases de análisis. Se emplea un análisis gramatical para obtener candidatas a clase, atributos y operaciones, a partir de narraciones basadas en texto. Se definen criterios para definir una clase. Para definir las relaciones entre clases, se emplean tarjetas índice clase-responsabilidad-colaborador. Además, se aplican varios elementos de la notación UML para definir jerarquías, relaciones, asociaciones, agregaciones y dependencias entre clases. Se emplean paquetes de análisis para clasificar y agrupar clases, de manera que sean más manejables en sistemas grandes.

### PROBLEMAS Y PUNTOS POR EVALUAR

¿Es posible comenzar a codificar de inmediato después de haber creado un modelo de análisis? Explique su respuesta y luego defienda el punto de vista contrario.

Una regla práctica del análisis es que el modelo “debe centrarse en los requerimientos visibles dentro del dominio del problema o negocio”. ¿Qué tipos de requerimientos no son visibles en dichos dominios? Dé algunos ejemplos.

¿Cuál es el propósito del análisis del dominio? ¿Cómo se relaciona con el concepto de patrones de requerimientos?

¿Es posible desarrollar un modelo de análisis eficaz sin desarrollar los cuatro elementos que aparecen en la figura 6.3? Explique su respuesta.

Se pide al lector que construya uno de los siguientes sistemas:

- Sistema de inscripción a la universidad basado en red.
- Sistema de procesamiento de órdenes basado en web para una tienda de computadoras.
- Sistema de facturación simple para un negocio pequeño.
- Libro de cocina basado en internet, construido en un horno eléctrico o de microondas.

Seleccione el sistema que le interese y desarrolle un diagrama entidad-relación que describa los objetos de datos, relaciones y atributos.

El departamento de obras públicas de una gran ciudad ha decidido desarrollar un sistema de seguimiento y reparación de baches, basado en web (SSRB).

Cuando se reportan los baches, se registran en “sistema de reparación del departamento de obras públicas” y se les asigna un número de identificación, almacenado según la calle, tamaño (en una escala de 1 a 10), ubicación (en medio, curveta, etc.), distrito (se determina con la dirección en la calle) y prioridad de reparación (determinada por el tamaño del bache). Los datos de la orden de trabajo se asocian con cada bache e incluyen su ubicación y tamaño, número de identificación del equipo de reparación, número de personas en dicho equipo, equipo asignado, horas dedicadas a la reparación, estado del bache (trabajo en proceso, reparado, reparación temporal, no reparado), cantidad de material de relleno utilizado y costo de la reparación (calculado a partir de las horas dedicadas, número de personas, materiales y equipo empleado). Por último, se crea un archivo de daños para mantener la información sobre daños reportados debido al bache, y se incluye el nombre y dirección del ciudadano, número telefónico, tipo de daño y cantidad de dinero por el daño. El SSRB es un sistema en línea, todas las solicitudes se harán en forma interactiva.



- a) Dibuje un diagrama UML para el caso de uso del sistema SSRB. Tendrá que hacer algunas suposiciones sobre la manera en la que un usuario interactúa con el sistema.
- b) Desarrolle un modelo de clase para el sistema SSRB.

Escriba un caso de uso basado en formato para el sistema de administración del hogar *CasaSegura* descrito de manera informal en el recuadro de la sección 6.5.4.

Desarrolle un conjunto completo de tarjetas índice de modelo CRC, sobre el producto o sistema que elija como parte del problema 6.5.

Revise con sus compañeros las tarjetas índice CRC. ¿Cuántas clases, responsabilidades y colaboradores adicionales fueron agregados como consecuencia de la revisión?

¿Qué es y cómo se usa un paquete de análisis?

### LECTURAS ADICIONALES Y FUENTES DE INFORMACIÓN

Los casos de uso son el fundamento de todos los enfoques del modelado de los requerimientos. El tema se analiza con amplitud en Rosenberg y Stephens (*Use Case Driven Object Modeling with UML: Theory and Practice*, Apress, 2007), Denny (*Succeeding with Use Cases: Working Smart to Deliver Quality*, Addison-Wesley, 2005), Alexander y Maiden (eds.) (*Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*, Wiley, 2004), Bittner y Spence (*Use Case Modeling*, Addison-Wesley, 2002), Cockburn [Coc01b] y en otras referencias mencionadas en los capítulos 5 y 6.

El modelado de datos constituye un método útil para examinar el espacio de información. Los libros de Hoberman [Hob06] y Simsion y Witt [Sim05] hacen tratamientos razonablemente amplios. Además, Allen y Terry (*Beginning Relational Data Modeling*, 2a. ed., Apress, 2005), Allen (*Data Modeling for Everyone*, Word Press, 2002), Teorey et al. (*Database Modeling and Design: Logical Design*, 4a. ed., Morgan Kaufmann, 2005) y Carlis y Maguire (*Mastering Data Modeling*, Addison-Wesley, 2000) presentan métodos de aprendizaje detallados para crear modelos de datos de calidad industrial. Un libro interesante escrito por Hay (*Data Modeling Patterns*, Dorset House, 1995) presenta patrones comunes de modelos de datos que se encuentran en muchos negocios diferentes.

Análisis de técnicas de modelado UML que pueden aplicarse tanto para el análisis como para el diseño se encuentran en O'Docherty (*Object-Oriented Analysis and Design: Understanding System Development with UML 2.0*, Wiley, 2005), Arlow y Neustadt (*UML, 2 and the Unified Process*, 2a. ed., Addison-Wesley, 2005), Roques (*UML in Practice*, Wiley, 2004), Dennis et al. (*Systems Analysis and Design with UML Version 2.0*, Wiley, 2004), Larman (*Applying UML and Patterns*, 2a. ed., Prentice-Hall, 2001) y Rosenberg y Scott (*Use Case Driven Object Modeling with UML*, Addison-Wesley, 1999).

En internet existe una amplia variedad de fuentes de información sobre el modelado de requerimientos. En el sitio web del libro, en [www.mhhe.com/engcs/comsci/pressman/professional/olc/ser.htm](http://www.mhhe.com/engcs/comsci/pressman/professional/olc/ser.htm), se halla una lista actualizada de referencias en web que son relevantes para el modelado del análisis.

## Lectura 5. Diseño e implementación 1



## Capítulo 4      Diseño e implementación

### Diseño

El diseño es el primer paso en la fase de desarrollo de cualquier producto o sistema de ingeniería. El objetivo del diseño es producir un modelo o representación de una entidad que se va a construir posteriormente [Pressman, 1998].

Hay tres características que sirven como parámetros generales para la evaluación de un buen diseño [McGlaughlin, 1991]. Estos parámetros son los siguientes:

1. El diseño debe implementar todos los requisitos explícitos obtenidos en la etapa de análisis.
2. El diseño debe ser una guía que puedan leer y entender los que construyen el código y los que prueban y mantienen el software.
3. El diseño debe proporcionar una idea completa de lo que es el software.

El diseño es la primera de las tres actividades técnicas que implica un proceso de ingeniería de software; estas etapas son diseño, codificación (en el caso de este proyecto Desarrollo e Implementación) y pruebas [Pressman, 1998]. Generalmente la fase de diseño produce:

- **Diseño de datos** esencialmente se encarga de transformar el modelo de dominio de la información creado durante el análisis. Para el tipo de datos manejado en este proyecto es de suma importancia este diseño, ya que de él dependerá el buen funcionamiento de la aplicación. Éste diseño está basado en el esquema entidad- relación obtenido en análisis, del cual se originan las tablas que serán utilizadas por ésta herramienta computacional.
- **Diseño arquitectónico** se definen las relaciones entre los principales elementos estructurales del programa. Para un software empresarial, como el que se va a desarrollar e implementar, es importante tomar en cuenta esta fase de diseño, dado

que en esta representación se establece la estructura modular del software que se desarrolla, tipo de arquitectura, tipo de conexión entre la aplicación y el DBMS, el manejo de reportes, entre otros. La figura 4.1 muestra la arquitectura del sistema.

- **Diseño de interfaz** describe cómo se comunica el software con los sistemas que operan con él, y con los operadores que lo emplean. En el caso de la herramienta de software propuesta la comunicación que se genera con los operadores es a través de ambientes gráficos que llevan de la mano al usuario para que pueda realizar sus tareas cotidianas. La comunicación con otros sistemas, DBMS y Crystal Reports, se realiza a través de mecanismos de acceso a base de datos y componentes VCL (Visual Component Library).
- **Diseño procedural** transforma elementos estructurales de la arquitectura del programa en una descripción procedural de los componentes del software. Para el desarrollo de esta aplicación es importante éste diseño, debido a que del análisis previo se obtuvieron los módulos a desarrollar y esta parte definirá los pasos a seguir en cada uno de los procesos y funciones que formarán cada módulo, así como los procesos necesarios para conectar la aplicación con el DBMS y Crystal Reports.

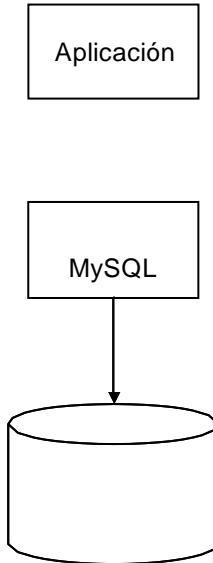


Figura 4.1 Esquema general



En el diseño de estructura y comportamiento de objetos los siguientes componentes son definidos:

- ¿Qué clases serán implementadas? Los tipos de objetos de AEO guiarán esta decisión.
- ¿Qué estructura de datos buscará una clase? La estructura de datos puede ser representada a través de un diagrama.
- ¿Qué operaciones ofrecerá cada clase y cuáles serían sus métodos? Las operaciones son listadas y eventualmente sus métodos son especificados.
- ¿Cómo serán implementadas las clases con herencia y cómo cambiarán los datos y las especificaciones de las operaciones?
- ¿Qué variantes existen? Se identifican variantes entre clases (“Igual que, excepto”)

En el apéndice C se encontrarán los diagramas que ilustran los puntos definidos anteriormente en cuanto al Diseño OO.

#### **4.1.1 Modelo de comportamiento de workflows**

Para el diseño de workflows es necesario comprender el modelado de comportamiento, el cual describe los pasos que se siguen durante la ejecución del workflow y que caracterizan su comportamiento. Esta descripción incluye el momento de disparo de cada una de las actividades, qué acciones se deben de realizar en caso de error y el criterio para la elección de agentes.

Se distinguen tres tipos de disparadores de tareas: por iniciativa de un agente, esto es, una tarea disparada por un empleado; por un evento externo, como la recepción de un mensaje; o por una señal de tiempo, dígase el procesamiento de una lista de órdenes comienza a las 6:00 en punto.

## **4.2 Reingeniería**

Como ya se mencionó la solución del Grupo Pro esta desarrollada en Excel, sus componentes gráficos son muy pobres y las funcionalidades no son las más adecuadas para las actividades de la empresa.

Actualmente los sistemas computacionales, en su mayoría, están diseñados con ventanas y formas que hagan más fácil su manejo y acceso a las funciones. Normalmente éstas ventanas están compuestas de un conjunto de componentes gráficos, como son los campos de texto, botones, etiquetas, checklist, entre otros; y que tienen la característica de que el usuario puede controlarlos con la ayuda del mouse sin excluir al teclado.

Dados estos cambios en el desarrollo y diseño a través del tiempo, nuestra propuesta es el desarrollo de un sistema que sustituya la solución en Excel, haciendo un proceso de reingeniería de la solución, usando herramientas actuales y adaptables para los usuarios finales. Para este fin se ha tomado en cuenta las operaciones básicas realizadas por la solución de Excel, aunque para el nuevo sistema se realizó un nuevo análisis, haciendo modificaciones en el manejo de algunas de las operaciones y desarrollando nuevas funcionalidades.

El código desarrollado en el proyecto es totalmente nuevo, solo se tomó en cuenta la forma de operación de la empresa para el desarrollo de los mismos. Para la creación de las interfaces gráficas se tomó como guía la manera en que Excel presentaba los campos de ingreso de datos, para que al usuario final no le fuera tan difícil el adaptarse a este nuevo sistema; sin embargo en algunos casos fue necesario hacer un diseño totalmente nuevo debido a que no se encontraba nada desarrollado en Excel.

Como consecuencia de las restricciones de Excel, la información almacenada en sus libros contenía incongruencia, errores redundancias. Esto llevó a hacer una reingeniería en cuanto a que se tomó la información valiosa y se desechó la redundancia, sin alterar la gramática de sus datos.

## 4.2.1 Herramientas utilizadas

### Delphi

Para la codificación de SIC se decidió utilizar Delphi, en su versión 7, debido a que cuenta con varios mecanismos de acceso a datos lo cual permite tener conexión con varios manejadores de datos, Oracle, Inter Base, MySQL, entre otros; la manipulación de los datos



es ágil, se genera un único ejecutable, es flexible, de fácil manejo, por otro lado es una herramienta RAD (Rapid Application Development). Se debe aclarar que Delphi es una herramienta de desarrollo, no un lenguaje de programación; Object Pascal es el lenguaje que utiliza Delphi, el cual es una evolución de Pascal.

## **MySQL**

Como manejador de base de datos se empleó MySQL, ya que es un sistema gratuito que se encuentra disponible a través de Internet; cuenta con lo necesario para poder llevar a cabo un buen funcionamiento del sistema y es aceptable para los fines del mismo.

Las razones para escoger MySQL como solución de misión crítica para la administración de datos son [Gilfillan, 2003]:

- **Coste:** el coste de MySQL es gratuito para la mayor parte de los usos y su servicio de asistencia resulta económico
- **Asistencia:** MySQL AB ofrece contratos de asistencia a precios razonables y existe una nutrida y activa comunidad MySQL
- **Velocidad:** MySQL es mucho más rápido que la mayor parte de sus rivales
- **Funcionalidad:** MySQL dispone de muchas de las funciones que exigen los desarrolladores profesionales, como compatibilidad completa con ACID, compatibilidad para la mayor parte de SQL ANSI, volcados online, duplicación, funciones SSL e integración con la mayor parte de los entornos de programación. Así mismos, se desarrolla y actualiza de forma mucho más rápida que muchos de sus rivales, por lo que prácticamente todas las funciones estándar de MySQL todavía no están en fase de desarrollo
- **Portabilidad:** MySQL se ejecuta en la inmensa mayoría de sistemas operativos y, la mayor parte de los casos, los datos se pueden transferir de un sistema a otro sin dificultad.

## **Crystal Reports**

Para el desarrollo de los reportes que se utilizaron en la aplicación se utilizó el reporteador Crystal Reports en su versión 10, debido a que es uno de los más poderosos reporteadores

disponibles con la habilidad de obtener datos de cualquier fuente de datos [Taylor, 2004]. Puede utilizarse Crystal Reports para generar informes de cualquiera de los programas de base de datos estándares para PC; además tiene un poderoso servidor de informes Web, que permite distribuir los reportes a través del Web [Harvest, 2004].

## Implementación

La figura 4.2 muestra la arquitectura del sistema desarrollado en este proyecto.

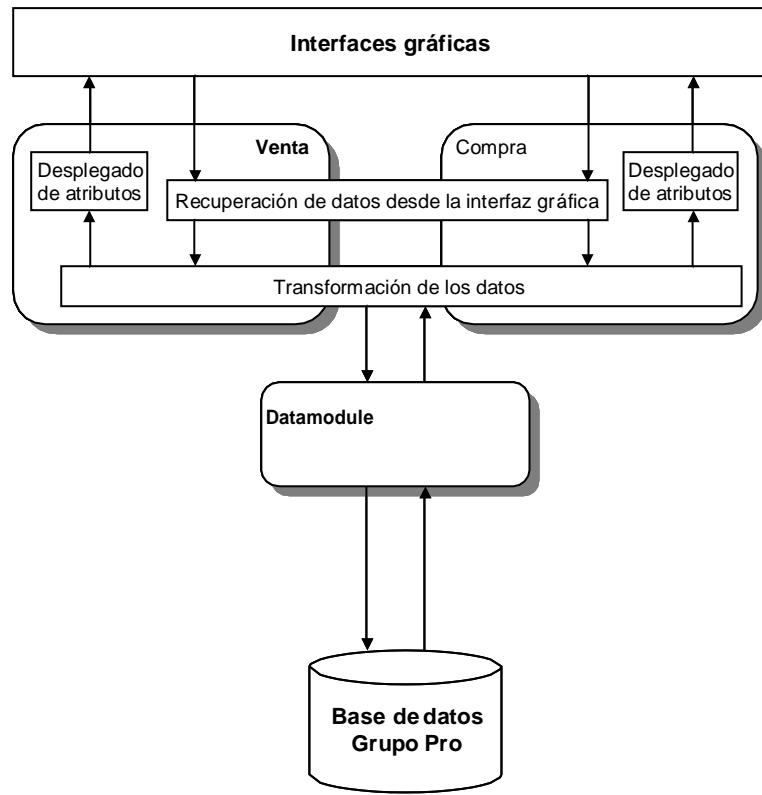


Figura 4.2 Arquitectura del sistema

Para la implementación se organizaron los módulos por secciones, de acuerdo a su funcionalidad; esto quiere decir que se agruparon las funciones del software en conjuntos bien definidos de acuerdo a los resultados que ofrece cada una de éstas. Las divisiones se presentan en el software como menús, en donde cada uno agrupa un número de funciones.

En cuanto a la interfaz, los menús principales son las categorías de los procesos que surgieron en el análisis (compras, ventas, inventario, producción, administración del



sistema y catálogos). A su vez cada uno de estos menús principales está dividido en submenús, los cuales especifican las tareas que pueden ser realizadas según la función del menú principal. MainForm es la ventana principal; es donde se encuentra el menú desde el cual se manda a llamar a todas las demás ventanas de los submenús Figura 4.3.



Figura 4.3 Pantalla principal

Es importante tener presente cada una de las categorías con las funciones que realiza, por lo que a continuación la tabla 4.1 muestra la estructura final de cada menú principal y sus submenús.

<b>Compras</b>	Orden de compra Registro de Compra Pago a proveedores
Cancelaciones	
Cancelación total/parcial de orden de compra	Notas de crédito
Notas de cargo	
<b>Ventas</b>	Pedido de venta.
Facturas	
Pago de clientes	Cancelaciones
Cancelación total/parcial de pedidos	Cancelación total de factura
Notas de venta	Notas de crédito
<hr/>	
	Notas de cargo

---

<b>Inventarios</b>	Traspasos
Facturación de traspasos	Movimientos varios
<b>Producción</b>	Manejo de fórmulas Alta de fórmulas Baja de fórmulas
Modificación de fórmulas	Orden de producción
	Cierre de orden de producción
<b>Administración del sistema</b>	Permisos de usuarios Usuarios Alta de usuario Baja de usuario Modificación de usuario Consulta de usuario Compañías Alta de compañía Modificación de compañía Consulta de compañía Almacén Alta de almacén Modificación de almacén Consulta de almacén Respaldo
<b>Catálogos</b>	Tipo de cambio Clientes
Alta de cliente Baja de cliente	
Modificación de cliente Consulta de cliente	
Proveedores	
Alta de proveedor Baja de proveedor	
Modificación de proveedor Consulta de proveedor	
Vendedores	
Alta de vendedor Baja de vendedor	
Modificación de vendedor Productos	
Alta de producto	
Modificación de producto Consulta de producto	
Familia	
Alta de familia	
Modificación de familia Consulta de familia	
Color	
Alta de color Modificación de color Consulta de color	
Clasificación	
Alta de clasificación Modificación de clasificación Consulta de clasificación	
Línea	

---



---

Alta de línea	
Modificación de línea	Consulta de línea
Marca	
Alta de marca	Modificación de marca
<b>Consulta de marca</b>	

---

Tabla 4.1 Organización del menú

Cabe mencionar que cada submenú del menú principal es una unidad; donde cada una de ellas realiza ciertas operaciones dependiendo de la actividad que se desea realizar, pedido, facturación, pagos, entre otros.

Delphi utiliza unidades, las cuales tiene una estructura fundamental para su funcionamiento. Esta estructura se muestra a continuación [Cantú, 2003]:

1. Palabra reservada **unit** seguida del nombre de la unidad
2. La palabra reservada **interface**, la cual continúa hasta el inicio de la sección **implementation**. En esta sección se declaran las constantes, tipos, variables, procedimientos y funciones, las cuales estarán disponibles para los clientes, esto es, para otras unidades o programas que deseen usar los elementos de esta sección. Estas entidades son llamadas públicas, debido a que un cliente puede acceder a ellas, como si éstas estuvieran declaradas en el mismo cliente. La declaración de interfaz de un procedimiento o función incluye sólo el título de la rutina. Esta sección puede incluir su propia cláusula **uses**, la cual debe aparecer inmediatamente después de la palabra **interface**.
3. La cláusula **uses** consiste de la palabra reservada **uses**, seguida de una o más nombres de unidades delimitados por comas, seguidos de un punto y coma (;). En la mayoría de los casos, todas las unidades necesarias son colocadas en la cláusula **uses** cuando se genera el proyecto y mantiene un archivo de origen.
4. Palabra reservada **implementation**, la cual continúa hasta el inicio de la sección **initialization** o, si no existe esta sección, hasta el fin de la unidad. Esta sección define los procesos y funciones que se han declarado en la sección **interface**. Adicionalmente a la definición de procesos públicos y funciones, se pueden declarar constantes, tipos (incluyendo clases), variables, procedimientos, y las

funciones que son privadas, es decir, inaccesibles a los clientes. Además puede contener su propia cláusula **uses**, la cuál debe aparecer inmediatamente después de la palabra **implementation**.

5. Para hacer comentarios en Delphi se puede hacer de diferentes formas:  
“//” para comentar una línea y “{ }” para comentar una línea o un bloque.

En el ejemplo siguiente se muestra el uso de la estructura de las unidades de Delphi.

```
unit usuario_inicio;
interface
  uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs,
    StdCtrls, ExtCtrls, ComCtrls, Mask;

  type
    Tinicio = class (TForm)
      usuario:
        TLabel;
      aceptarbtn: TButton;
    procedure aceptarbtnClick(Sender: TObject);
  private
    { Private declarations }
  public
    nombre,compania, usuarios: string;
    { Public declarations }
  end
  ; var
    inicio: Tinicio;

  implementation
  uses MAIN, datamodule;

  procedure Tinicio.aceptarbtnClick(Sender: TObject);
  begin
    // implementación del proceso.
  end
  ;
end
```

Figura 4.4 Unidad de Delphi

Las unidades Datamodule y Rutinas son utilizadas por la mayoría de las unidades, debido a que contienen funciones y procesos necesarios ya sea, para el manejo de la información o, acciones recurrentes, por lo tanto es conveniente mencionar su funcionamiento.



La unidad Rutinas, cuenta con los procesos y funciones, como formato de moneda, el cual es requerido para la realización de ciertas operaciones comerciales.

La unidad Datamodule cuenta con los componentes necesarios para la conexión a la base de datos y para la recuperación de la información. A través de esta unidad las demás unidades podrán recuperar, manipular y actualizar la información. Ésta unidad utiliza los componentes del mecanismo de acceso a bases de datos DBX (**dbExpress**) de Delphi para la conexión con MySQL, debido a que está basado en la arquitectura cliente/servidor, sus controladores son mucho más sencillos y eficientes que los controladores BDE (Borland Database Engine) o ADO (Active Data Objects), es una solución multiplataforma, que permite hacer la conexión con el servidor, recuperar datos, manipularlos y efectuar actualizaciones [Charte, 2003]. Los componentes empleados para dicha conexión son:

- **TSQLConnection.** Para establecer la conexión con una base de datos. Éste componente precisa tres datos fundamentales: el nombre del controlador, el nombre de la función de entrada ha dicho controlador y el nombre de la biblioteca del cliente de la base de datos.
- **TSQLQuery.** Para establecer los querys necesarios para la recuperación del conjunto de datos.

Para lograr un mejor entendimiento de las funciones del sistema, la explicación de los componentes de estos será dividida según las operaciones comerciales que se realizan, y que podemos dividirlas en los grandes grupos: Compras Ventas, Manejo de Inventario, Productos, administración del sistema y los procesos que se modelaron como workflows.

## Venta

El proceso de venta que se lleva a cabo en las compañías del Grupo Pro se debe componer de la siguiente manera.

## Pedido

Este proceso comprende el listado de los productos que un cliente desea comprar, o bien ordenar producir. En el pedido se incluyen las claves de los productos y sus cantidades, a parte de los datos del cliente y un presupuesto aproximado del total.

Para la implementación de este proceso se creó el objeto Pedido (mostrado desde el análisis). Este objeto contiene los datos de lo que podemos llamar el encabezado del pedido. Por otro lado tenemos el objeto Línea\_Pedido el cuál define, principalmente, la clave del producto, sus características, su cantidad surtida, por surtir y precio unitario.

Los métodos del objeto Pedido son: alta de pedido, cancelación total o parcial de pedido, obtención de la moneda en la que se presupuestó el pedido y cálculo del importe total.

El objeto Línea\_Pedido contiene métodos que permiten la agregación, eliminación y modificación de cada una de ellas. También contiene métodos de validación de cantidad y de precio, así como el cálculo del importe por línea.

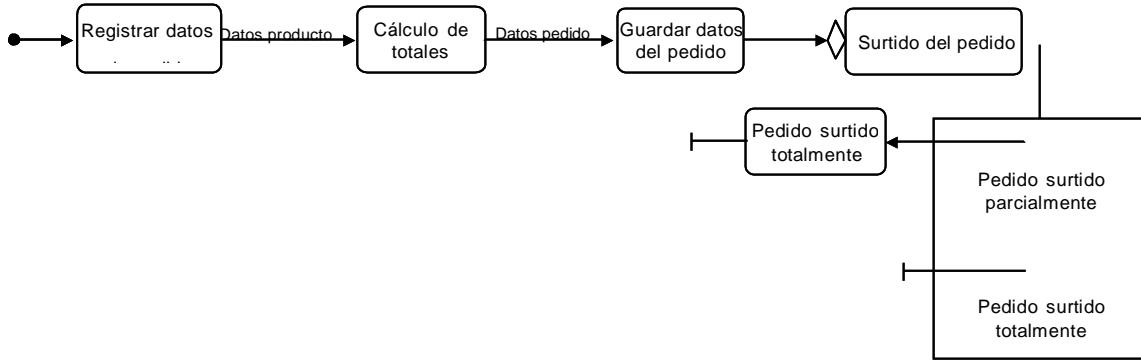


Figura 4.5 Diagrama de transición de estados del pedido

## Factura

Especifica la materialización de la venta. Se debe mencionar que el proceso de facturación se puede realizar de dos formas: (1) La facturación directa de un pedido; (2) Facturación de



productos que no se encuentran en el catálogo y no atañen al giro de la empresa, dígase venta de mobiliario.

El objeto Factura contiene los datos generales del cliente, así como los datos indispensables que permiten la generación de la factura. También se tiene el objeto Línea\_Factura, el cuál contiene los datos necesarios de los productos vendidos al cliente.

Los métodos concernientes al objeto Factura son: alta de factura, cancelación total de factura, cálculo del importe total. Debe verificar la fecha que se indica en la elaboración de la factura, pues existe la posibilidad de que ésta sea anterior o posterior al día del movimiento, lo cual conlleva a manejo de autorizaciones. Partiendo de la fecha indicada, se debe obtener el tipo de cambio que será aplicado. En caso de que ésta provenga de un pedido, los datos del cliente deben ser obtenidos de éste; si se factura un producto ajeno al giro comercial de consorcio, los datos pueden ser obtenidos o simplemente ingresados por el usuario. Línea\_Factura puede ser editada y agregada.

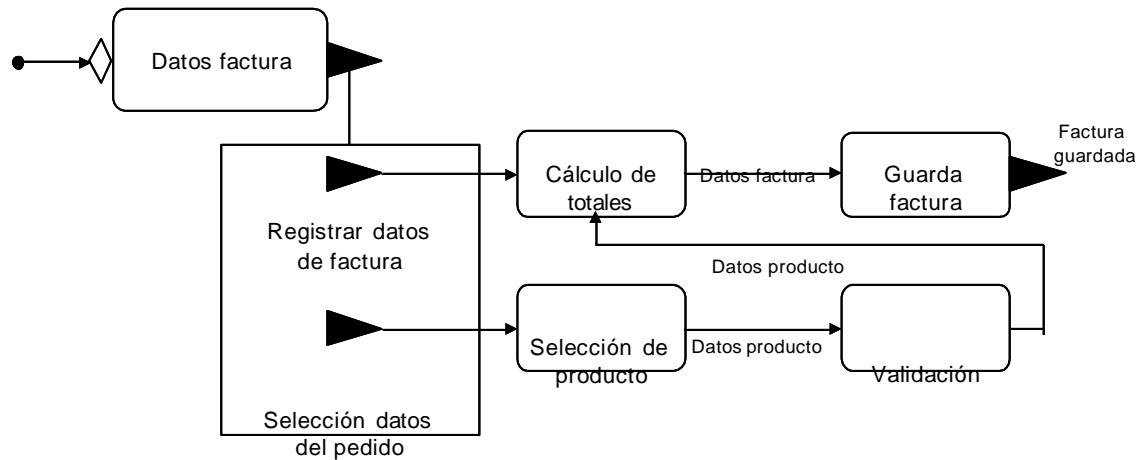


Figura 4.6 Diagrama de transición de estados de la factura

### Nota de venta

Este objeto y su objeto asociado, Línea\_Notaventa, tiene un comportamiento equivalente a Factura y Línea\_Factura, con la diferencia comercial de que ésta no tiene implicaciones fiscales ni muchos menos está asociada a algún pedido. Son ventas de mostrador.

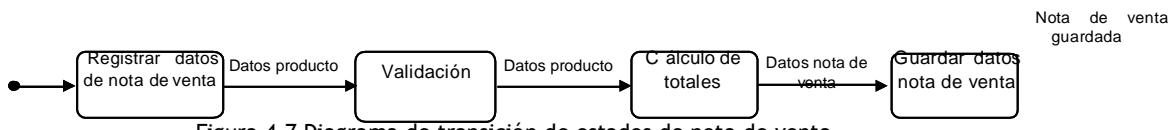


Figura 4.7 Diagrama de transición de estados de nota de venta

### Nota de cargo

Este objeto se utiliza cuando, en la elaboración de una factura, hubo un error humano en el ingreso de los precios, siendo estos menores a los correctos. Esto implica generar un documento en el que se especifique la diferencia a pagar por el cliente. En este objeto también se maneja un encabezado que contiene los datos de la factura asociada. El objeto Línea\_NotaCargo indica los atributos obtenidos del objeto Línea\_Factura a los cuales se aplicará el cargo.

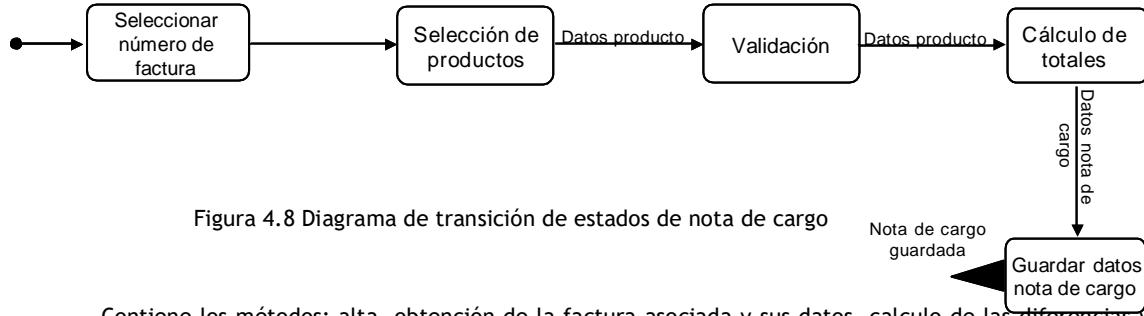


Figura 4.8 Diagrama de transición de estados de nota de cargo

Contiene los métodos: alta, obtención de la factura asociada y sus datos, cálculo de las diferencias y totales. Línea\_NotaCargo obtiene los datos de Línea\_Factura de la factura asociada y el cálculo de los importes por línea.

### Nota de crédito

Este documento puede ser generado por dos causas: (1) la devolución de productos por parte del cliente; (2) la factura fue generada por un importe mayor al debido. Estas situaciones llevan a otorgar al cliente un crédito.

Contiene los métodos: alta, obtención de la factura asociada y sus datos, cálculo de las diferencias y totales. Línea\_NotaCrédito obtiene los datos de Línea\_Factura de la factura asociada y el cálculo de los importes por línea.





Figura 4.9 Diagrama de transición de estados de nota de crédito

## **Cliente**

Son las personas a las cuales el consorcio realiza la venta de productos, contiene atributos para la identificación y manejo de los mismos. Este objeto está asociado a los objetos de Venta. Tiene los métodos principales alta, baja, modificación, consulta y pagos. Estos últimos se efectúan sobre las notas de venta y facturas, y pueden estar afectados por las Notas de Cargo y Crédito respectivas. Además existen métodos auxiliares como: verificación de saldos y de RFC.

## **Vendedor**

Es la persona asignada a la atención de cliente, cada cliente tiene asignado un único vendedor. Para el momento de desarrollo de este proyecto el manejo de la información de los vendedores no es de gran relevancia, pero servirá de base para el desarrollo posterior del proceso de cálculo de comisiones.

## **Compra**

El proceso de compra que se lleva a cabo en las compañías del Grupo Pro se debe componer de la siguiente manera.

## **Orden de compra**

Es un listado de productos requeridos para compra seleccionados según la existencia física actual, los cuales serán solicitados a un proveedor. Tiene una estructura maestro-detalle; donde el maestro tiene los datos del proveedor y características de la orden, y el detalle contiene los productos así como sus características y cantidades a solicitar. Como métodos

se encuentran el alta de la orden, registro de la entrega de los productos y sus cantidades, cancelación de la orden, el cálculo del total de la orden. Linea\_OrdenCompra contiene los métodos agrega, edita y elimina línea, verifica cantidad y precio, así como el cálculo del importe por línea.

### **Nota de crédito y cargo**

Son objetos cuyo comportamiento es similar al de Nota de Crédito y Cargo de ventas, con la diferencia esencial de que afectan los cobros que hace el proveedor a la compañía.

### **Proveedor**

Son mas personas o empresas a las cuales se les solicitan productos y materias primas. Este objeto contiene atributos para la identificación y manejo de los proveedores; por su parte estos están asociados a los objetos involucrados en las compras. Sus métodos principales son alta, baja, modificación, consulta y pagos. Los pagos se efectúan una vez que han sido registrados los productos, las cantidades a pagar se ven afectadas por las notas de crédito y cargo respectivas. Por otro lado existen métodos secundarios como: verificación de saldos y de RFC.

### **Inventario**

Tiene los atributos correspondientes a cada producto involucrado en un movimiento de inventario, además posee datos que identifican y permiten manipular los movimientos de inventario. Los métodos contenidos en inventario son: Movimientos de entrada y salida, traspasos intercompañías y traspasos entre almacenes, mismos que están conformados por movimientos de entrada y salida, verificación de existencias por producto, el cual será detallado a fondo en la implementación de workflows.

El proceso de traspasos en el caso de ser intercompañías se realiza de la siguiente manera: primero se selecciona el producto a traspasar de cualquiera de los almacenes disponibles, posteriormente se hace la selección de la empresa a la cual se desea traspasar, el almacén al que se desea ingresar, y selección de la clave correspondiente en la empresa destino. Una vez echa la selección se procede a sacar el almacén seleccionado el producto



para ingresarla al almacén 9 de la empresa de salida, para posteriormente pasar el producto al almacén seleccionado de la empresa destino.

En el caso de que el proceso de traspaso sea entre almacenes se procede a seleccionar el almacén de salida, el producto de salida, el almacén de entrada y la clave del producto con la cual se ingresará al almacén deseado. Este traspaso puede ser utilizado cuando se hace la conversión de las unidades en las que se vende un producto.

### **Producto**

Contiene los atributos correspondientes a los productos que son manejados por el consorcio. Tiene los métodos alta, modificación, consulta, verificación de contratipos, verificación de características, orden y cierre de producción, así como el alta, baja, modificación y consulta de fórmulas.

### **Administración del sistema**

#### **Usuario**

Son las personas que manejarán la aplicación, cuenta con los atributos necesarios para su identificación, asignación y restricción en el uso de los diferentes procesos. Tiene los métodos alta, baja, modificación, consulta y verificación de permisos. El método verificación de permisos funciona utilizando álgebra booleana, pues ésta permite el fácil “encendido” y “apagado” de los permisos. La operación AND permite la activación de los procesos; la operación OR nos ayuda a armar un número decimal que es tratado como binario, el cual indica que permisos tiene disponibles el usuario.

#### **Almacén**

Contiene la descripción de los almacenes disponibles. Tiene los métodos alta, modificación y consulta.

## **Compañía**

Contiene los datos referentes a las compañías que conforman el consorcio. Tiene los métodos: alta, modificación y consulta.

## **Implementación de workflows**

Para el desarrollo de los procesos verificación de existencia de productos, pagos pendientes de clientes y a proveedores, generación de archivo de contabilidad primero se especificarán los componentes necesarios para su funcionamiento, en base al modelado del comportamiento de workflows.

### **Verificación de existencias .**

Este workflow es el encargado de, periódicamente, revisar las existencias de los productos a una fecha determinada.

Para esto es necesario realizar las siguientes tareas:

1. Calcular las existencias físicas en inventario de cada uno de los productos.
2. Obtener, de las órdenes de compra, qué productos y en qué cantidad, serán supuestamente recibidos a una fecha específica. La entrega de los productos se ve afectada por la puntualidad de los proveedores, es por eso que se toma un supuesto de las cantidades a recibir.
3. Verificar, de los pedidos de clientes, qué productos y en qué cantidad deben ser entregados en el periodo analizado.
4. Finalmente se debe obtener los productos y las cantidades de éstos que se están produciendo, para saber si se puede contar o no con ellos al día que se está consultando.
5. Ya teniendo estas cantidades se procede a realizar la siguiente operación: Existencia en inventario + cantidad a recibir de un proveedor + cantidad a obtener de producción - cantidad a entregar en los pedidos. Esta operación nos indica la existencia supuesta global a una fecha.



- Finalmente se compara la existencia supuesta global con la cantidad mínima requerida en inventario para obtener un listado de los productos que deben ser, ya sea producidos, o bien, solicitados a los proveedores.

Las 4 primeras tareas pueden realizarse de manera paralela. Al tener los resultados de todas ellas, se procede a las últimas 2 de manera secuencial.

Este proceso, o workflow, va a ejecutarse en el primer ingreso al sistema en el día por parte del usuario correspondiente, esto es, aquel que tiene los permisos necesarios para visualizar esta información. Aquí podemos ver un ejemplo de la definición de roles requerida en el modelado de workflows.

En caso de error, se notificará al usuario el estatus del proceso y la causa del problema.

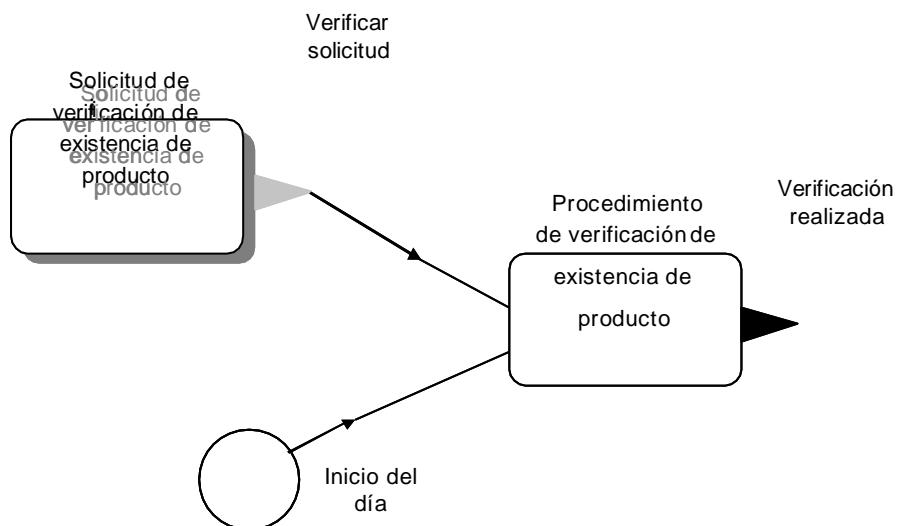


Figura 4.10 Diagrama de transición de estados de verificación de existencias

### **Pagos pendientes de clientes y a proveedores**

Primero se debe aclarar que estos procesos son muy similares en cuanto a funcionamiento, su diferencia consiste en la información que maneja. En el control de pagos pendientes de clientes la información es obtenida de las tablas Factura y PagoCliente. En el caso de los pagos pendientes a proveedores se toman en cuenta las tablas Registro de Compra y PagoProveedor.

El control del crédito en los pagos de las facturas se efectúa de la siguiente manera:

1. Se deben obtener las facturas (o registros de compras) que no han sido saldadas en su totalidad.
2. Se verifica el tipo de crédito que tienen los clientes relacionados a las facturas obtenidas de la tarea anterior.
3. Se calculan los días transcurridos desde la realización de la factura hasta la fecha en la que se ejecuta el proceso.
4. Se comparan los días de crédito asignados con los días transcurridos.
5. Se presenta un reporte de las facturas (o registros de compras) a las que les restan 3 días de crédito y todas aquellas que ya son vencidas.

La sucesión de las acciones en este proceso son totalmente secuenciales. Este workflow se accionará, de la misma manera que el anterior, en el primer ingreso al sistema del usuario correspondiente.

En caso de error, el usuario visualizará un mensaje en el que se indica la razón de la falla.

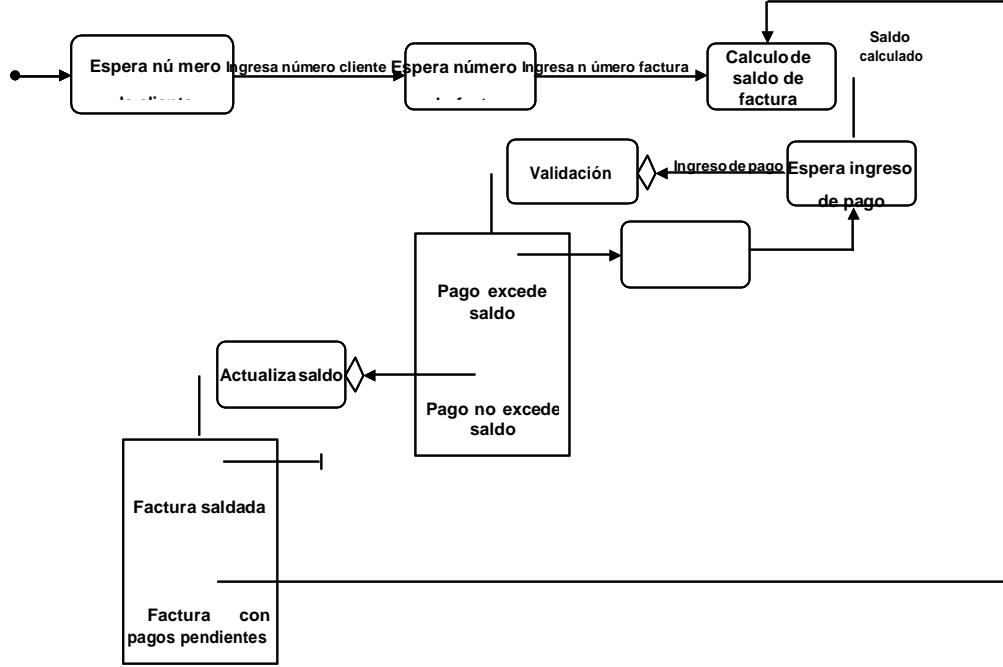


Figura 4.11 Diagrama de transición de estados de pagos



## **Generación de archivo de contabilidad**

Como se mencionó al inicio del documento, este sistema no maneja información relativa a contabilidad, es por eso que se hace necesaria la exportación de datos de compras y ventas, para su ingreso en el sistema de contabilidad, completamente ajeno a este proyecto.

Para la obtención del un archivo de texto con la información requerida por el sistema de contabilidad se debe:

1. Obtener todos los movimientos comerciales realizados en el periodo de tiempo especificado por el usuario, sus importes, fechas y moneda en que se efectuaron. Estos movimientos son: facturas, órdenes de compra, notas de crédito y cargo en ventas, notas de crédito y cargo en compras, pagos de clientes y pagos a proveedores.
2. Relacionar los movimientos con los clientes y proveedores respectivos, así como las cuentas contables correspondientes a cada uno de ellos.
3. Agrupar estos registros por día de realización.
4. Convertir los importes de dólares americanos a su equivalencia en moneda nacional según el tipo de cambio que aplicó en la fecha del movimiento.
5. Calcular los importes totales por día, en moneda nacional y en dólares americanos.
6. Generar el archivo de texto según el formato especificado por el sistema de contabilidad.

La generación del archivo de exportación de datos se efectuará según el periodo definido por el usuario en la configuración del proceso; o bien, al momento en que el usuario haga la petición de ejecución.

Si llegase a ocurrir un error, el usuario será informado de la causa de éste y se detendrá la generación del documento.

## Lectura 6. Diseño e implementación 2



---

# 7

# Diseño e implementación

## Objetivos

Los objetivos de este capítulo son introducirlo al diseño de software orientado a objetos con el uso del UML, así como resaltar las preocupaciones relevantes en la implementación. Al estudiar este capítulo:

- comprenderá las actividades más importantes dentro de un proceso de diseño general orientado a objetos;
- identificará algunos de los diferentes modelos que pueden usarse para documentar un diseño orientado a objetos;
- conocerá la idea de patrones de diseño y cómo éstos son una forma de reutilizar el conocimiento y la experiencia de diseño;
- se introducirá en los conflictos clave que debe considerar al implementar el software, incluida la reutilización de software y el desarrollo de código abierto.

## Contenido

Diseño orientado a objetos con el uso del UML  
Patrones de diseño  
Conflictos de implementación  
Desarrollo de código abierto

El diseño y la implementación del software es la etapa del proceso de ingeniería de software en que se desarrolla un sistema de software ejecutable. Para algunos sistemas simples, el diseño y la implementación del software es ingeniería de software, y todas las demás actividades se fusionan con este proceso. Sin embargo, para sistemas grandes, el diseño y la implementación del software son sólo uno de una serie de procesos (ingeniería de requerimientos, verificación y validación, etcétera) implicados en la ingeniería de software.

Las actividades de diseño e implementación de software se encuentran invariablemente entrelazadas. El diseño de software es una actividad creativa donde se identifican los componentes del software y sus relaciones, con base en los requerimientos de un cliente. La implementación es el proceso de realizar el diseño como un programa. Algunas veces, hay una etapa de diseño separada y este último se modela y documenta. En otras ocasiones, un diseño se halla en la mente del programador o se bosqueja burdamente en un pizarrón o en hojas de papel. El diseño trata sobre cómo resolver un problema, de modo que siempre existe un proceso de diseño. Sin embargo, en ocasiones no es necesario o adecuado describir con detalle el diseño usando el UML u otro lenguaje de descripción de diseño.

Diseño e implementación están estrechamente vinculados y usted, por lo general, debe tomar en cuenta los conflictos de implementación cuando desarrolle un diseño. Por ejemplo, usar el UML para documentar un diseño puede ser lo correcto si está programado en un lenguaje orientado a objetos, como Java o C#. Es menos útil cuando se desarrolla en un lenguaje de escritura dinámica, como Python, y no tiene sentido en absoluto si implementa su sistema al configurar un paquete comercial. Como se estudió en el capítulo 3, los métodos ágiles suelen funcionar a partir de bosquejos informales del diseño y dejan a los programadores muchas de las decisiones de diseño.

Una de las decisiones de implementación más importantes, que se toman en una etapa inicial de un proyecto de software, consiste en determinar si debe comprar o diseñar el software de aplicación. En un gran rango de dominios, ahora es posible comprar sistemas comerciales (COTS) que se adapten y personalicen según los requerimientos de los usuarios. Por ejemplo, si desea implementar un sistema de registros médicos, puede comprar un paquete que ya se use en hospitales. Es posible que sea más barato y rápido aplicar este enfoque en vez de desarrollar un sistema en un lenguaje de programación convencional.

Al desarrollarse de esta forma una aplicación, el proceso de diseño se preocupa sobre cómo usar las características de configuración de dicho sistema, para entregar los requerimientos del mismo. Por lo general, no se desarrollan modelos de diseño del sistema, como los modelos de los objetos del sistema y sus interacciones. En el capítulo 16 se estudia el enfoque basado en COTS.

Se supone que la mayoría de los lectores de este libro cuentan con algo de experiencia en diseño e implementación de programas. Esto se adquiere conforme se aprende a programar y se dominan los elementos de un lenguaje de programación como Java o Python. Probablemente usted se instruyó en las buenas prácticas de programación cuando estudió lenguajes de programación, así como al depurar los programas que desarrolla. Por lo tanto, aquí no se cubren temas de programación. En cambio, este capítulo tiene dos metas:

1. Mostrar cómo el modelado de sistemas y el diseño arquitectónico (que se estudian en los capítulos 5 y 6) se ponen en práctica en el desarrollo de un diseño de software orientado a objetos.



### Métodos de diseño estructurado

Los métodos de diseño estructurado refieren que el diseño del software debe realizarse en una forma metódica. Diseñar un sistema incluye seguir los pasos del método, así como corregir el diseño de un sistema en niveles cada vez más detallados. En la década de 1990 había algunos métodos en competencia para diseño orientado a objetos. Sin embargo, los creadores de los métodos de uso más común se reunieron e inventaron el UML, que unificó las anotaciones usadas con los diferentes métodos.

En lugar de enfocarse en los métodos, la mayoría de las discusiones son ahora sobre procesos donde el diseño se ve como parte del proceso global de desarrollo del software. El proceso racional unificado (RUP, por las siglas de *Rational Unified Process*) es una buena muestra de proceso de desarrollo genérico.

<http://www.SoftwareEngineering-9.com/Web/Structured-methods/>

2. Introducir importantes temas de implementación que no se tratan normalmente en los libros de programación. En ellos se incluyen la reutilización de software, la administración de la configuración y el desarrollo de código abierto.

Como hay gran variedad de plataformas de desarrollo, el capítulo no se desvía hacia algún lenguaje de programación o tecnología de implementación específicos. Por lo tanto, todos los ejemplos se presentan usando el UML en vez de un lenguaje de programación como Java o Python.

## 7.1 Orientado a objetos con el uso del UML

Un sistema orientado a objetos se constituye con objetos que interactúan y mantienen su propio estado local y ofrecen operaciones sobre dicho estado. La representación del estado es privada y no se puede acceder directamente desde afuera del objeto. Los procesos de diseño orientado a objetos implican el diseño de clases de objetos y las relaciones entre dichas clases; tales clases definen tanto los objetos en el sistema como sus interacciones. Cuando el diseño se realiza como un programa en ejecución, los objetos se crean dinámicamente a partir de estas definiciones de clase.

Los sistemas orientados a objetos son más fáciles de cambiar que aquellos sistemas desarrollados usando enfoques funcionales. Los objetos incluyen datos y operaciones para manipular dichos datos. En consecuencia, pueden entenderse y modificarse como entidades independientes. Cambiar la implementación de un objeto o agregar servicios no afectará a otros objetos del sistema. Puesto que los objetos se asocian con cosas, con frecuencia hay un mapeo claro entre entidades del mundo real (como componentes de hardware) y sus objetos controladores en el sistema. Esto mejora la comprensibilidad y, por ende, la mantenibilidad del diseño.

Para desarrollar un diseño de sistema desde el concepto hasta el diseño detallado orientado a objetos, hay muchas cuestiones por hacer:

1. Comprender y definir el contexto y las interacciones externas con el sistema.

3. Identificar los objetos principales en el sistema.
4. Desarrollar modelos de diseño.
5. Especificar interfaces.

Como todas las actividades creativas, el diseño no es un proceso secuencial tajante. El diseño se desarrolla al obtener ideas, proponer soluciones y corregir dichas soluciones conforme la información se encuentra disponible. Cuando surjan problemas, se tendrá inevitablemente que regresar y volver a intentar. En ocasiones se exploran opciones a detalle para observar si funcionan; en otras, se ignoran hasta los detalles finales del proceso. En consecuencia, en el texto no se ilustra deliberadamente este proceso como un diagrama simple, debido a que ello implicaría que el diseño se pudiera considerar como una secuencia clara de actividades. De hecho, todas las actividades anteriores están entrelazadas y, por consiguiente, influyen entre sí.

Estas actividades de proceso se explican al diseñar parte del software para la estación meteorológica a campo abierto que se presentó en el capítulo 1. Las estaciones meteo-rológicas a campo abierto se despliegan a áreas remotas. Cada estación meteorológica registra información meteorológica local y la transmite periódicamente, a través de un vínculo satelital, a un sistema de información meteorológica.

### **Contexto e interacciones del sistema**

La primera etapa en cualquier proceso de diseño de software es desarrollar la comprensión de las relaciones entre el software que se diseñará y su ambiente externo. Esto es esencial para decidir cómo proporcionar la funcionalidad requerida del sistema y cómo estructurar el sistema para que se comunique con su entorno. La comprensión del contexto permite también determinar las fronteras del sistema.

El establecimiento de las fronteras del sistema ayuda a decidir sobre las características que se implementarán en el sistema que se va a diseñar, así como sobre las de otros sistemas asociados. En este caso, es necesario decidir cómo se distribuirá la funcionalidad entre el sistema de control para todas las estaciones meteorológicas y el software embedido en la estación meteorológica en sí.

Los modelos de contexto del sistema y los modelos de interacción presentan vistas complementarias de las relaciones entre un sistema y su entorno:

1. Un modelo de contexto del sistema es un modelo estructural, que muestra los otros sistemas en el entorno del sistema a desarrollar.
2. Un modelo de interacción es un modelo dinámico que indica la forma en que el sistema interactúa con su entorno conforme se utiliza.

El modelo del contexto de un sistema puede representarse mediante asociaciones, las cuales muestran simplemente que existen algunas relaciones entre las entidades que intervienen en la asociación. La naturaleza de las relaciones es ahora específica. Por lo tanto, es posible documentar el entorno del sistema con un simple diagrama de bloques, que mantiene las entidades en el sistema y sus asociaciones. Esto se expone en la figura 7.1, la cual indica que los sistemas en el entorno de cada estación meteorológica son un sistema



### Casos de uso de la estación meteorológica

Reporte del clima: envía datos meteorológicos al sistema de información meteorológica

Reporte de estatus: manda información de estatus al sistema de información meteorológica

Reinicio: si la estación meteorológica se apaga, reinicia el sistema

Apagar: desconecta la estación meteorológica

Reconfigurar: vuelve a configurar el software de la estación meteorológica

Ahorro de energía: pone la estación meteorológica en modo de ahorro de energía

Control remoto: envía comandos de control a cualquier subsistema de la estación meteorológica

<http://www.SoftwareEngineering-9.com/Web/WS/Usecases.html>

de información meteorológica, un sistema de satélite a bordo y un sistema de control. La información cardinal en el vínculo muestra que hay un sistema de control, pero existen muchas estaciones meteorológicas, un satélite y un sistema de información meteorológica general.

Al modelar las interacciones de un sistema con su entorno, se debe usar un enfoque abstracto que no contenga muchos detalles. Una forma de hacerlo es usar un modelo de caso de uso. Como se estudió en los capítulos 4 y 5, cada caso de uso representa una interacción con el sistema. Cada posible interacción se menciona en una elipse, y la entidad externa involucrada en la interacción se representa con una figurilla.

En la figura 7.2 se presenta el modelo de caso de uso para la estación meteorológica. Ahí se demuestra que la estación meteorológica interactúa con el sistema de información meteorológica para reportar datos meteorológicos y el estatus del hardware de la estación meteorológica. Otras interacciones son a través de un sistema de control que puede emitir comandos de control específicos a la estación meteorológica. Como se explicó en el capítulo 5, en el UML se usa una figura para representar otros sistemas, así como a usuarios.

Cada uno de estos casos de uso tiene que describirse en lenguaje natural estructurado. Eso ayudaría a los diseñadores a identificar objetos en el sistema y les daría claridad acerca de lo que se pretende que haga el sistema. Para esta descripción, se usa un formato estándar que identifica con sencillez qué información se intercambia, cómo

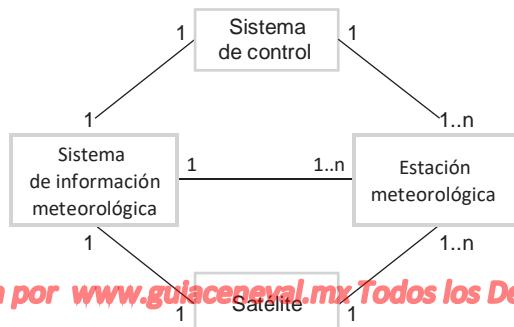


Figura 7.1 Contexto de uso de la estación meteorológica. Se muestra el sistema para la estación meteorológica y sus interacciones con el sistema de control, el sistema de información meteorológica y el satélite.

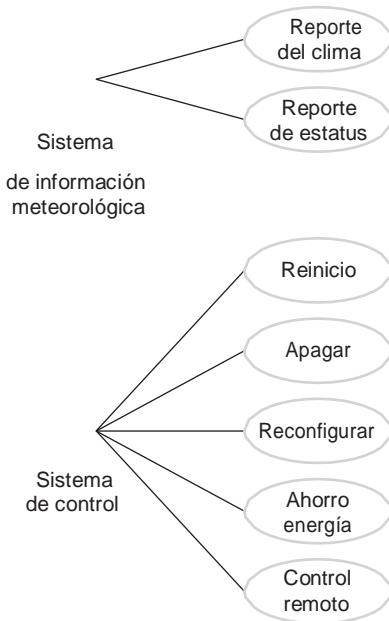


Figura 7.2 Casos de uso para estación meteorológica

se inicia la interacción, etcétera. Eso se muestra en la figura 7.3, que describe el caso de uso sobre el reporte del clima de la figura 7.2. En la Web hay ejemplos de algunos otros casos de uso.

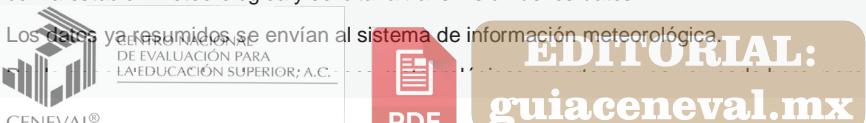
### Diseño arquitectónico

Figura 7.3 Descripción de caso de uso: reporte del clima

Una vez definidas las interacciones entre el sistema de software y el entorno del sistema, se aplica esta información como base para diseñar la arquitectura del sistema. Desde luego, es preciso combinar esto con el conocimiento general de los principios del diseño

Sistema	Estación meteorológica
Caso de uso	Reporte del clima
Actores	Sistema de información meteorológica, estación meteorológica
Datos	La estación meteorológica envía un resumen de datos meteorológicos, recopilados de los instrumentos en el periodo de recolección, al sistema de información meteorológica. Los datos enviados incluyen las temperaturas, máxima, mínima y promedio de la tierra y el aire; asimismo, las presiones de aire máxima, mínima y promedio; la rapidez del viento, máxima, mínima y promedio; la totalidad de la lluvia y la dirección del viento que se muestrea a intervalos de cinco minutos.

El sistema de información meteorológica establece un vínculo de comunicación satelital con la estación meteorológica y solicita la transmisión de los datos.



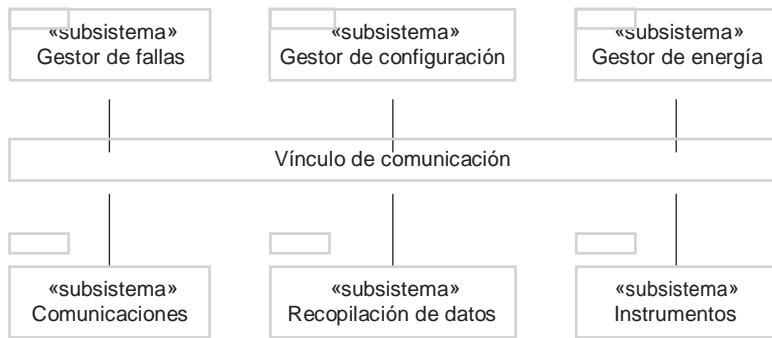


Figura 7.4 Arquitectura de alto nivel de la estación meteorológica

arquitectónico y con la comprensión más detallada del dominio. Identifique los principales componentes que constituyen el sistema y sus interacciones, y posteriormente organice los componentes utilizando un patrón arquitectónico como un modelo en capas o cliente-servidor, aunque esto no sea esencial en esta etapa.

La figura 7.4 muestra el diseño arquitectónico de alto nivel para el software de la estación meteorológica. Esta última se compone de subsistemas independientes que se comunican a través de mensajes de radiodifusión en una infraestructura común, que se presenta como el vínculo “comunicación” en la figura 7.4. Cada subsistema escucha los mensajes en esa infraestructura y capta los mensajes que se dirigen a él. Éste es otro estilo arquitectónico de uso común, además de los descritos en el capítulo 6.

Por ejemplo, cuando el subsistema de comunicaciones recibe un comando de control, como desconectarse, el comando es recibido por cada uno de los otros subsistemas, que entonces se apagan en la forma correcta. El beneficio clave de esta arquitectura consiste en que es fácil soportar diferentes configuraciones de subsistemas debido a que el emisor del mensaje no necesita dirigir el mensaje a un subsistema específico.

La figura 7.5 expone la arquitectura del subsistema de recolección de datos, incluida en la figura 7.4. Los objetos “transmisor” y “receptor” se ocupan de administrar las comunicaciones, y el objeto WeatherData (datos meteorológicos) encapsula la información que se recolecta de los instrumentos y se transmite al sistema de información meteorológica. Este arreglo sigue el patrón productor-consumidor, estudiado en el capítulo 20.

### **Identificación de clase de objeto**

En esta etapa del proceso de diseño, es necesario tener algunas ideas sobre los objetos esenciales en el sistema que se diseña. Conforme aumente su comprensión del diseño, corregirá estas ideas de los objetos del sistema. La descripción del caso de uso ayuda a identificar objetos y operaciones del sistema. A partir de la descripción del caso de uso “reporte del clima”, es evidente que se necesitarán objetos que representen los instrumentos que recopilan los datos meteorológicos, así como un objeto que simbolice el resumen de los datos meteorológicos. También se suele requerir de un(unos) objeto(s) de sistema de alto nivel que encapsule(n) las interacciones del sistema definidas en los

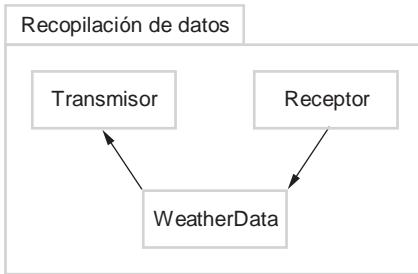


Figura 7.5 Arquitectura de sistema de recopilación de datos

casos de uso. Con dichos objetos en mente, usted puede comenzar a identificar las clases de objetos en el sistema.

Hay varias propuestas sobre cómo identificar las clases de objetos en los sistemas orientados a objetos:

1. Use un análisis gramatical de una descripción en lenguaje natural del sistema a construir. Objetos y atributos son sustantivos; operaciones o servicios son verbos (Abbott, 1983).
2. Utilice entidades tangibles (cosas) en el dominio de aplicación (como aeronave), roles (como administrador o médico), eventos (como peticiones), interacciones (como reuniones), ubicaciones (como oficinas), unidades organizacionales (como compañías), etcétera (Coad y Yourdon, 1990; Shlaer y Mellor, 1988; Wirfs-Brock *et al.*, 1990).
3. Emplee un análisis basado en escenarios, donde a la vez se identifiquen y analicen varios escenarios de uso de sistema. A medida que se analiza cada escenario, el equipo responsable del análisis debe identificar los objetos, los atributos y las operaciones requeridos (Beck y Cunningham, 1989).

En la práctica, debe usar varias fuentes de conocimiento para descubrir clases de objetos. Dichas clases, así como los atributos y las operaciones que se identificaron al comienzo a partir de la descripción informal del sistema, pueden ser un punto de inicio para el diseño. Entonces es posible usar más información del conocimiento del dominio de aplicación o del análisis del escenario para corregir y ampliar los objetos iniciales. Esta información se recopila de los documentos de requerimientos, de discusiones con usuarios o de análisis de los sistemas existentes.

En la estación meteorológica a campo abierto, la identificación de objetos se basa en el hardware tangible del sistema. Aun cuando no se tiene suficiente espacio para incluir todos los objetos de sistema, en la figura 7.6 se muestran cinco clases de objetos. Los objetos termómetro de tierra, anemómetro y barómetro pertenecen al dominio de aplicación; en tanto que los objetos WeatherStation (estación meteorológica) y WeatherData (datos meteorológicos) se identificaron a partir de la descripción del sistema y de la descripción del escenario (caso de uso):

1. La clase de objeto WeatherStation proporciona la interfaz básica de la estación meteorológica con su entorno. Sus operaciones reflejan las interacciones que se



reportWeather ()	getTemperature
reportStatus () powerSave	windSpeeds
(instrumentos)	windDirections
remoteControl (comandos)	pressures
reconfigure (comandos)	rainfall
restart (instrumentos)	
shutdown (instrumentos)	

collect ()	summarize
()	

Termómetro de tierra	Anemómetro	Barómetro
gt_Ident temperature	an_Ident windSpeed windDirection	bar_Ident pressure height
get() test()	get() test()	get() test()

Figura 7.6 Objetos de estación meteorológica

muestran en la figura 7.3. En este caso, aunque se usa una sola clase de objeto para encapsular todas las interacciones, en otros diseños se pueden elaborar varias clases diferentes de la interfaz del sistema.

- 2 La clase de objeto WeatherData es responsable de procesar el comando del report- Weather (reporte del clima). Envía los datos resumidos desde los aparatos de la estación meteorológica hacia el sistema de información meteorológica.
- 3 Las clases de objetos “termómetro de tierra”, “anemómetro” y “barómetro” se relacionan directamente con instrumentos en el sistema. Reflejan las entidades de hard- ware tangibles en el sistema, y las operaciones se relacionan con el control de dicho hardware. Tales objetos operan de manera autónoma para recopilar datos en la frecuencia especificada y almacenar localmente los datos recopilados. Estos datos se entregan por encargo al objeto WeatherData.

El conocimiento del dominio de aplicación se usa para identificar otros objetos, atributos y servicios. Se sabe que las estaciones meteorológicas se localizan generalmente en lugares remotos e incluyen varios instrumentos que algunas veces funcionan mal. Las fallas en los instrumentos deben reportarse automáticamente. Esto implica que se necesitan atributos y operaciones para comprobar el buen funcionamiento de los instrumentos. Existen muchas estaciones meteorológicas remotas, de modo que cada estación meteorológica debe tener su propio identificador (*identifier*).

En esta etapa del proceso de diseño hay que enfocarse en los objetos en sí, sin pensar en cómo podrían implementarse. Una vez identificados los objetos, corrija el diseño del objeto. Busque características comunes y luego elabore la jerarquía de herencia para el sistema. Por ejemplo, puede identificar una superclase “instrumento” que defina las características comunes de todos los instrumentos, como un identificador y las operaciones *get* (conseguir) y *test* (probar). También es posible agregar nuevos atributos y operaciones a la superclase, como un atributo que mantenga la frecuencia de recolección de datos.

## Modelos de diseño

Como se estudió en el capítulo 5, los modelos de diseño o sistema muestran los objetos o clases de objetos en un sistema. También indican las asociaciones y relaciones entre tales entidades. Dichos modelos son el puente entre los requerimientos y la implementación de un sistema. Deben ser abstractos, de manera que el detalle innecesario no oculte las relaciones entre ellos y los requerimientos del sistema. Sin embargo, deben incluir suficiente detalle para que los programadores tomen decisiones de implementación.

Por lo general, este tipo de conflicto se supera cuando se desarrollan modelos con diferentes niveles de detalle. Donde existan vínculos cercanos entre ingenieros de requerimientos, diseñadores y programadores, los modelos abstractos pueden ser entonces todo lo que se requiera. Es posible tomar decisiones de diseño específico a medida que se implementa el sistema, y los problemas se resuelven mediante discusiones informales. Cuando los vínculos entre especificadores, diseñadores y programadores del sistema son indirectos (por ejemplo, donde un sistema se diseña en una parte de una organización, pero se implementa en otra), es probable que se necesiten modelos más detallados.

Por consiguiente, un paso importante en el proceso de diseño es determinar los modelos de diseño que se necesitarán y el nivel de detalle requerido en dichos modelos. Lo anterior depende del tipo de sistema a desarrollar. Usted diseña un sistema de procesamiento secuencial de datos de forma diferente que un sistema embebido de tiempo real, así que necesitará distintos modelos de diseño. El UML soporta 13 diversos tipos de modelos, pero, como se estudió en el capítulo 5, rara vez los usará todos. Minimizar el número de la producción de modelos reduce los costos del diseño y el tiempo requerido para completar el proceso de diseño.

Al usar el UML para la elaboración de un diseño, usted por lo regular desarrollará dos tipos de modelo de diseño:

1. Modelos estructurales, que describen la estructura estática del sistema usando las clases de objetos y sus relaciones. Las relaciones importantes que pueden documentarse en esta etapa son las relaciones de generalización (herencia), las relaciones usa/ usado por y las relaciones de composición.
2. Modelos dinámicos, que explican la estructura dinámica del sistema y muestran las interacciones entre los objetos del sistema. Las interacciones que pueden documentarse incluyen la secuencia de peticiones de servicio realizadas por los objetos, así como los cambios de estado que activan las interacciones de dichos objetos.

En las primeras fases del proceso de diseño, se considera que existen tres modelos que son útiles particularmente para agregar detalle a los modelos de caso de uso y arquitectónico:

1. Modelos de subsistema, que exponen los agrupamientos lógicos de objetos en subsistemas coherentes. Se representan mediante una forma de diagrama de clase en el que cada subsistema se muestra como un paquete con objetos encerrados. Los modelos de subsistema son modelos estáticos (estructurales).



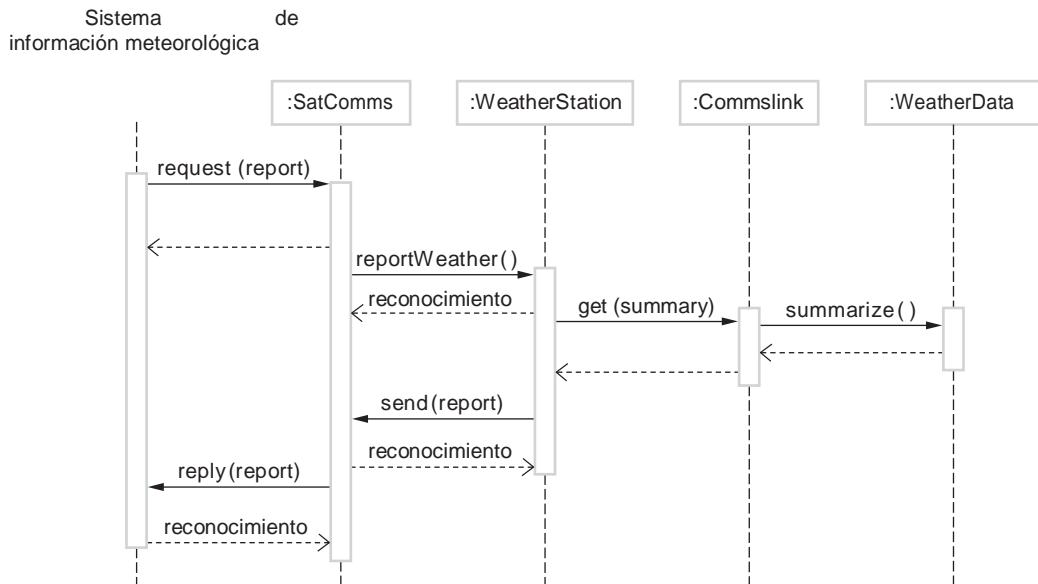


Figura 7.7 Diagrama de secuencia que describe recolección de datos

- 2 Modelos de secuencia, que ilustran la secuencia de interacciones de objetos. Se representan mediante una secuencia UML o un diagrama de colaboración. Los modelos de secuencia son modelos dinámicos.
- 3 Modelos de máquina de estado, que muestran cómo los objetos individuales cambian su estado en respuesta a eventos. Se representan en UML a través de diagramas de estado. Los modelos de máquina de estado son modelos dinámicos.

Un modelo de subsistema es un modelo estático útil, pues señala cómo está organizado un diseño en grupos de objetos relacionados lógicamente. En la figura 7.4 ya se expuso este tipo de modelo, para mostrar los subsistemas en el sistema del mapeo meteorológico. Al igual que los modelos de subsistemas, puede diseñar también modelos de objeto detallados, que presenten todos los objetos en los sistemas y sus asociaciones (herencia, generalización, agregación, etcétera). Sin embargo, hay un peligro al hacer demasiado modelado. No debe tomar decisiones detalladas acerca de la implementación que en realidad debería dejarse a los programadores del sistema.

Los modelos de secuencia son modelos dinámicos que describen, para cada modo de interacción, la secuencia de interacciones de objeto que tienen lugar. Cuando se documenta un diseño, debe producirse un modelo de secuencia por cada interacción significativa. Si usted desarrolló un modelo de caso de uso, entonces debe haber un modelo de secuencia para cada caso de uso que identifique.

La figura 7.7 es un ejemplo de modelo de secuencia, que se muestra como un diagrama de secuencia UML. Este diagrama indica la secuencia de interacciones que tienen lugar cuando un sistema externo solicita los datos resumidos de la estación meteorológica. Los diagramas de secuencia se leen de arriba abajo:

1. El objeto SatComms (comunicaciones de satélite) recibe una petición del sistema de información meteorológica para recopilar un reporte del clima de una estación

meteorológica. Reconoce la recepción de esta petición. La flecha continua en el mensaje enviado señala que el sistema externo no espera una respuesta, sino que puede realizar otro procesamiento.

2. SatComms envía un mensaje a WeatherStation, vía un vínculo satelital, para crear un resumen de los datos meteorológicos recolectados. De nuevo, la flecha continua indica que SatComms no se suspende en espera de una respuesta.
3. WeatherStation envía un mensaje a un objeto Commslink (vínculos comunicacionales) para resumir los datos meteorológicos. En este caso, la punta de flecha discontinua revela que la instancia del objeto WeatherStation espera una respuesta.
4. Commslink llama al método summarize (resumir) en el objeto WeatherData y espera una respuesta.
5. El resumen de datos meteorológicos se calcula y regresa a WeatherStation vía el objeto Commslink.
6. WeatherStation entonces llama al objeto SatComms para transmitir los datos resumidos al sistema de información meteorológica, a través del sistema de comunicaciones de satélite.

Los objetos SatComms y WeatherStation se implementan como procesos concurrentes, cuya ejecución puede suspenderse y resumirse. La instancia del objeto SatComms escucha los mensajes del sistema externo, decodifica dichos mensajes e inicia operaciones de estación meteorológica.

Los diagramas de secuencia se usan para modelar el comportamiento combinado de un grupo de objetos, pero quizás también se deseé resumir el comportamiento de un objeto o un subsistema, en respuesta a mensajes y eventos. Para hacerlo, se usa un modelo de máquina de estado que muestre cómo la instancia objeto cambia de estado dependiendo de los mensajes que recibe. El UML incluye diagramas de estado, inventados inicialmente por Harel (1987) para describir modelos de máquina de estado.

La figura 7.8 es un diagrama de estado para el sistema de estación meteorológica que indica cómo responde a las peticiones de varios servicios.

Puede leer este diagrama del siguiente modo:

1. Si el estado del sistema es Shutdown (apagado), entonces puede responder a un mensaje restart(), reconfigure() o powerSave() (reiniciar, reconfigurar o ahorrar energía, respectivamente). La flecha sin etiqueta con la burbuja negra evidencia que el estado Shutdown es el estado inicial. Un mensaje restart() causa una transición a operación normal. Los mensajes powerSave() y reconfigure() producen una transición a un estado donde el sistema se reconfigura a sí mismo. El diagrama de estado muestra que la reconfiguración sólo se permite cuando el sistema ha estado apagado.
2. En el estado Running (operación), el sistema espera más mensajes. Si recibe un mensaje Shutdown(), el objeto regresa al estado apagado.
3. Si capta un mensaje reportWeather() (reporte del clima), el sistema avanza al estado Summarizing (resumir). Cuando el resumen está completo, el sistema avanza hacia un estado Transmitting (transmisión),



donde la información se transfiere al sistema remoto. Luego regresa al estado Running.

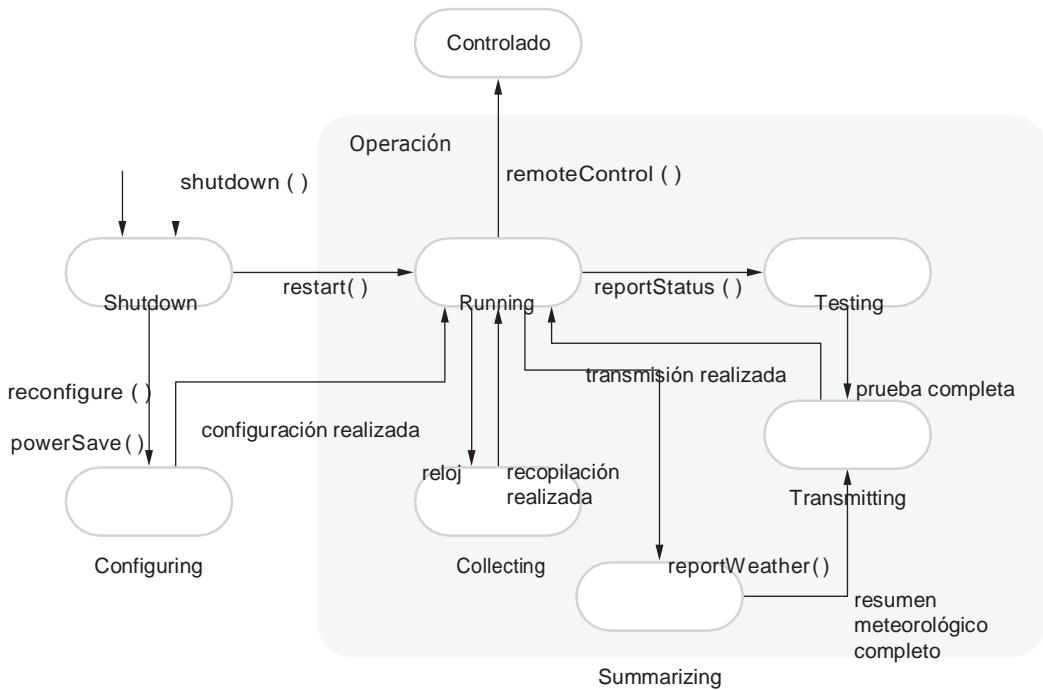


Figura 7.8 Diagrama de estado para la estación meteorológica

4. Si recibe un mensaje `reportStatus()` (reportar estatus), el sistema avanza hacia el estado `Testing` (probar), luego al estado `Transmitting`, antes de regresar al estado `Running`.
5. Si recibe una señal del reloj, el sistema se mueve hacia el estado `Collecting` (recolección), donde recaba los datos de los instrumentos. A cada instrumento se le instruye a su vez para recolectar sus datos de los sensores asociados.
6. Si recibe un mensaje `remoteControl()`, el sistema avanza hacia un estado controlado donde responde a diferentes conjuntos de mensajes desde la sala de control remoto. Éstos no se muestran en el diagrama.

Los diagramas de estado son modelos útiles de alto nivel de un sistema o la operación de un objeto. Por lo general, no se requiere un diagrama de estado para todos los objetos en el sistema. Muchos de los objetos en un sistema son relativamente simples y un modelo de estado añade detalle innecesario al diseño.

### Especificación de interfaz

Una parte importante de cualquier proceso de diseño es la especificación de las interfaces entre los componentes en el diseño. Es necesario especificar las interfaces de modo que los objetos y subsistemas puedan diseñarse en paralelo. Una vez especificada la interfaz, los desarrolladores de otros objetos pueden suponer que se implementará la interfaz.

El diseño de interfaz se preocupa por la especificación del detalle de la interfaz



PDF

**EDITORIAL:**  
guiaceneval.mx



---

7.1 ■ Diseño orientado a objetos con el uso del UML **190**  
hacia un objeto o un grupo de objetos. Esto significa definir las firmas y la semántica de  
los

Figura 7.9 Interfaces de la estación meteorológica

«interfaz» Reporte	«interfaz» Control remoto
weatherReport (WS-Ident): Wreport statusReport (WS-Ident): Sreport	startInstrument (instrument): iStatus stopInstrument (instrument): iStatus collectData (instrument): iStatus provideData (instrument): string

servicios que ofrecerá el objeto o un grupo de objetos. Las interfaces pueden especificarse en el UML con la misma notación de un diagrama de clase. Sin embargo, no hay sección de atributos y debe incluirse el estereotipo UML «interface» en la parte del nombre. La semántica de la interfaz se define mediante el lenguaje de restricción de objeto (OCL). Esto se explica en el capítulo 17, donde se estudia la ingeniería de software basada en componentes. También muestra una forma alternativa para representar interfaces en el UML.

En un diseño de interfaz no se deben incluir detalles de la representación de datos, pues los atributos no se definen en una especificación de interfaz. Sin embargo, debe tener operaciones para acceder a los datos y actualizarlos. Puesto que la representación de datos está oculta, puede cambiar fácilmente sin afectar los objetos que usan dichos datos. Esto conduce a un diseño que inherentemente es más mantenable. Por ejemplo, una representación de arreglo en pila puede cambiarse a una representación en lista, sin afectar otros objetos que usen la pila. En contraste, con frecuencia tiene sentido exponer los atributos en un modelo de diseño estático, pues es la forma más compacta de ilustrar las características esenciales de los objetos.

No hay una relación simple 1:1 entre objetos e interfaces. El mismo objeto puede tener muchas interfaces, cada una de las cuales es un punto de vista de los métodos que ofrece. Esto se soporta directamente en Java, donde las interfaces se declaran por separado de los objetos, y los objetos “implementan” interfaces. De igual modo, puede accederse a un grupo de objetos a través de una sola interfaz.

La figura 7.9 indica dos interfaces que pueden definirse para la estación meteorológica. La interfaz de la izquierda es una interfaz de reporte que precisa los nombres de operación que se usan para generar reportes del clima y de estatus. Éstos se mapean directamente a operaciones en el objeto WeatherStation. La interfaz de control remoto proporciona cuatro operaciones, que se mapean en un solo método en el objeto WeatherStation. En este caso, las operaciones individuales se codifican en la cadena (*string*) de comando asociada con el método remoteControl, que se muestra en la figura 7.6.

## 7.2 Patrones de diseño

Los patrones de diseño se derivaron de ideas planteadas por Christopher Alexander (Alexander *et al.*, 1977), quien sugirió que había ciertos patrones comunes de diseño de construcción que eran relativamente agradables y efectivos. El patrón es una descripción del problema y la esencia de su solución, de modo que la solución puede reutilizarse en diferentes configuraciones. El patrón no es una especificación detallada. Más bien, puede



Nombre del patrón: Observer

**Descripción:** Separa el despliegue del estado de un objeto del objeto en sí y permite el ofrecimiento de despliegues alternativos. Cuando cambia el estado del objeto, todos los despliegues se notifican automáticamente y se actualizan para reflejar el cambio.

**Descripción del problema:** En muchas situaciones hay que proporcionar múltiples despliegues de información del estado, tales como un despliegue gráfico y un despliegue tabular. Tal vez no se conozcan todos éstos cuando se especifica la información. Todas las presentaciones alternativas deben soportar la interacción y, cuando cambia el estado, los despliegues en su totalidad deben actualizarse.

Este patrón puede usarse en todas las situaciones en que se requiera más de un formato de despliegue para información del estado y donde no es necesario que el objeto mantenga la información del estado para conocer sobre los formatos específicos de despliegue utilizados.

**Descripción de la solución:** Esto implica dos objetos abstractos, Subject y Observer, y dos objetos concretos,

ConcreteSubject (sujeto concreto) y ConcreteObject (objeto concreto), que heredan los atributos de los objetos abstractos relacionados. Los objetos abstractos contienen operaciones generales que son aplicables en todas las situaciones. El estado a desplegar se mantiene en ConcreteSubject, que hereda operaciones de Subject y le permite agregar y remover Observers (cada observador corresponde a un despliegue) y emite una notificación cuando cambia el estado.

ConcreteObserver (observador concreto) mantiene una copia del estado de ConcreteSubject e implementa la interfaz Update() de Observer que permite que dichas copias se conserven al paso. ConcreteObserver automáticamente despliega el estado y refleja los cambios siempre que se actualice el estado.

El modelo UML del patrón se ilustra en la figura 7.12.

**Consecuencias:** El sujeto sólo conoce al Observer abstracto y no los detalles de la clase concreta. Por lo tanto, existe un acoplamiento mínimo entre dichos objetos. Debido a esta falta de conocimiento, son imprácticas las optimizaciones que mejoran el rendimiento del despliegue. Los cambios al sujeto podrían generar un conjunto de actualizaciones vinculadas a observadores, de las cuales algunas quizás no sean necesarias.

Figura 7.10 El patrón Observer (observador)

considerarla como un descripción de sabiduría y experiencia acumuladas, una solución bien probada a un problema común.

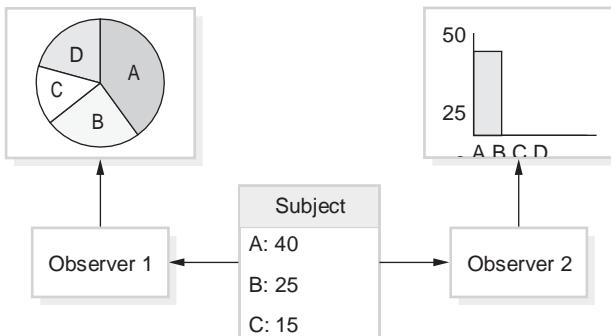
Una cita del sitio Web de Hillside Group (<http://hillside.net>), que se dedica a mantener información acerca de patrones, resume su papel en la reutilización:

*Los patrones y los lenguajes de patrón son formas de describir mejores prácticas, buenos diseños, y captan la experiencia de tal manera que es posible que otros reutilicen esta experiencia.*

Los patrones causaron un enorme impacto en el diseño de software orientado a objetos. Como son soluciones probadas a problemas comunes, se convirtieron en un vocabulario para hablar sobre un diseño. Por lo tanto, usted puede explicar su diseño al describir los patrones que utilizó. Esto es en particular verdadero para los patrones de diseño más conocidos que originalmente describió la “Banda de los cuatro” en su libro de patrones (Gamma *et al.*, 1995). Otras descripciones de patrón en especial importantes son las publicadas en una serie de libros por autores de Siemens, una gran compañía tecnológica europea (Buschmann *et al.*, 1996; Buschmann *et al.*, 2007a; Buschmann *et al.*, 2007b; Kircher y Jain, 2004; Schmidt *et al.*, 2000).

Los patrones de diseño se asocian usualmente con el diseño orientado a objetos.

Figura 7.11 Despliegues múltiples



la experiencia en un patrón es igualmente aplicable a cualquier tipo de diseño de software. De este modo, usted podría tener patrones de configuración para sistemas COTS. Los patrones son una forma de reutilizar el conocimiento y la experiencia de otros diseñadores.

Los cuatro elementos esenciales de los patrones de diseño, definidos por la “Banda de los cuatro” en su libro de patrones, son:

1. Un nombre que sea una referencia significativa al patrón.
2. Una descripción del área problemática que enuncie cuándo puede aplicarse el patrón.
3. Una descripción de solución de las partes de la solución de diseño, sus relaciones y responsabilidades. No es una descripción concreta de diseño; es una plantilla para que una solución de diseño se instale en diferentes formas. Esto con frecuencia se expresa gráficamente y muestra las relaciones entre los objetos y las clases de objetos en la solución.
4. Un estado de las consecuencias, los resultados y las negociaciones, al aplicar el patrón. Lo anterior ayuda a los diseñadores a entender si es factible usar o no un patrón en una situación particular.

Gamma y sus coautores descomponen la descripción del problema en motivación (una descripción del porqué es útil el patrón) y aplicabilidad (una descripción de las situaciones en que puede usarse el patrón). Con la descripción de la solución, explican la estructura del patrón, los participantes, las colaboraciones y la implementación.

Para ilustrar la descripción del patrón, en el texto se usa el patrón Observer, tomado del libro de Gamma *et al.* (1995). Esto se muestra en la figura 7.10. En la descripción del texto, se emplean los cuatro elementos esenciales de descripción y también se incluye un breve enunciado sobre qué puede hacer el patrón. Este patrón se utiliza en situaciones donde se requieren diferentes presentaciones del estado de un objeto. Separa el objeto que debe desplegarse de las diferentes formas de presentación. Lo vemos en la figura 7.11, que muestra dos presentaciones gráficas del mismo conjunto de datos.

Las representaciones gráficas se usan por lo general para explicar las clases de objetos en patrones y sus relaciones. Ello complementa la descripción del patrón y agrega



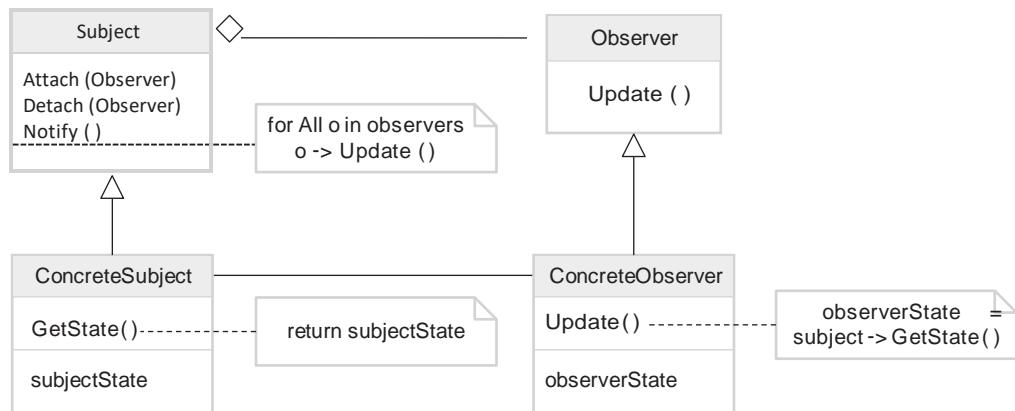


Figura 7.12 Modelo UML del patrón Observer

detalles a la descripción de la solución. La figura 7.12 es la representación en UML del patrón Observer.

Para usar patrones en su diseño, se debe reconocer que cualquier problema de diseño que enfrente es posible que tenga un patrón asociado para aplicarse. Los ejemplos de tales problemas, documentados en el libro de patrones original de la “Banda de los cuatros”, incluyen:

1. Señalar a varios objetos que cambiaron el estado de algún otro objeto (patrón Observer).
2. Ordenar las interfaces en un número de objetos relacionados que a menudo se hayan desarrollado incrementalmente (patrón Façade, fachada).
3. Proporcionar una forma estándar para ingresar a los elementos en una colección, sin importar cómo se implementó dicha colección (patrón Iterator, iterador).
4. Permitir la posibilidad de extender la funcionalidad de una clase existente en tiempo de operación (patrón Decorator, decorador).

Los patrones soportan reutilización de concepto de alto nivel. Cuando intente reutilizar componentes ejecutables, estará restringido inevitablemente por decisiones de diseño detalladas que los implementadores tomaron de dichos componentes. Éstas varían desde los algoritmos particulares usados para implementar los componentes, hasta los objetos y tipos en las interfaces del componente. Cuando dichas decisiones de diseño entran en conflicto con los requerimientos particulares, la reutilización de componentes resulta imposible o introduce inefficiencias en su sistema. El uso de patrones significa que se reutilizan las ideas, pero podría adaptar la implementación para ajustarse al sistema que se desarrolla.

Cuando usted comienza el diseño de un sistema, quizás sea difícil saber, por adelantado, si necesitará un patrón particular. Por lo tanto, el uso de patrones en un proceso de diseño con frecuencia implica el desarrollo de un diseño, experimentar un problema y, luego, reconocer que puede usarse un patrón. En efecto, esto es posible si se enfoca en los 23 patrones de propósito general documentados en el libro de patrones original.

embargo, si su problema es diferente, quizá descubra que es difícil encontrar un patrón adecuado entre los cientos de patrones propuestos.

Los patrones son una gran idea, pero para usarlos de manera efectiva se necesita experiencia en diseño de software. Hay que reconocer las situaciones donde se aplicaría un patrón. Los programadores inexpertos, incluso si leyeron los libros acerca de patrones, siempre descubrirán que es difícil decidir si deben reutilizar un patrón o necesitan desarrollar una solución de propósito especial.

## 7.3 Conflictos de implementación

La ingeniería de software incluye todas las actividades implicadas en el desarrollo de software, desde los requerimientos iniciales del sistema hasta el mantenimiento y la administración del sistema desplegado. Una etapa crítica de este proceso es, desde luego, la implementación del sistema, en la cual se crea una versión ejecutable del software. La implementación quizás requiera el desarrollo de programas en lenguajes de programación de alto o bajo niveles, o bien, la personalización y adaptación de sistemas comerciales genéricos para cubrir los requerimientos específicos de una organización.

Se supone que la mayoría de los lectores de este libro comprenderán los principios de programación y tendrán alguna experiencia al respecto. Como este capítulo tiene la intención de ofrecer un enfoque independiente de lenguaje, no se centró en conflictos de la buena práctica de programación, pues para esto se tendrían que usar ejemplos específicos de lenguaje. En su lugar, se introducen algunos aspectos de implementación que son muy importantes para la ingeniería de software que, por lo general, no se tocan en los textos de programación. Éstos son:

1. *Reutilización* La mayoría del software moderno se construye por la reutilización de los componentes o sistemas existentes. Cuando se desarrolla software, debe usarse el código existente tanto como sea posible.
2. *Administración de la configuración* Durante el proceso de desarrollo se crean muchas versiones diferentes de cada componente de software. Si usted no sigue la huella de dichas versiones en un sistema de gestión de configuración, estará proclive a incluir en su sistema las versiones equivocadas de dichos componentes.
3. *Desarrollo de huésped-objetivo* La producción de software no se ejecuta por lo general en la misma computadora que el entorno de desarrollo de software. En vez de ello, se diseña en una computadora (el sistema huésped) y se ejecuta en una computadora separada (el sistema objetivo). Los sistemas huésped y objetivo son algunas veces del mismo tipo, aunque suelen ser completamente diferentes.

### Reutilización

De la década de 1970 a la de 1990, gran parte del nuevo software se desarrolló desde cero. Escribir todo el código en un lenguaje de programación de alto nivel. La única reutilización o software significativo era la reutilización de funciones y objetos en las



librerías de lenguaje de programación. Sin embargo, los costos y la presión por fechas significaban que este enfoque se volvería cada vez más inviable, sobre todo para sistemas comerciales y basados en Internet. En consecuencia, surgió un enfoque al desarrollo basado en la reutilización del software existente y ahora se emplea generalmente para sistemas empresariales, software científico y, cada vez más, en ingeniería de sistemas embebidos.

La reutilización de software es posible en algunos niveles diferentes:

1. *El nivel de abstracción* En este nivel no se reutiliza el software directamente, sino más bien se utiliza el conocimiento de abstracciones exitosas en el diseño de su software. Los patrones de diseño y los arquitectónicos (tratados en el capítulo 6) son vías de representación del conocimiento abstracto para la reutilización.
2. *El nivel objeto* En este nivel se reutilizan directamente los objetos de una librería en vez de escribir uno mismo en código. Para implementar este tipo de reutilización, se deben encontrar librerías adecuadas y descubrir si los objetos y métodos ofrecen la funcionalidad que se necesita. Por ejemplo, si usted requiere procesar mensajes de correo en un programa Java, tiene que usar objetos y métodos de una librería JavaMail.
3. *El nivel componente* Los componentes son colecciones de objetos y clases de objetos que operan en conjunto para brindar funciones y servicios relacionados. Con frecuencia se debe adaptar y extender el componente al agregar por cuenta propia cierto código. Un ejemplo de reutilización a nivel componente es donde usted construye su interfaz de usuario mediante un marco. Éste es un conjunto de clases de objetos generales que aplica manipulación de eventos, gestión de despliegue, etcétera. Agrega conexiones a los datos a desplegar y escribe el código para definir detalles de despliegue específicos, como plantilla de la pantalla y colores.
4. *El nivel sistema* En este nivel se reutilizan sistemas de aplicación completos. Usualmente esto implica cierto tipo de configuración de dichos sistemas. Puede hacerse al agregar y modificar el código (si reutiliza una línea de producto de software) o al usar la interfaz de configuración característica del sistema. La mayoría de los sistemas comerciales se diseñan ahora de esta forma, donde se adapta y reutilizan sistemas COTS (comerciales) genéricos. A veces este enfoque puede incluir la reutilización de muchos sistemas diferentes e integrarlos para crear un nuevo sistema.

Al reutilizar el software existente, es factible desarrollar nuevos sistemas más rápidamente, con menos riesgos de desarrollo y también costos menores. Puesto que el software reutilizado se probó en otras aplicaciones, debe ser más confiable que el software nuevo. Sin embargo, existen costos asociados con la reutilización:

1. Los costos del tiempo empleado en la búsqueda del software para reutilizar y valorar si cubre sus necesidades o no. Es posible que deba poner a prueba el software para asegurarse de que funcionará en su entorno, especialmente si éste es diferente. Toda la documentación de desarrollo.
2. Donde sea aplicable, los costos por comprar el software reutilizable. Para

sistemas comerciales grandes, dichos costos suelen ser muy elevados.



3. Los costos por adaptar y configurar los componentes de software o sistemas reutilizables, con la finalidad de reflejar los requerimientos del sistema que se desarrolla.
4. Los costos de integrar elementos de software reutilizable unos con otros (si usa software de diferentes fuentes) y con el nuevo código que haya desarrollado. Integrar software reutilizable de diferentes proveedores suele ser difícil y costoso, ya que los proveedores podrían hacer conjeturas conflictivas sobre cómo se reutilizará su software respectivo.

Cómo reutilizar el conocimiento y el software existentes sería el primer punto a considerar cuando se inicie un proyecto de desarrollo de software. Hay que contemplar las posibilidades de reutilización antes de diseñar el software a detalle, pues tal vez usted quiera adaptar su diseño para reutilización de los activos de software existentes. Como se estudió en el capítulo 2, en un proceso de desarrollo orientado a la reutilización uno busca elementos reutilizables y, luego, modifica los requerimientos y el diseño para hacer un mejor uso de ellos.

Para un gran número de sistemas de aplicación, la ingeniería de software significa en realidad reutilización de software. En consecuencia, en este libro se dedican a este tema varios capítulos de la sección de tecnologías de software (capítulos 16, 17 y 19).

## **Administración de la configuración**

En el desarrollo de software, los cambios ocurren todo el tiempo, de modo que la administración del cambio es absolutamente esencial. Cuando un equipo de individuos desarrolla software, hay que cerciorarse de que los miembros del equipo no interfieran con el trabajo de los demás. Esto es, si dos personas trabajan sobre un componente, los cambios deben coordinarse. De otro modo, un programador podría realizar cambios y sobrescribir en el trabajo de otro. También se debe garantizar que todos tengan acceso a las versiones más actualizadas de componentes de software; de lo contrario, los desarrolladores pueden rehacer lo ya hecho. Cuando algo salga mal con una nueva versión de un sistema, se debe poder retroceder a una versión operativa del sistema o componente.

Administración de la configuración es el nombre dado al proceso general de gestionar un sistema de software cambiante. La meta de la administración de la configuración es apoyar el proceso de integración del sistema, de modo que todos los desarrolladores tengan acceso en una forma controlada al código del proyecto y a los documentos, describir qué cambios se realizaron, así como compilar y vincular componentes para crear un sistema. Por lo tanto, hay tres actividades fundamentales en la administración de la configuración:

1. Gestión de versiones, donde se da soporte para hacer un seguimiento de las diferentes versiones de los componentes de software. Los sistemas de gestión de versiones incluyen facilidades para que el desarrollo esté coordinado por varios programadores. Esto evita que un desarrollador sobrescriba un código que haya sido enviado al sistema por alguien más.
2. Integración de sistema, donde se da soporte para ayudar a los desarrolladores a definir qué versiones de componentes se usan para crear cada versión de un sistema. Luego, esta descripción se utiliza para elaborar automáticamente un sistema al compilar y vincular los componentes requeridos.

3. Rastreo de problemas, donde se da soporte para que los usuarios reporten bugs y otros problemas, y también para que todos los desarrolladores sepan quién trabaja en dichos problemas y cuándo se corrigen.

Las herramientas de administración de la configuración de software soportan cada una de las actividades anteriores. Dichas herramientas pueden diseñarse para trabajar en conjunto en un sistema de gestión de cambio global, como ClearCase (Bellagio y Milligan, 2005). En los sistemas de administración de configuración integrada, se diseñan en conjunto las herramientas de gestión de versiones, integración de sistema y rastreo del problema. Comparten un estilo de interfaz de usuario y se integran a través de un repositorio de código común.

Alternativamente, pueden usarse herramientas por separado, instaladas en un entorno de desarrollo integrado. La gestión de versiones puede soportarse mediante un sistema de gestión de versiones como Subversion (Pilato *et al.*, 2008), que puede soportar desarrollo multisitio o multiequipo. El soporte a la integración de sistema podría construirse en el lenguaje o apoyarse en un conjunto de herramientas por separado, como el sistema de construcción GNU. Esto incluye lo que quizás sea la herramienta de integración mejor conocida, hecha en Unix. El rastreo de bugs o los sistemas de rastreo de conflictos, como Bugzilla, se usan para reportar bugs y otros conflictos, así como para seguir la pista sobre si se corrigieron o no.

En virtud de su importancia en la ingeniería de software profesional, en el capítulo 25 se analizan con más detenimiento la administración de la configuración y del cambio.

### **Desarrollo huésped-objetivo**

La mayoría del desarrollo de software se basa en un modelo huésped-objetivo. El software se desarrolla en una computadora (el huésped), aunque opera en una máquina separada (el objetivo). En un sentido más amplio, puede hablarse de una plataforma de desarrollo y una plataforma de ejecución. Una plataforma es más que sólo hardware. Incluye el sistema operativo instalado más otro software de soporte como un sistema de gestión de base de datos o, para plataformas de desarrollo, un entorno de desarrollo interactivo.

En ocasiones, las plataformas de desarrollo y ejecución son iguales, lo que posibilita diseñar el software y ponerlo a prueba en la misma máquina. Sin embargo, es más común que sean diferentes, de modo que es necesario mover el software desarrollado a la plataforma de ejecución para ponerlo a prueba, u operar un simulador en su máquina de desarrollo.

Los simuladores se usan con frecuencia al elaborar sistemas embebidos. Se simulan dispositivos de hardware, tales como sensores, y los eventos en el entorno donde el sistema se podrá en funcionamiento. Los simuladores aceleran el proceso de desarrollo para sistemas embebidos, pues cada desarrollador puede contar con su propia plataforma de ejecución, sin tener que descargar el software al hardware objetivo. No obstante, los simuladores son costosos de desarrollar y, por lo tanto, a menudo sólo se encuentran disponibles para las arquitecturas de hardware más conocidas.

Si el sistema objetivo tiene instalado middleware u otro software que necesite usar, en tal caso el sistema se debe poner a prueba utilizando dicho software. Tal vez no resulte práctico instalar dicho software en su máquina de desarrollo, incluso si es la misma que la plataforma objetivo, debido a restricciones de licencia. Ante tales circunstancias, usted necesita transferir su código desarrollado a la plataforma de ejecución, con la finalidad de poner a prueba el sistema.





## Diagramas de despliegue UML

Los diagramas de despliegue UML muestran cómo los componentes de software se despliegan físicamente en los procesadores; es decir, el diagrama de despliegue muestra el hardware y el software en el sistema, así como el middleware usado para conectar los diferentes componentes en el sistema. En esencia, los diagramas de despliegue se pueden considerar como una forma de definir y documentar el entorno objetivo.

<http://www.SoftwareEngineering-9.com/Web/Deployment/>

Una plataforma de desarrollo de software debe ofrecer una variedad de herramientas para soportar los procesos de ingeniería de software. Éstas pueden incluir:

1. Un compilador integrado y un sistema de edición dirigida por sintaxis que le permitan crear, editar y compilar código.
2. Un sistema de depuración de lenguaje.
3. Herramientas de edición gráfica, tales como las herramientas para editar modelos UML.
4. Herramientas de prueba, como JUnit (Massol, 2003) que operen automáticamente un conjunto de pruebas sobre una nueva versión de un programa.
5. Herramientas de apoyo de proyecto que le ayuden a organizar el código para diferentes proyectos de desarrollo.

Al igual que dichas herramientas estándar, su sistema de desarrollo puede incluir herramientas más especializadas, como analizadores estáticos (que se estudian en el capítulo 15). Normalmente, los entornos de desarrollo para equipos también contemplan un servidor compartido que opera un sistema de administración del cambio y la configuración y, si acaso, un sistema para soportar la gestión de requerimientos.

Las herramientas de desarrollo de software se agrupan con frecuencia para crear un entorno de desarrollo integrado (IDE), que es un conjunto de herramientas de software que apoyan diferentes aspectos del desarrollo de software, dentro de cierto marco común e interfaz de usuario. Por lo común, los IDE se crean para apoyar el desarrollo en un lenguaje de programación específico, como Java. El lenguaje IDE puede elaborarse específicamente, o ser una exemplificación de un IDE de propósito general, con herramientas de apoyo a lenguaje específico.

Un IDE de propósito general es un marco para colocar herramientas de software, que brinden facilidades de gestión de datos para el software a desarrollar, y mecanismos de integración, que permitan a las herramientas trabajar en conjunto. El entorno Eclipse (Carlson, 2005) es el IDE de propósito general mejor conocido. Este entorno se basa en una arquitectura de conexión (*plug-in*), de modo que pueda especializarse para diferentes lenguajes y dominios de aplicación (Clayberg y Rubel, 2006). Por consiguiente, es posible instalar Eclipse y personalizarlo según sus necesidades específicas al agregar plug-ins (enchufables o conectables), para soportar el desarrollo de sistemas en red en Java o ingeniería de sistemas embebidos usando C.

Como parte del proceso de desarrollo, se requiere tomar decisiones sobre cómo se desplegará el software desarrollado en la plataforma objetivo. Esto es directo para

---

sistemas



embebidos, donde el objetivo es usualmente una sola computadora. Sin embargo, para sistemas distribuidos, es necesario decidir sobre las plataformas específicas donde se desplegarán los componentes. Los conflictos que hay que considerar al tomar esta decisión son:

1. *Los requerimientos de hardware y software de un componente* Si un componente se diseña para una arquitectura de hardware específica, o se apoya en algún otro sistema de software, tiene que desplegarse por supuesto en una plataforma que brinde el soporte requerido de hardware y software.
2. *Los requerimientos de disponibilidad del sistema* Los sistemas de alta disponibilidad pueden necesitar que los componentes se desplieguen en más de una plataforma. Esto significa que, en el caso de una falla de plataforma, esté disponible una implementación alternativa del componente.
3. *Comunicaciones de componentes* Si hay un alto nivel de tráfico de comunicaciones entre componentes, por lo general tiene sentido desplegarlos en la misma plataforma o en plataformas que estén físicamente cercanas entre sí. Esto reduce la latencia de comunicaciones, es decir, la demora entre el tiempo que transcurre desde el momento en que un componente envía un mensaje hasta que otro lo recibe.

Puede documentar sus decisiones sobre el despliegue de hardware y software usando diagramas de despliegue UML, que muestran cómo los componentes de software se distribuyen a través de plataformas de hardware.

Si desarrolla un sistema embebido, quizás deba tomar en cuenta las características del objetivo, como su tamaño físico, capacidades de poder, necesidad de respuestas en tiempo real para eventos de sensor, características físicas de los actuadores, y sistema operativo de tiempo real. En el capítulo 20 se estudia la ingeniería de los sistemas embebidos.

## Desarrollo 7.4 de código abierto

El desarrollo de código abierto es un enfoque al desarrollo de software en que se publica el código de un sistema de software y se invita a voluntarios a participar en el proceso de desarrollo (Raymond, 2001). Sus raíces están en la Free Software Foundation (<http://www.fsf.org>), que aboga porque el código fuente no debe ser propietario sino, más bien, tiene que estar siempre disponible para que los usuarios lo examinen y modifiquen como deseen. Existía la idea de que el código estaría controlado y sería desarrollado por un pequeño grupo central, en vez de por usuarios del código.

El software de código abierto extendió esta idea al usar Internet para reclutar a una población mucho mayor de desarrolladores voluntarios. La mayoría de ellos también son

usuarios del código. En principio al menos, cualquier contribuyente a un proyecto de código abierto puede reportar y corregir bugs, así como proponer nuevas características y funcionalidades. Sin embargo, en la práctica, los sistemas exitosos de código abierto aún se apoyan en un grupo central de desarrolladores que controlan los cambios al software. Desde luego, el producto mejor conocido de código abierto es el sistema operativo Linux, utilizado ampliamente como sistema servidor y, cada vez más, como un entorno de escritorio. Otros productos de código abierto importantes son Java, el



Apache y el sistema de gestión de base de datos mySQL. Los protagonistas principales en la industria de cómputo, como IBM y Sun, soportan el movimiento de código abierto y basan su software en productos de código abierto. Existen miles de otros sistemas y componentes de código abierto menos conocidos que también podrían usarse.

Por lo general, es muy barato o incluso gratuito adquirir software de código abierto. Usualmente, el software de código abierto se descarga sin costo. Sin embargo, si usted quiere documentación y soporte, entonces tal vez deba pagar por ello; aún así, los costos son por lo común bastante bajos. El otro beneficio clave para usar productos de código abierto es que los sistemas de código abierto mayores son casi siempre muy confiables. La razón de esto es una gran población de usuarios que quiere corregir los problemas por sí misma, en lugar de reportarlos al desarrollador y esperar una nueva versión del sistema. Los bugs se descubren y reparan con más rapidez que lo que normalmente sería posible con software propietario.

Para una compañía que desarrolla software, existen dos conflictos de código abierto que debe considerar:

1. ¿El producto que se desarrollará deberá usar componentes de código abierto?
2. ¿Deberá usarse un enfoque de código abierto para el desarrollo del software?

Las respuestas a dichas preguntas dependen del tipo de software que se desarrollará, así como de los antecedentes y la experiencia del equipo de desarrollo.

Si usted diseña un producto de software para su venta, entonces resultan críticos tanto el tiempo en que sale al mercado como la reducción en costos. Si se desarrolla en un dominio donde estén disponibles sistemas en código abierto de alta calidad, puede ahorrar tiempo y dinero al usar dichos sistemas. Sin embargo, si usted desarrolla software para un conjunto específico de requerimientos organizativos, entonces quizás el uso de componentes de código abierto no sea una opción. Tal vez tenga que integrar su software con sistemas existentes que sean compatibles con los sistemas en código abierto disponibles. No obstante, incluso entonces podría ser más rápido y barato modificar el sistema en código abierto, en vez de volver a desarrollar la funcionalidad que necesita.

Cada vez más compañías de productos usan un enfoque de código abierto para el desarrollo. Sus modelos empresariales no dependen de la venta de un producto de software, sino de la comercialización del soporte para dicho producto. Consideran que involucrar a la comunidad de código abierto permitirá que el software se desarrolle de manera más económica, más rápida y creará una comunidad de usuarios para el software. A pesar de ello, de nuevo, esto sólo es aplicable realmente para productos de software en general, y no para aplicaciones específicas de la organización.

Muchas compañías creen que adoptar un enfoque de código abierto revelará conocimiento empresarial confidencial a sus competidores y, por consiguiente, son reticentes a adoptar tal modelo de desarrollo. No obstante, si usted trabaja en una pequeña compañía y abre la fuente de su software, esto puede garantizar a los clientes que podrán soportar el software en caso de que la compañía salga del mercado.

Publicar el código de un sistema no significa que la comunidad en general necesariamente ayudará con su desarrollo. Los productos más exitosos de código abierto han sido productos de plataforma, en vez de sistemas de aplicación. Hay un número limitado de desarrolladores que pueden interesarse en sistemas de aplicación especializados. En sí, elaborar un sistema de software en código abierto no garantiza la inclusión de la comunidad.

## Licencia de código abierto

Aunque un principio fundamental del desarrollo en código abierto es que el código fuente debe estar disponible por entero, esto no significa que cualquiera puede hacer lo que desee con el código. Por ley, el desarrollador del código (una compañía o un individuo) todavía es propietario del código. Puede colocar restricciones sobre cómo se le utiliza al incluir condiciones legales en una licencia de software de código abierto (St. Laurent, 2004). Algunos desarrolladores de código abierto creen que si un componente de código abierto se usa para desarrollar un nuevo sistema, entonces dicho sistema también debe ser de código abierto. Otros están satisfechos de que su código se use sin esta restricción. Los sistemas desarrollados pueden ser propietarios y venderse como sistemas de código cerrado.

La mayoría de las licencias de código abierto se derivan de uno de tres modelos generales:

1. La licencia pública general GNU se conoce como licencia “recíproca”; de manera simple, significa que si usted usa software de código abierto que esté permitido bajo la licencia GPL, entonces debe hacer que dicho software sea de código abierto.
2. La licencia pública menos general GNU es una variante de la licencia anterior, en la que usted puede escribir componentes que se vinculen con el código abierto, sin tener que publicar el código de dichos componentes. Sin embargo, si cambia el componente permitido, entonces debe publicar éste como código abierto.
3. La licencia Berkeley Standard Distribution es una licencia no recíproca, lo cual significa que usted no está obligado a volver a publicar algún cambio o modificación al código abierto. Puede incluir el código en sistemas propietarios que se vendan. Si usa componentes de código abierto, debe reconocer al creador original del código.

Los temas sobre permisos son importantes porque si usa software de código abierto como parte de un producto de software, entonces tal vez esté obligado por los términos de la licencia a hacer que su propio producto sea de código abierto. Si trata de vender su software, quizás desee mantenerlo en secreto. Esto significa que tal vez quiera evitar el uso de software de código abierto con licencia GPL en su desarrollo.

Si construye un software que opere en una plataforma de código abierto, como Linux, en tal caso las licencias no son problema. Sin embargo, tan pronto como comience a incluir componentes de código abierto en su software, necesita establecer procesos y bases de datos para seguir la pista de lo que se usó y sus condiciones de licencia. Bayersdorfer (2007) sugiere que las compañías que administran proyectos que usan código abierto deben:

1. Establecer un sistema para mantener la información sobre los componentes de código abierto que se descargan y usan. Tienen que conservar una copia de la licencia para cada componente que sea válida al momento en que se usó el componente. Las licencias suelen cambiar, así que necesita conocer las condiciones acordadas.
2. Estar al tanto de los diferentes tipos de licencias y entender cómo está autorizado un componente antes de usarlo. Puede decidir el uso de un componente en un sistema, pero no en otro, porque planea usar dichos sistemas en diferentes formas.



3. Estar al tanto de las rutas de evolución para los componentes. Necesita conocer un poco sobre el proyecto de código abierto donde se desarrollaron los componentes, para entender cómo pueden cambiar en el futuro.
4. Educar al personal acerca del código abierto. No es suficiente tener procedimientos para asegurar el cumplimiento de las condiciones de la licencia. También es preciso educar a los desarrolladores sobre el código abierto y el permiso de éste.
5. Tener sistemas de auditoría. Los desarrolladores, con plazos ajustados, pueden sentirse tentados a quebrantar los términos de una licencia. Si es posible, debe tener software para detectar y evitar esto último.
6. Participar en la comunidad de código abierto. Si se apoya en productos de código abierto, debe participar en la comunidad y ayudar a apoyar su desarrollo.

El modelo empresarial de software está cambiando. Se ha vuelto cada vez más difícil edificar una empresa mediante la venta de sistemas de software especializado. Muchas compañías prefieren hacer su software en código abierto y entonces vender soporte y consultoría a los usuarios del software. Es probable que esta tendencia se incremente, con el uso creciente de software de código abierto y con cada vez más software disponible de esta forma.

## PUNTOS CLAVE

- El diseño y la implementación del software son actividades entrelazadas. El nivel de detalle en el diseño depende del tipo de sistema a desarrollar y de si se usa un enfoque dirigido por un plan o uno ágil.
- Los procesos del diseño orientado a objetos incluyen actividades para diseñar la arquitectura del sistema, identificar objetos en el sistema, describir el diseño mediante diferentes modelos de objeto y documentar las interfaces de componente.
- Durante un proceso de diseño orientado a objetos, puede elaborarse una variedad de modelos diferentes. En ellos se incluyen modelos estáticos (modelos de clase, modelos de generalización, modelos de asociación) y modelos dinámicos (modelos de secuencia, modelos de máquina de estado).
- Las interfaces de componente deben definirse con precisión, de modo que otros objetos puedan usarlos. Para definir interfaces es posible usar un estereotipo de interfaz UML.
- Cuando se desarrolla software, siempre debe considerarse la posibilidad de reutilizar el software existente, ya sea como componentes, servicios o sistemas completos.
- La administración de la configuración es el proceso de gestionar los cambios a un sistema de software en evolución. Es esencial cuando un equipo de personas coopera para desarrollar software.

- La mayoría del desarrollo de software es desarrollo huésped-objetivo. Se usa un IDE en una máquina para desarrollar el huésped, que se transfiere a una máquina objetivo para su ejecución.
- El desarrollo de código abierto requiere hacer públicamente disponible el código fuente de un sistema. Esto significa que muchos individuos tienen la posibilidad de proponer cambios y mejoras al software.

## LECTURAS SUGERIDAS

*Design Patterns: Elements of Reusable Object-oriented Software.* Éste es el manual original de patrones de hardware que introdujo los patrones de software a una amplia comunidad. (E. Gamma,

R. Helm, R. Johnson y J. Vlissides, Addison-Wesley, 1995.)

*Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and Iterative Development, 3rd edition.* Larman escribe claramente sobre el diseño orientado a objetos y también analiza el uso del UML. Ésta es una buena introducción al uso de patrones en el proceso de diseño. (C. Larman, Prentice Hall, 2004.)

*Producing Open Source Software: How to Run a Successful Free Software Project.* Este libro es una guía completa de los antecedentes del software de código abierto, los conflictos de permiso y las prácticas de operar un proyecto de desarrollo en código abierto. (K. Fogel, O'Reilly Media Inc., 2008.)

En el capítulo 16 se sugieren más lecturas acerca de la reutilización de software, y en el capítulo 25 otras referentes a la administración de la configuración.

## EJERCICIOS

Con la notación estructurada que se muestra en la figura 7.3, especifique los casos de uso de la estación meteorológica para Report status (reporte de estatus) y Reconfigure (reconfigurar). Tiene que hacer conjeturas razonables acerca de la funcionalidad que se requiere aquí.

Suponga que el MHC-PMS se desarrollará usando un enfoque orientado a objetos. Dibuje un diagrama de caso de uso, que muestre al menos seis posibles casos de uso para este sistema.

Con la notación gráfica UML para clases de objetos, diseñe las siguientes clases de objetos, e identifique los atributos y las operaciones. Use su experiencia para decidir sobre los atributos y las operaciones que deban asociarse con estos objetos:

- Teléfono.
- Impresora para computadora personal.
- Sistema de estéreo personal.
- Cuenta bancaria.
- Catálogo de biblioteca.



Con los objetos de la estación meteorológica, identificados en la figura 7.6 como punto de partida, identifique más objetos que puedan usarse en este sistema. Diseñe una jerarquía de herencia para los objetos que haya identificado.

Desarrolle el diseño de la estación meteorológica para mostrar la interacción entre el subsistema de recolección de datos y los instrumentos que recolectan datos meteorológicos. Utilice diagramas de secuencia para mostrar esta interacción.

Identifique posibles objetos en los siguientes sistemas y desarrolle para ellos un diseño orientado a objetos. Puede hacer conjeturas razonables sobre los sistemas cuando derive el diseño.

- Un sistema de diario grupal y administración del tiempo tiene la intención de apoyar los horarios de las reuniones y citas de un grupo de compañeros de trabajo. Cuando se hace una cita para muchas personas, el sistema encuentra un espacio común en cada uno de sus diarios y arregla la cita para esa hora. Si no hay espacios comunes disponibles, interactúa con el usuario para reordenar su diario personal, con la finalidad de hacer espacio para la cita.
- Una estación de llenado (estación de gasolina) se configurará para operación completamente automatizada. Los conductores pasan su tarjeta de crédito a través de un lector conectado a la bomba de gasolina; la tarjeta se verifica mediante comunicación con una computadora en la compañía de crédito, y se establece un límite de combustible. Luego, el conductor puede tomar el combustible requerido. Cuando se completa la entrega de combustible y la manguera de la bomba regresa a su soporte, el costo del combustible surtido se carga a la cuenta de la tarjeta de crédito del conductor. La tarjeta de crédito se regresa después de la deducción.

Si la tarjeta es inválida, la bomba la devuelve antes de despachar el combustible.

Dibuje un diagrama de secuencia que muestre las interacciones de los objetos en un sistema de diario grupal, cuando un grupo de individuos organizan una reunión.

Dibuje un diagrama de estado UML que señale los posibles cambios de estado en el diario grupal o en el sistema de llenado de la estación.

Con ejemplos, explique por qué es importante la administración de la configuración, cuando un equipo de individuos desarrolla un producto de software.

Una pequeña compañía desarrolló un producto especializado que se configura de manera especial para cada cliente. Los clientes nuevos, por lo general, tienen requerimientos específicos para incorporar en su sistema, y pagan para que esto se desarrolle. La compañía tiene oportunidad de licitar para un nuevo contrato, el cual representaría más del doble de su base de clientes. El nuevo cliente también quiere tener cierta participación en la configuración del sistema. Explique por qué, en estas circunstancias, puede ser buena idea que la compañía que posee el software lo convierta en código abierto.

Abbott, R. (1983). "Program Design by Informal English Descriptions". *Comm. ACM*, 26(11), 882-94.

Alexander, C., Ishikawa, S. y Silverstein, M. (1977). *A Pattern Language: Towns, Building, Construction*. Oxford: Oxford University Press.



- Bayersdorfer, M. (2007). "Managing a Project with Open Source Components". *ACM Interactions*, **14** (6), 33–4.
- Beck, K. y Cunningham, W. (1989). "ALaboratoryforTeachingObject-OrientedThinking". *Proc. OOPSLA'89* (Conference on Object-oriented Programming, Systems, Languages and Applications), ACM Press. 1-6.
- Bellagio, D. E. y Milligan, T. J. (2005). *Software Configuration Management Strategies and IBM Rational Clearcase: A Practical Introduction*. Boston: Pearson Education (IBM Press).
- Buschmann, F., Henney, K. y Schmidt, D. C. (2007a). *Pattern-oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing*. New York: John Wiley & Sons.
- Buschmann, F., Henney, K. y Schmidt, D. C. (2007b). *Pattern-oriented Software Architecture Volume 5: On Patterns and Pattern Languages*. New York: John Wiley & Sons.
- Buschmann, F., Meunier, R., Rohnert, H. y Sommerlad, P. (1996). *Pattern-oriented Software Architecture Volume 1: A System of Patterns*. New York: John Wiley & Sons.
- Carlson, D. (2005). *Eclipse Distilled*. Boston: Addison-Wesley.
- Clayberg, E. y Rubel, D. (2006). *Eclipse: Building Commercial-Quality Plug-Ins*. Boston: Addison Wesley.
- Coad, P. y Yourdon, E. (1990). *Object-oriented Analysis*. Englewood Cliffs, NJ: Prentice Hall.
- Gamma, E., Helm, R., Johnson, R. y Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Mass.: Addison-Wesley.
- Harel, D. (1987). "Statecharts: A Visual Formalism for Complex Systems". *Sci. Comput. Programming*, **8** (3), 231–74.
- Kircher, M. y Jain, P. (2004). *Pattern-Oriented Software Architecture Volume 3: Patterns for Resource Management*. New York: John Wiley & Sons.
- Massol, V. (2003). *JUnit in Action*. Greenwich, CT: Manning Publications.
- Pilato, C., Collins-Sussman, B. y Fitzpatrick, B. (2008). *Version Control with Subversion*. Sebastopol, Calif.: O'Reilly Media Inc.
- Raymond, E. S. (2001). *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol, Calif.: O'Reilly Media, Inc.
- Schmidt, D., Stal, M., Rohnert, H. y Buschmann, F. (2000). *Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects*. New York: John Wiley & Sons.
- Shlaer, S. y Mellor, S. (1988). *Object-Oriented Systems Analysis: Modeling the World in Data*. Englewood Cliffs, NJ: Yourdon Press.
- St. Laurent, A. (2004). *Understanding Open Source and Free Software Licensing*. Sebastopol, Calif.: O'Reilly Media Inc.
- Wirfs-Brock, R., Wilkerson, B. y Weiner, L. (1990). *Designing Object-Oriented Software*. Englewood Cliffs, NJ: Prentice Hall.

## Lectura 7. Gestión de proyectos



---

## 22

# Gestión de proyectos

### Objetivos

El objetivo de este capítulo es introducirlo a la gestión de proyectos de software y a dos importantes actividades relacionadas, esto es, la gestión del riesgo y de personal. Al estudiar este capítulo:

- conocerá las principales tareas de los administradores de proyectos de software;
- se introducirá a la noción de gestión del riesgo y a algunos de los riesgos que pueden surgir en los proyectos de software;
- comprenderá los factores que influyen en la motivación personal y qué significan para los administradores de proyectos de software;
- entenderá los conflictos clave que influyen en el trabajo en equipo, tales como la composición del equipo, la organización y la comunicación.

### Contenido

Gestión del riesgo  
Gestión de personal  
Trabajo en equipo

La gestión de proyectos de software es una parte esencial de la ingeniería de software. Los proyectos necesitan administrarse porque la ingeniería de software profesional está sujeta siempre a restricciones organizacionales de presupuesto y fecha. El trabajo del administrador del proyecto es asegurarse de que el proyecto de software cumpla y supere tales restricciones, además de que entregue software de alta calidad. La buena gestión no puede garantizar el éxito del proyecto. Sin embargo, la mala gestión, por lo general, da como resultado una falla del proyecto: el software puede entregarse tarde, costar más de lo estimado originalmente o no cumplir las expectativas de los clientes.

Desde luego, los criterios de éxito para la gestión del proyecto varían de un proyecto a otro, pero, para la mayoría de los proyectos, las metas importantes son:

1. Entregar el software al cliente en el tiempo acordado.
2. Mantener costos dentro del presupuesto general.
3. Entregar software que cumpla con las expectativas del cliente.
4. Mantener un equipo de desarrollo óptimo y con buen funcionamiento.

Tales metas no son únicas para la ingeniería de software, pero sí lo son para todos los proyectos de ingeniería. Sin embargo, la ingeniería de software es diferente en algunas formas a otros tipos de ingeniería que hacen a la gestión del software particularmente desafiante. Algunas de estas diferencias son:

1. *El producto es intangible* Un administrador de un astillero o un proyecto de ingeniería civil pueden ver el producto conforme se desarrolla. Si hay retraso en el calendario, es visible el efecto sobre el producto: es evidente que algunas partes de la estructura no están terminadas. El software es intangible. No se puede ver ni tocar. Los administradores de proyectos de software no pueden constatar el progreso con sólo observar el artefacto que se construye. Más bien, ellos se apoyan en otros para crear la prueba que pueden utilizar al revisar el progreso del trabajo.
2. *Los grandes proyectos de software con frecuencia son proyectos excepcionales* Los grandes proyectos de software se consideran en general diferentes en ciertas formas de los proyectos anteriores. Por eso, incluso los administradores que cuentan con vasta experiencia pueden encontrar difícil anticiparse a los problemas. Aunado a esto, los vertiginosos cambios tecnológicos en computadoras y comunicaciones pueden volver obsoleta la experiencia de un administrador. Las lecciones aprendidas de proyectos anteriores pueden no ser aplicables a nuevos proyectos.
3. *Los procesos de software son variables y específicos de la organización* El proceso de ingeniería para algunos tipos de sistema, como puentes y edificios, es bastante comprendido. Sin embargo, los procesos de software varían considerablemente de una organización a otra. Aunque se ha producido un notorio avance en la estandarización y el mejoramiento de los procesos, no es posible predecir de manera confiable cuándo un proceso de software particular conducirá a problemas de desarrollo. Esto es especialmente cierto si el proyecto de software es parte de un proyecto de ingeniería de sistemas más amplio.



técnicamente innovadores. Los proyectos de ingeniería (como los nuevos sistemas de transporte) que son reformadores, normalmente también tienen problemas de calendario. Dadas las dificultades, quizás sea asombroso que tantos proyectos de software jueguen a tiempo y dentro del presupuesto!

Es imposible efectuar una descripción laboral estándar para un administrador de proyecto de software. La labor varía enormemente en función de la organización y el producto de software a desarrollar. No obstante, la mayoría de los administradores, en alguna etapa, toman la responsabilidad de varias o todas las siguientes actividades:

1. *Planeación del proyecto* Los administradores de proyecto son responsables de la planeación, estimación y calendarización del desarrollo del proyecto, así como de la asignación de tareas a las personas. Supervisan el trabajo para verificar que se realice de acuerdo con los estándares requeridos y monitorizan el avance para comprobar que el desarrollo esté a tiempo y dentro del presupuesto.
2. *Informes* Los administradores de proyectos por lo común son responsables de informar del avance de un proyecto a los clientes y administradores de la compañía que desarrolla el software. Deben ser capaces de comunicarse en varios niveles, desde codificar información técnica detallada hasta elaborar resúmenes administrativos. Deben redactar documentos concisos y coherentes que sinteticen información crítica de reportes detallados del proyecto. Es necesario que esta información se presente durante las revisiones de avance.
3. *Gestión del riesgo* Los administradores de proyecto tienen que valorar los riesgos que pueden afectar un proyecto, monitorizar dichos riesgos y emprender acciones cuando surjan problemas.
4. *Gestión de personal* Los administradores de proyecto son responsables de administrar un equipo de personas. Deben elegir a los integrantes de sus equipos y establecer formas de trabajar que conduzcan a desempeño efectivo del equipo.
5. *Redactar propuestas* La primera etapa en un proyecto de software puede implicar escribir una propuesta para obtener un contrato de trabajo. La propuesta describe los objetivos del proyecto y cómo se realizará. Por lo general, incluye estimaciones de costo y calendarización, además de justificar por qué el contrato del proyecto debería concederse a una organización o un equipo particular. La escritura de propuestas es una tarea esencial, pues la supervivencia de muchas compañías de software depende de contar con suficientes propuestas aceptadas y concesiones de contratos. Es posible que no haya lineamientos establecidos para esta tarea; la escritura de propuestas es una habilidad que se adquiere a través de práctica y experiencia.

En este capítulo nos centraremos en la gestión del riesgo y la gestión de personal. La planeación de proyectos es un tema importante por derecho propio, que se estudia en el capítulo 23.

---

La gestión del riesgo es una de las tareas más sustanciales para un administrador de proyecto. La gestión del riesgo implica anticipar riesgos que pudieran alterar el calendario del



proyecto o la calidad del software a entregar, y posteriormente tomar acciones para evitar dichos riesgos (Hall, 1998; Ould, 1999). Podemos considerar un riesgo como algo que es preferible que no ocurra. Los riesgos pueden amenazar el proyecto, el software que se desarrolla o a la organización. Por lo tanto, existen tres categorías relacionadas de riesgo:

1. *Riesgos del proyecto* Los riesgos que alteran el calendario o los recursos del proyecto. Un ejemplo de riesgo de proyecto es la renuncia de un diseñador experimentado. Encontrar un diseñador de reemplazo con habilidades y experiencia adecuadas puede demorar mucho tiempo y, en consecuencia, el diseño del software tardará más tiempo en completarse.
2. *Riesgos del producto* Los riesgos que afectan la calidad o el rendimiento del software a desarrollar. Un ejemplo de riesgo de producto es la falla que presenta un componente que se adquirió al no desempeñarse como se esperaba. Esto puede afectar el rendimiento global del sistema, de modo que es más lento de lo previsto.
3. *Riesgos empresariales* Riesgos que afectan a la organización que desarrolla o adquiere el software. Por ejemplo, un competidor que introduce un nuevo producto es un riesgo empresarial. La introducción de un producto competitivo puede significar que las suposiciones hechas sobre las ventas de los productos de software existentes sean excesivamente optimistas.

Desde luego, estos tipos de riesgos se traslanan. Si un programador experimentado abandona un proyecto, esto puede ser un riesgo del proyecto porque, incluso si se sustituye de manera inmediata, el calendario se alterará. Siempre se requiere tiempo para que un nuevo miembro del proyecto comprenda el trabajo realizado, de manera que no puede ser inmediatamente productivo. En consecuencia, la entrega del sistema podría demorarse. La salida de un miembro del equipo también puede ser un riesgo del producto, porque un sustituto tal vez no sea tan experimentado y, por lo tanto, podría cometer errores de programación. Finalmente, puede ser un riesgo empresarial, porque la experiencia de dicho programador es vital para obtener nuevos contratos.

Es necesario registrar los resultados del análisis del riesgo en el plan del proyecto, junto con un análisis de consecuencias, que establece las consecuencias del riesgo para el proyecto, el producto y la empresa. La gestión de riesgos efectiva facilita hacer frente a los problemas y asegurar que éstos no conduzcan a un presupuesto inaceptable o a retrasos en el calendario.

Los riesgos específicos que podrían afectar un proyecto dependen del proyecto y el entorno de la organización donde se desarrolla el software. Sin embargo, también existen riesgos comunes que no se relacionan con el tipo de software a desarrollar y que pueden ocurrir en cualquier proyecto. En la figura 22.1 se muestran algunos de estos riesgos comunes.

La gestión del riesgo es particularmente importante para los proyectos de software, debido a la incertidumbre inherente que enfrentan la mayoría de proyectos. Ésta se deriva de requerimientos vagamente definidos, cambios de requerimientos que obedecen a cambios en las necesidades del cliente, dificultades en estimar el tiempo y los recursos requeridos para el desarrollo de software, o bien, se deriva de diferencias en las habilidades individuales. Es necesario anticipar los riesgos; comprender el efecto de estos riesgos sobre el proyecto, el producto y la empresa; y dar los pasos adecuados para evitar dichos riesgos. Tal vez se necesita diseñar planes de contingencia de manera que, si ocurren los riesgos, se puedan tomar acciones inmediatas de recuperación.

Riesgo	Repercute en	Descripción
Rotación de personal	Proyecto	Personal experimentado abandonará el proyecto antes de que éste se termine.
Cambio administrativo	Proyecto	Habrá un cambio de gestión en la organización con diferentes prioridades.
Indisponibilidad de hardware	Proyecto	Hardware, que es esencial para el proyecto, no se entregará a tiempo.
Cambio de requerimientos	Proyecto y producto	Habrá mayor cantidad de cambios a los requerimientos que los anticipados.
Demoras en la especificación	Proyecto y producto	Especificaciones de interfaces esenciales no están disponibles a tiempo.
Subestimación del tamaño	Proyecto y producto	Se subestimó el tamaño del sistema.
Bajo rendimiento de las herramientas CASE	Producto	Las herramientas CASE, que apoyan el proyecto, no se desempeñan como se anticipaba.
Cambio tecnológico	Empresa	La tecnología subyacente sobre la cual se construye el sistema se sustituye con nueva tecnología.
Competencia de productos	Empresa	Un producto competitivo se comercializa antes de que el sistema esté completo.

Figura 22.1 Ejemplos de riesgos comunes para el proyecto, el producto y la empresa

En la figura 22.2 se ilustra una idea general del proceso de gestión del riesgo. Comprende varias etapas:

1. *Identificación del riesgo* Hay que identificar posibles riesgos para el proyecto, el producto y la empresa.
2. *Análisis de riesgos* Se debe valorar la probabilidad y las consecuencias de dichos riesgos.
3. *Planeación del riesgo* Es indispensable elaborar planes para enfrentar el riesgo, evitarlo o minimizar sus efectos en el proyecto.
4. *Monitorización del riesgo* Hay que valorar regularmente el riesgo y los planes para atenuarlo, y revisarlos cuando se aprenda más sobre el riesgo.

Es preciso documentar los resultados del proceso de gestión del riesgo en un plan de gestión del riesgo. Éste debe incluir un estudio de los riesgos que enfrenta el proyecto, un análisis de dichos riesgos e información de cómo se gestionará el riesgo cuando es probable que se convierta en un problema.

El proceso de gestión del riesgo es un proceso iterativo que continúa a lo largo del proyecto. Una vez desarrollado un plan de gestión del riesgo inicial, se monitoriza la situación para detectar riesgos emergentes. Conforme está disponible más información referente a los riesgos, habrá que volver a analizar los riesgos y decidir si cambió la prioridad del riesgo. Entonces tal vez sea necesario cambiar los planes para evitar el riesgo y gestionar la contingencia.



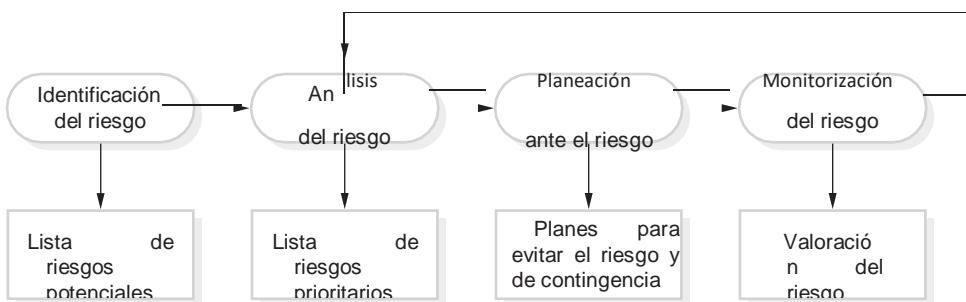


Figura 22.2  
El proceso de gestión del riesgo

### Identificación del riesgo

La identificación del riesgo es la primera etapa del proceso de gestión del riesgo. Se ocupa de identificar los riesgos que pudieran plantear una mayor amenaza al proceso de ingeniería de software, al software a desarrollar, o a la organización que lo desarrolla. La identificación del riesgo puede ser un proceso de equipo en el que este último se reúne para pensar en posibles riesgos. O bien, el administrador del proyecto, con base en su experiencia, identifica los riesgos más probables o críticos.

Como punto de partida para la identificación del riesgo, se recomienda utilizar una lista de verificación de diferentes tipos de riesgo. Existen al menos seis tipos de riesgos que pueden incluirse en una lista de verificación:

1. **Riesgos tecnológicos** Se derivan de las tecnologías de software o hardware usadas para desarrollar el sistema.
2. **Riesgos personales** Se asocian con las personas en el equipo de desarrollo.
3. **Riesgos organizacionales** Se derivan del entorno organizacional donde se desarrolla el software.
4. **Riesgos de herramientas** Resultan de las herramientas de software y otro software de soporte que se usa para desarrollar el sistema.
5. **Riesgos de requerimientos** Proceden de cambios a los requerimientos del cliente y del proceso de gestionarlos.
6. **Riesgos de estimación** Surgen de las estimaciones administrativas de los recursos requeridos para construir el sistema.

La figura 22.3 brinda algunos ejemplos de posibles riesgos en cada una de estas categorías. Al concluir el proceso de identificación de riesgos, se tendrá una larga lista de eventualidades que podrían ocurrir y afectar al producto, al proceso y a la empresa. Entonces se necesita reducir esta lista a un tamaño razonable. Si existen demasiados riesgos, será prácticamente imposible seguir la huella de todos ellos.

### Ánalisis de riesgo

Durante el proceso de análisis de riesgos, hay que considerar cada riesgo identificado y realizar un juicio acerca de la probabilidad y gravedad de dicho riesgo. No hay una forma sencilla de hacer esto. Usted debe apoyarse en su propio juicio y en la experiencia

Tipo de riesgo	Riesgos posibles
Tecnológico	La base de datos que se usa en el sistema no puede procesar tantas transacciones por segundo como se esperaba. (1) Los componentes de software de reutilización contienen defectos que hacen que no puedan reutilizarse como se planeó. (2)
Personal	Es imposible reclutar personal con las habilidades requeridas.(3) El personal clave está enfermo e indisponible en momentos críticos.(4) No está disponible la capacitación requerida para el personal. (5)
De organización	La organización se reestructura de modo que diferentes administraciones son responsables del proyecto. (6) Problemas financieros de la organización fuerzan reducciones en el presupuesto del proyecto. (7)
Herramientas	El código elaborado por las herramientas de generación de código de software es inefficiente. (8) Las herramientas de software no pueden trabajar en una forma integrada. (9)
Requerimientos	Se proponen cambios a los requerimientos que demandan mayor trabajo de rediseño. (10) Los clientes no entienden las repercusiones de los cambios a los requerimientos. (11)
Estimación	Se subestima el tiempo requerido para desarrollar el software. (12) Se subestima la tasa de reparación de defectos. (13) Se subestima el tamaño del software. (14)

Figura 22.3  
Ejemplos  
de diferentes  
tipos de riesgos

obtenida en los proyectos anteriores y los problemas que surgieron en ellos. No es posible hacer valoraciones precisas y numéricas de la probabilidad y gravedad de cada riesgo. En vez de ello, habrá que asignar el riesgo a una de ciertas bandas:

1. La probabilidad del riesgo puede valorarse como muy baja (< 10%), baja (del 10 al 25%), moderada (del 25 al 50%), alta (del 50 al 75%) o muy alta (> 75%).
2. Los efectos del riesgo pueden estimarse como catastróficos (amenazan la supervivencia del proyecto), graves (causarían grandes demoras), tolerables (demoras dentro de la contingencia permitida) o insignificantes.

Luego hay que tabular los resultados de este proceso de análisis mediante una tabla clasificada de acuerdo con la gravedad del riesgo. La figura 22.4 ilustra esto para los riesgos que se identificaron en la figura 22.3. Desde luego, aquí la valoración de la probabilidad y seriedad son arbitrarias. Para hacer esta valoración, se necesita información detallada del proyecto, el proceso, el equipo de desarrollo y la organización.

Desde luego, tanto la probabilidad como la valoración de los efectos de un riesgo pueden cambiar conforme se disponga de más información acerca del riesgo y a medida que se implementen planes de gestión del riesgo. Por lo tanto, esta tabla se debe actualizar durante cada iteración del proceso de riesgo.

Una vez analizados y clasificados los riesgos, valore cuáles son los más significativos. Su juicio debe depender de una combinación de la probabilidad de que el riesgo surja junto con los efectos de dicho riesgo. En general, los riesgos catastróficos deben considerarse tan pronto como los riesgos graves con más de una probabilidad moderada de ocurrencia.



Riesgo	Probabilidad	Efectos
Problemas financieros de la organización fuerzan reducciones en el presupuesto del proyecto. (7)	Baja	Catastrófico
Es imposible reclutar personal con las habilidades requeridas. (3)	Alta	Catastrófico
El personal clave está enfermo e indisponible en momentos críticos. (4)	Moderada	Grave
Los componentes de software de reutilización contienen defectos que hacen que no puedan reutilizarse como se planeó. (2)	Moderada	Grave
Se proponen cambios a los requerimientos que demandan mayor trabajo de rediseño. (10)	Moderada	Grave
La organización se reestructura de modo que diferentes administraciones son responsables del proyecto. (6)	Alta	Grave
La base de datos que se usa en el sistema no puede procesar tantas transacciones por segundo como se esperaba. (1)	Moderada	Grave
Se subestima el tiempo requerido para desarrollar el software. (12)	Alta	Grave
Las herramientas de software no pueden trabajar en una forma integrada. (9)	Alta	Tolerable
Los clientes no entienden las repercusiones de los cambios a los requerimientos. (11)	Moderada	Tolerable
No está disponible la capacitación requerida para el personal. (5)	Moderada	Tolerable
Se subestima la tasa de reparación de defecto. (13)	Moderada	Tolerable
Se subestima el tamaño del software. (14)	Alta	Tolerable
El código elaborado por las herramientas de generación de código de software es ineficiente. (8)	Moderada	Insignificante

Figura 22.4 Tipos de riesgos y ejemplos

Boehm (1988) recomienda identificar y monitorizar los 10 riesgos principales, pero considera que esta cifra es más bien arbitraria. El número correcto de riesgos a monitorizar debe depender del proyecto. Pueden ser cinco o 15. Sin embargo, el número de riesgos elegidos para monitorizar debe ser manejable. Un número de riesgos muy grande requiere recopilar demasiada información. A partir de los riesgos identificados en la figura 22.4, es adecuado considerar los ocho riesgos que tienen consecuencias catastróficas o graves (figura 22.5).

### Planeación del riesgo

El proceso de planeación del riesgo considera cada uno de los riesgos clave identificados y desarrolla estrategias para manejarlos. Para cada uno de los riesgos, usted debe considerar las acciones que puede tomar para minimizar la perturbación del proyecto si se produce el problema identificado en el riesgo. También debe pensar en la información que tal vez necesite recopilar mientras observa el proyecto para que pueda anticipar los problemas.

Riesgo	Estrategia
Problemas financieros de la organización	Prepare un documento informativo para altos ejecutivos en el que muestre cómo el proyecto realiza una aportación muy importante a las metas de la empresa y presente razones por las que los recortes al presupuesto del proyecto no serían efectivos en costo.
Problemas de reclutamiento	Alerte al cliente de dificultades potenciales y de la posibilidad de demoras; investigue la compra de componentes.
Enfermedad del personal	Reorganice los equipos de manera que haya más traslape de trabajo y, así, las personas comprendan las labores de los demás.
Componentes defectuosos	Sustituya los componentes potencialmente defectuosos con la compra de componentes de conocida fiabilidad.
Cambios de requerimientos	Obtenga información de seguimiento para valorar el efecto de cambiar los requerimientos; maximice la información que se oculta en el diseño.
Reestructuración de la organización	Prepare un documento informativo para altos ejecutivos en el que muestre cómo el proyecto realiza una aportación muy importante a las metas de la empresa.
Rendimiento de la base de datos	Investigue la posibilidad de comprar una base de datos de mayor rendimiento.
Subestimación del tiempo de desarrollo	Investigue los componentes comprados; indague el uso de un generador de programa.

Figura 22.5  
Estrategias para ayudar a gestionar el riesgo

Nuevamente, no hay un proceso simple que pueda seguirse para la planeación de contingencias. Se apoya en el juicio y la experiencia del administrador del proyecto.

La figura 22.5 muestra posibles estrategias de gestión del riesgo que se identificaron como los principales riesgos (es decir, aquellos que son graves o intolerables) que se incluyen en la figura 22.4. Dichas estrategias se establecen en tres categorías:

1. *Estrategias de evitación* Seguir estas estrategias significa que se reducirá la probabilidad de que surja el riesgo. Un ejemplo de estrategia de evitación del riesgo es la estrategia de enfrentar los componentes defectuosos que se muestran en la figura 22.5.
2. *Estrategias de minimización* Seguir estas estrategias significa que se reducirá el efecto del riesgo. Un ejemplo de estrategia de minimización de un riesgo es la estrategia para las enfermedades del personal que se indica en la figura 22.5.
3. *Planes de contingencia* Seguir estas estrategias significa que se está preparado para lo peor y se tiene una estrategia para hacer frente a ello. Un ejemplo de estrategia de contingencia es la estrategia para los problemas financieros de la organización que se indica en la figura 22.5.

Aquí se observa una clara analogía con las estrategias utilizadas en los sistemas críticos para garantizar fiabilidad, seguridad y protección, cuando hay que evitar, tolerar o recuperarse de las fallas. Desde luego, es mejor usar una estrategia que evitar el riesgo. Si esto no es posible, se debe usar una estrategia que reduzca las posibilidades de que el riesgo cause graves efectos. Finalmente, se debe contar con estrategias para enfrentar el riesgo cuando éste surja. Tales estrategias deben reducir el efecto global de un riesgo en el proyecto o el producto.



Tipo de riesgo	Indicadores potenciales
Tecnológico	Entrega tardía de hardware o software de soporte; muchos problemas tecnológicos reportados.
Personal	Baja moral de personal; malas relaciones entre miembros del equipo; alta rotación de personal.
De organización	Chismes en la organización; falta de acción de los altos ejecutivos.
Herramientas	Renuencia de los miembros del equipo para usar herramientas; quejas acerca de las herramientas CASE; demandas por estaciones de trabajo mejor equipadas.
Requerimientos	Muchas peticiones de cambio de requerimientos; quejas de los clientes.
Estimación	Falla para cumplir con el calendario acordado; falla para corregir los defectos reportados.

Figura 22.6

Indicadores de riesgo

### **Monitorización del riesgo**

La monitorización del riesgo es el proceso para comprobar que no han cambiado sus suposiciones sobre riesgos del producto, el proceso y la empresa. Hay que valorar regularmente cada uno de los riesgos identificados para decidir si este riesgo se vuelve más o menos probable. También se tiene que considerar si los efectos del riesgo han cambiado o no. Para hacer esto, observe otros factores, como el número de peticiones de cambio de requerimientos, lo que da pistas acerca de la probabilidad del riesgo y sus efectos. Dichos factores dependen claramente de los tipos de riesgos. La figura 22.6 proporciona algunos ejemplos de factores que pueden ser útiles para valorar estos tipos de riesgos.

Los riesgos deben monitorizarse comúnmente en todas las etapas del proyecto. En cada revisión administrativa, es necesario reflexionar y estudiar cada uno de los riesgos clave por separado. También hay que decidir si es más o menos probable que surja el riesgo, y si cambiaron la gravedad y las consecuencias del riesgo.

## **22.2 Gestión de personal**

Las personas que trabajan en una organización de software son los activos más importantes. Cuesta mucho dinero reclutar y retener al buen personal, así que depende de los administradores de software garantizar que la organización obtenga el mejor aprovechamiento posible por su inversión. En las compañías y economías exitosas, esto se logra cuando la organización respeta a las personas y les asigna responsabilidades que reflejan sus habilidades y experiencia.

Es importante que los administradores de proyecto de software comprendan los conflictos técnicos que influyen en el trabajo del desarrollo de software. Sin embargo, por desgracia, los buenos ingenieros de software no necesariamente son buenos administradores de personal. Los ingenieros de software con frecuencia tienen grandes

habilidades técnicas, pero pueden carecer de habilidades más sutiles que les permitan motivar y dirigir a un equipo de desarrollo de proyecto. Como administrador de proyecto, usted deberá estar al tanto de los problemas potenciales de administrar personal y debe tratar de desarrollar habilidades de gestión de recursos humanos.

Desde la perspectiva del autor, existen cuatro factores críticos en la gestión de personal:

1. *Consistencia* Todas las personas en un equipo de proyecto deben recibir un trato similar. Nadie espera que todas las distinciones sean idénticas, pero las personas podrían sentir que sus aportaciones a la organización se menoscapan.
2. *Respeto* Las personas tienen distintas habilidades y los administradores deben respetar esas diferencias. Todos los miembros del equipo deben recibir una oportunidad para aportar. Desde luego, en algunos casos, usted encontrará que las personas simplemente no se ajustan al equipo y no pueden continuar, pero es importante no adelantar conclusiones sobre esto en una etapa temprana del proyecto.
3. *Inclusión* Las personas contribuyen efectivamente cuando sienten que otros las escuchan y que sus propuestas se toman en cuenta. Es importante desarrollar un ambiente laboral donde se consideren todas las visiones, incluso las del personal más joven.
4. *Honestidad* Como administrador, siempre debe ser honesto acerca de lo que está bien y lo que está mal en el equipo. También debe ser honesto respecto a su nivel de conocimiento técnico y voluntad para comunicar al personal más conocimiento cuando sea necesario. Si trata de encubrir la ignorancia o los problemas, con el tiempo, éstos saldrán a la luz y perderá el respeto del grupo.

La gestión de personal es algo que debe basarse en la experiencia, en lugar de aprenderse en un libro. El objetivo de esta sección, y la siguiente sobre el trabajo en equipo, es introducir de manera sencilla algunos de los problemas más importantes de la gestión de personal y del equipo que afectan la gestión de proyectos de software. Se espera que este material lo sensibilice a algunos de los problemas que pueden encontrar los administradores cuando se enfrentan con equipos de individuos técnicamente talentosos.

### 22.2.1 Motivación del personal

Como administrador de proyecto, usted necesitará motivar a las personas con quienes trabaja, de manera que éstas contribuyan con lo mejor de sus habilidades. Motivación significa organizar el trabajo y el ambiente laboral para alentar a los individuos a desempeñarse tan efectivamente como sea posible. Si las personas no están motivadas, no estarán interesadas en la actividad que realizan. Así que trabajarán con lentitud, y será más probable que cometan errores y que no contribuyan con las metas más amplias del equipo o la organización.

Para fomentar este ánimo, hay que saber un poco acerca de qué motiva a la gente. Maslow (1954) sugiere que las personas se sienten motivadas para cubrir sus necesidades, las cuales se ordenan en una serie de niveles, como se muestra en la figura 22.7. Los niveles más bajos de esta jerarquía representan necesidades fundamentales de alimentación, sueño, agua, etcetera, y la necesidad de sentirse seguro en un ambiente. Las necesidades sociales se relacionan con el hecho de sentirse parte de un grupo social. Las

---

necesidades de estima representan

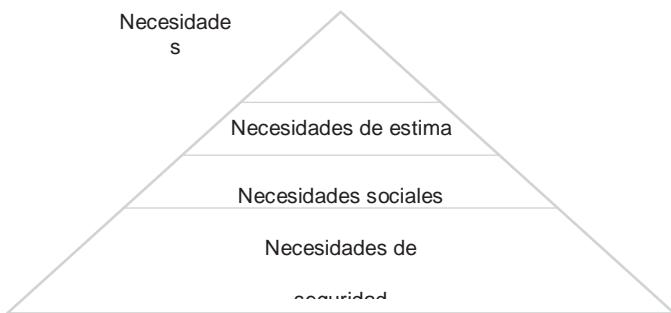


Figura 22.7 Jerarquía de necesidades humanas

la necesidad de sentirse respetado por otros, y las necesidades de autorrealización tienen que ver con el desarrollo personal. Las personas requieren cubrir las necesidades de nivel inferior, como el hambre, antes de las necesidades de nivel superior, que son más abstractas.

Las personas que trabajan en organizaciones de desarrollo de software, por lo general, no están hambrientas ni sedientas ni físicamente amenazadas por su ambiente. Por lo tanto, asegurarse de que se cubren las necesidades sociales, de estima y autorrealización de las personas es más importante desde un punto de vista administrativo.

1. Para satisfacer las necesidades sociales, es preciso dar a las personas tiempo para reunirse con sus compañeros de trabajo y proporcionarles lugares de socialización. Esto es relativamente sencillo cuando todos los miembros de un equipo de desarrollo trabajan en el mismo lugar. Aunque cada vez es más común que los miembros del equipo no laboren en el mismo edificio o ni siquiera en la misma ciudad o estado. Pueden trabajar para diferentes organizaciones o la mayor parte del tiempo desde casa.

Los sistemas de redes sociales y las teleconferencias facilitan las comunicaciones, pero los sistemas electrónicos son más efectivos una vez que las personas se conocen entre sí. Por lo tanto, es necesario programar algunas reuniones cara a cara en etapas tempranas del proyecto, de manera que la gente pueda interactuar directamente con otros miembros del equipo. Mediante esta interacción directa, las personas se vuelven parte de un grupo social y aceptan las metas y prioridades de dicho grupo.

2. Para cubrir las necesidades de estima, es necesario demostrar a las personas que son valoradas por la organización. El reconocimiento público de los logros es una forma simple, aunque efectiva, de hacer esto. Desde luego, las personas también deben sentir que se les paga de acuerdo con sus habilidades y experiencia.
3. Finalmente, para cubrir las necesidades de autorrealización, es necesario dar responsabilidad a las personas por su trabajo, asignarles tareas demandantes (pero no imposibles) y ofrecer un programa de capacitación donde puedan desarrollar sus habilidades. La capacitación es una importante influencia motivadora, pues la mayoría de las personas desean adquirir nuevos conocimientos y aprender nuevas habilidades.



### Estudio de caso: Motivación

Alice es administradora de un proyecto de software en una compañía que desarrolla sistemas de alarma. Esta compañía quiere ingresar al creciente mercado de tecnología de apoyo para ayudar a que los ancianos y las personas discapacitadas vivan de manera independiente. Se solicitó a Alice dirigir un equipo de seis desarrolladores que pueden diseñar nuevos productos basados en tecnología de alarma de la empresa.

El proyecto de tecnología de apoyo de Alice comienza bien. Dentro del equipo se desarrollan tanto buenas relaciones de trabajo como nuevas ideas creativas.

El equipo decide desarrollar un sistema de mensajería entre pares usando televisores digitales vinculados a la red de alarma para comunicaciones. Sin embargo, meses después en el proyecto, Alice nota que Dorothy, una experta en diseño de hardware, comienza a llegar tarde al trabajo, que la calidad de su trabajo se deteriora y, cada vez más, no parece comunicarse con otros miembros del equipo.

Alice platica el problema informalmente con otros miembros del equipo para tratar de descubrir si cambiaron las circunstancias personales de Dorothy, y si esto pudiera afectar su trabajo. Como ninguno de ellos sabe algo al respecto, Alice decide hablar con Dorothy para tratar de entender el problema.

Después de negar al inicio que exista un problema, Dorothy admite que perdió interés en el trabajo. Ella esperaba desarrollar y usar sus habilidades de creación de interfaces de hardware. Sin embargo, debido a la gestión del producto que se eligió, tiene poca oportunidad de hacer esto. Básicamente, trabaja como programadora C con otros miembros del equipo.

Admite que aunque el trabajo es desafiante, ella está preocupada de no desarrollar sus habilidades de interfaces. Además, piensa que será difícil encontrar un trabajo

Figura 22.8  
Motivación individual

pierde interés en el trabajo y en el grupo en su conjunto. La calidad de su trabajo declina y se vuelve inaceptable. Esta situación debe enfrentarse rápidamente. Si no se resuelve el problema, los otros miembros del grupo se sentirán insatisfechos y pensarán que están haciendo una parte del trabajo que no les corresponde.

En este ejemplo, Alice trata de descubrir si las circunstancias personales de Dorothy son el origen del problema. Comúnmente, las dificultades personales afectan la motivación, porque las personas no pueden concentrarse en su trabajo. Tal vez haya que darles tiempo y apoyo para resolver dichos conflictos, aunque también hay que dejar en claro que siguen teniendo una responsabilidad con su empleador.

El problema de motivación de Dorothy es bastante común cuando los proyectos se desarrollan en una dirección imprevista. Las personas que esperan hacer un tipo de trabajo pueden terminar por hacer algo completamente diferente. Esto se vuelve un problema cuando los miembros del equipo quieren desarrollar sus habilidades en una forma que es distinta al rumbo que tomó el proyecto. En tales circunstancias, el administrador podría decidir que un integrante debe abandonar el equipo y encontrar oportunidades en alguna otra parte. Sin embargo, en este ejemplo, Alice decide tratar de convencer a Dorothy de que ampliar su experiencia es un paso positivo en su carrera. Concede a Dorothy más autonomía de diseño y organiza cursos de capacitación en ingeniería de software que le darán más oportunidades después de terminado su proyecto actual.

## Modelo de Madurez de Capacidad del Personal

El Modelo de Madurez de Capacidad del Personal (P-CMM) es un marco para valorar qué tan bien las organizaciones administran el desarrollo de su personal. Pone de relieve las mejores prácticas en la gestión de personal y ofrece una base para que las organizaciones mejoren los procesos de administración de los recursos humanos.

<http://www.SoftwareEngineering-9.com/Web/Management/P-CMM.html>

El modelo de motivación de Maslow es útil sólo hasta cierto punto, ya que adopta un punto de vista exclusivamente personal de la motivación. No considera de manera adecuada el hecho de que las personas se sienten parte de una organización, un grupo profesional y una o más culturas. Ésta no es tan sólo una cuestión de cubrir necesidades sociales: las personas pueden sentirse motivadas al ayudar a un grupo a lograr metas compartidas.

Ser miembro de un grupo cohesivo es enormemente motivador para la mayoría de la gente. Con frecuencia, las personas con trabajos satisfactorios disfrutan ir a trabajar, porque están motivadas por la gente con la que trabajan y por la actividad que realizan. Por lo tanto, además de pensar en la motivación individual, también hay que considerar cómo un grupo en su conjunto puede motivarse para lograr las metas de la organización. En la siguiente sección se estudian los conflictos de administrar grupos.

El tipo de personalidad también influye en la motivación. Bass y Duntzman (1963) clasifican a los profesionales en tres tipos:

1. Personas orientadas a las tareas, quienes están motivadas por el trabajo que realizan. En la ingeniería de software se trata de personas que están motivadas por el reto intelectual de desarrollar software.
2. Personas orientadas hacia sí mismas, quienes están motivadas principalmente por el éxito y el reconocimiento personales. Están interesadas en el desarrollo del software como un medio para lograr sus propias metas. Esto no significa que esos individuos sean egoístas y sólo piensen en sus propios intereses. En vez de ello, suelen tener metas a plazos más largos, como el avance profesional; de esta manera, se sienten motivados a tener éxito en su trabajo para conseguir dichas metas.
3. Personas orientadas a la interacción, quienes están motivadas por la presencia y las acciones de los compañeros de trabajo. Conforme el desarrollo de software se vuelve más centrado en el usuario, los individuos orientados a la interacción se involucran más en la ingeniería de software.

Las personas orientadas a la interacción comúnmente disfrutan al trabajar como parte de un grupo, mientras que quienes están orientadas a las tareas o hacia sí mismas preferirían actuar individualmente. Las mujeres tienen más probabilidad que los hombres de estar orientadas a la interacción. Con frecuencia son comunicadores más efectivos. En el estudio de caso de la figura 22.10 se muestra la mezcla de estos diferentes tipos de personalidad en grupos.

**EDITORIAL:**

guiaceneval.mx



La motivación de cada individuo se constituye con elementos de cada clase, pero, por lo regular, un tipo de motivación domina en algún momento dado. Sin embargo, los individuos pueden cambiar. Por ejemplo, el personal técnico que siente que no se recompensa de manera adecuada puede volverse orientado hacia sí mismo y anteponer el interés personal a los asuntos técnicos. Si un grupo trabaja particularmente bien, las personas orientadas hacia sí mismas pueden volverse más orientadas a la interacción.

## 22.3 Trabajo en equipo

La mayor parte del software profesional se desarrolla mediante equipos de proyecto, cuyo número de miembros varía entre dos y varios cientos de personas. Sin embargo, como es imposible que todos los integrantes de un grupo grande trabajen en conjunto en un solo problema, los equipos grandes habitualmente se dividen en grupos más pequeños. Cada grupo es responsable de desarrollar parte del sistema global. Como regla general, los grupos del proyecto de ingeniería de software no deben tener más de 10 miembros. Cuando se usan grupos pequeños se reducen los problemas de comunicación. Todos conocen a todos los demás, y el grupo en su conjunto puede reunirse en torno a una mesa para estudiar el proyecto y el software que desarrollan.

Conformar un grupo que tiene el equilibrio justo de habilidades técnicas, experiencia y personalidades es una tarea administrativa fundamental. Sin embargo, los grupos existentes son mucho más que una colección de individuos con el equilibrio justo de habilidades. Un buen equipo es cohesivo y tiene espíritu de grupo. Las personas que participan están motivadas tanto por el éxito del grupo como por sus metas personales.

En un grupo cohesivo, los miembros piensan que el equipo es más importante que los individuos que lo integran. Los miembros de un grupo cohesivo bien liderado son leales al equipo. Se identifican con las metas del grupo y con los demás miembros. Tratan de proteger al grupo, como entidad, de cualquier interferencia externa. Esto hace que el grupo sea sólido y pueda enfrentar problemas y situaciones inesperadas.

Los beneficios de crear un grupo cohesivo son:

1. *El grupo puede establecer sus propios estándares de calidad* Puesto que dichos estándares se establecen por consenso, éstos tienen más probabilidad de respetarse que los estándares externos impuestos sobre el grupo.
2. *Los individuos aprenden de los demás y se apoyan mutuamente* Las personas en el grupo aprenden de los demás. Las inhibiciones causadas por la ignorancia se minimizan mientras se promueve el aprendizaje mutuo.
3. *El conocimiento se comparte* Puede mantenerse la continuidad si sale un miembro del grupo. Otros en el grupo pueden tomar el control de las tareas críticas para asegurar que el proyecto no se altere en forma considerable.
4. *Se alientan la refactorización y el mejoramiento continuo* Los miembros del grupo trabajan de manera colectiva para entregar resultados de alta calidad. Los miembros revisan y mejoran el código fuente originalmente el diseño o programa.

### Estudio de caso: Espíritu de equipo

Alice, una experimentada administradora de proyecto, comprende la importancia de crear un grupo cohesivo. Conforme se desarrolla un nuevo producto, ella aprovecha la oportunidad de hacer participar a todos los miembros del grupo en la especificación y el diseño del producto, al hacer que analicen las posibles tecnologías con los miembros de sus familias de mayor experiencia. Alice también los alienta a llevar a esos familiares para reunirse con otros integrantes del grupo de desarrollo.

También organiza almuerzos mensuales para todos los integrantes del grupo. Dichos almuerzos son una oportunidad para que todos los miembros del equipo se reúnan de manera informal, platicuen acerca de los temas que les preocupan y se conozcan mutuamente. En el almuerzo, Alice comunica al grupo lo que sabe acerca de las noticias, políticas, estrategias, etcétera, de la organización. Luego, cada miembro del equipo explica brevemente lo que hace, y el grupo examina un tema general, como las ideas de un nuevo producto de los padres mayores.

Cada determinado tiempo, Alice organiza un día fuera para el grupo, en que el equipo pasa dos días en actualización tecnológica. Cada miembro del equipo prepara una actualización sobre una tecnología relevante y la presenta al grupo.

Figura 22.9 Cohesión grupal

Los buenos administradores de proyecto siempre tratan de alentar la cohesión grupal. Pueden organizar eventos sociales para los miembros del grupo y sus familias, además de establecer un sentido de identidad al ponerle nombre al grupo y establecer una compatibilidad y un territorio grupales, u organizar actividades explícitas de construcción grupal, como actividades deportivas y juegos.

Una de las formas más efectivas de fomentar la cohesión es ser comprensivo. Esto significa que hay que tratar a los miembros de los grupos como responsables y confiables, y poner la información a libre disposición. En ocasiones, los administradores sienten que no pueden revelar cierta información a todos los miembros del grupo. Esto invariablemente crea un clima de desconfianza. El simple intercambio de información es una forma efectiva de hacer que las personas se sientan valoradas y se reconozcan como parte de un grupo.

En el estudio de caso de la figura 22.9 se puede ver un ejemplo de esto. Alice connaît reuniones informales regulares donde informa a los otros miembros del grupo lo que sucede. Se asegura de involucrar a las personas en el desarrollo del producto al pedirles que proporcionen nuevas ideas derivadas de su propia experiencia familiar. Los días fuera también son buenas formas de promover la cohesión: las personas se relajan al reunirse mientras se ayudan mutuamente a aprender nuevas tecnologías.

Si un grupo es efectivo o no, en cierta medida, depende de la naturaleza del proyecto y la organización que realiza el trabajo. Si una organización se encuentra en un estado de turbulencia por reorganizaciones constantes e inseguridad laboral, es muy difícil que los miembros del equipo se enfoquen en el desarrollo de software. Sin embargo, aparte de los conflictos del proyecto y la organización, existen tres factores genéricos que afectan el trabajo en equipo:

- 1. Las personas en el grupo* Se necesita una combinación de personas en un grupo de proyecto, puesto que el desarrollo de software implica diversas actividades, como negociación con clientes, programación, pruebas y documentación.



2. *La organización grupal* Un grupo debe organizarse de forma que los individuos puedan contribuir con sus mejores habilidades y completar las tareas como se esperaba.
3. *Comunicaciones técnicas y administrativas* Es esencial la óptima comunicación entre los miembros del grupo, y entre el equipo de ingeniería de software y otras partes interesadas en el proyecto.

Como en todos los conflictos administrativos, reunir al equipo correcto no garantiza el éxito del proyecto. Muchas otras cosas pueden salir mal, incluidos los cambios en los negocios y en el ambiente empresarial. Sin embargo, si no se presta la atención debida a la composición, la organización y las comunicaciones del grupo, aumenta la probabilidad de que el proyecto enfrente dificultades.

## **Selección de los miembros del grupo**

La labor de un administrador o líder de equipo es crear un grupo cohesivo y organizar a los miembros del grupo para que puedan trabajar en conjunto de manera efectiva. Esto implica crear un grupo con el equilibrio correcto de habilidades técnicas y personalidades, así como organizarlo para que los miembros trabajen adecuadamente en conjunto. En ocasiones, se contrata a personas externas a la organización; no obstante, con más frecuencia, los grupos de ingeniería de software se componen de empleados actuales que tienen experiencia adquirida en otros proyectos. Con todo, los administradores pocas veces tienen absoluta libertad en la selección del equipo. Con frecuencia deben recurrir a las personas que estén disponibles en la compañía, aun cuando no sean ideales para el puesto.

Como se estudió en la sección 22.2.1, muchos ingenieros de software están motivados principalmente por su trabajo. Por lo tanto, los grupos de desarrollo a menudo están compuestos por personas que cuentan con ideas propias sobre qué problemas técnicos deben resolverse. Esto se refleja en los problemas que se reportan regularmente relacionados con estándares de interfaz ignorados, sistemas rediseñados conforme se codifican, arreglo innecesario del sistema, etcétera.

Un grupo con personalidades complementarias puede trabajar mejor que un grupo seleccionado exclusivamente por la habilidad técnica. Es probable que las personas que están motivadas por el trabajo sean las más fuertes técnicamente. Las personas que son orientadas hacia sí mismas tal vez serán mejores para impulsar el trabajo hacia delante para terminar la tarea. Las personas orientadas a la interacción ayudan a facilitar las comunicaciones dentro del grupo. Se considera que en el grupo es particularmente importante contar con personas orientadas a la interacción. A éstas les gusta hablar con los demás y son capaces de detectar tensiones y diferencias en una etapa temprana, antes de que tengan serias repercusiones sobre el grupo.

En el estudio de caso de la figura 22.10, se narró cómo Alice, la administradora del proyecto, trató de crear un grupo con personalidades complementarias. Este grupo particular tiene una buena combinación de personas orientadas a la interacción y a las tareas, pero, en la figura 22.8, ya se describió cómo la personalidad de Dorothy, quien está orientada hacia sí misma, causó problemas porque no realizó el trabajo esperado. También el rol de tiempo parcial de Fred, como experto de dominio, podría ser un problema del grupo. Él está principalmente interesado en los retos técnicos, así que posible-

### Estudio de caso: Composición de grupo

Al crear un grupo para el desarrollo de tecnología de apoyo, Alice está consciente de la importancia de seleccionar miembros con personalidades complementarias. Cuando entrevista a miembros potenciales del grupo, trata de valorar si están orientados a las tareas, hacia sí mismos u orientados a la interacción. Ella siente que su personalidad está orientada hacia sí misma, porque considera que el proyecto es una forma de hacerse notar ante los altos ejecutivos y buscar una promoción. Por ende, busca una o quizás dos personalidades orientadas a la interacción, e individuos orientados a las tareas para completar el equipo. La valoración final a la que llegó fue:

Alice: orientada hacia sí misma

Brian: orientado a tareas

Bob: orientado a tareas

Carol: orientada a la interacción

Dorothy: orientada hacia sí misma

Ed: orientado a la interacción

Figura 22.10  
Composición  
de grupo

mente no interactúe bien con otros miembros del grupo. El hecho de que no siempre es parte del equipo significa que puede no relacionarse bien con las metas del equipo.

En ocasiones es imposible elegir un grupo con personalidades complementarias. Si éste es el caso, el administrador del proyecto tiene que controlar al grupo de modo que las metas individuales no se antepongan a los objetivos de la organización y del grupo. Este control es más sencillo de lograr si todos los miembros del grupo participan en cada etapa del proyecto. La iniciativa individual es más factible cuando los miembros del grupo reciben instrucciones sin estar al tanto de la parte que desempeña su tarea en el proyecto global.

Por ejemplo, suponga que a un ingeniero de software se le asigna un diseño de programa para codificar, y observa lo que parecen ser posibles mejoras que podrían hacerse al diseño. Si implementa dichas mejoras sin comprender las razones del diseño original, cualquier cambio, aun cuando sea muy bien intencionado, puede tener implicaciones adversas para otras partes del sistema. Si todos los miembros del grupo participan en el diseño desde el principio, comprenderán por qué se tomaron las decisiones de diseño. Entonces, los miembros podrán identificarse con dichas decisiones en lugar de oponerse a ellas.

### Organización del grupo

La forma en que se organiza un grupo influye en las decisiones que toma dicho grupo, las maneras como se intercambia la información y las interacciones entre el grupo de desarrollo y los participantes externos del proyecto. Las preguntas organizacionales importantes para los administradores de proyecto incluyen:

1. ¿El administrador del proyecto debe ser el líder técnico del grupo? El líder técnico o arquitecto del sistema es responsable de las decisiones técnicas críticas tomadas durante el desarrollo del software. En ocasiones, el administrador del proyecto tiene





### Contratar a las personas correctas

Con frecuencia los administradores del proyecto son responsables de seleccionar al personal en la organización que se unirá a su equipo de ingeniería de software. Conseguir a las mejores personas posibles en este proceso es muy importante, pues las malas decisiones de selección implican un grave riesgo para el proyecto.

Los factores clave que deben influir en la selección de personal son: educación y capacitación, dominio de aplicación y experiencia tecnológica, habilidad de comunicación, adaptabilidad y habilidad para resolver problemas.

<http://www.SoftwareEngineering-9.com/Web/Management/Selection.html>

la habilidad y experiencia para desempeñar este papel. Sin embargo, en caso de grandes proyectos, es mejor asignar a un ingeniero con experiencia como el arquitecto del proyecto, quien tomará la responsabilidad del liderazgo técnico.

2. ¿Quién se encargará de tomar las decisiones técnicas críticas, y cómo se tomarán?  
¿Las decisiones las tomará el arquitecto del sistema, el administrador del proyecto o se llegará a un consenso entre un rango más amplio de miembros del equipo?
3. ¿Cómo se manejarán las interacciones con los participantes externos y los altos directivos de la compañía? En muchos casos, el administrador del proyecto será el responsable de dichas interacciones, asistido, si acaso, por el arquitecto del sistema. Sin embargo, un modelo de organización alternativo incluye una función exclusiva para las relaciones externas, lo que supone asignar para dicha función a una persona con habilidades de interacción adecuadas.
4. ¿Cómo es posible que los grupos logren integrar a personas que no se localizan en el mismo lugar? Ahora es común que los grupos incluyan a miembros de diferentes organizaciones y personas que trabajan desde casa o en oficinas compartidas. Esto debe tomarse en cuenta en los procesos de toma de decisiones grupales.
5. ¿Cómo puede compartirse el conocimiento a través del grupo? La organización del grupo afecta el intercambio de información, pues determinadas formas de organización son mejores que otras para compartir. Sin embargo, conviene evitar demasiado intercambio de información, ya que las personas pueden sobresaturarse y la información excesiva podría distraerlos de sus labores.

Los grupos de programación pequeños, por lo general, están organizados en una forma bastante informal. El líder del grupo participa en el desarrollo de software con los otros miembros del grupo. En un grupo informal, todo el equipo analiza el trabajo a realizar, y las tareas se asignan según la habilidad y la experiencia. Los miembros del grupo con mayor jerarquía pueden ser responsables del diseño arquitectónico. No obstante, el diseño y la implementación detallados son compromisos del miembro del equipo que se asigna a una tarea particular.

**Esta guía de estudio, fue producida por Mami para tu para tu. mx Todos los Derechos Reservados.**

Los grupos de programación extrema (Beck, 2000) siempre son grupos informales. Los apasionados de XP afirman que la estructura formal inhibe el intercambio de

información. En XP, muchas decisiones que normalmente se consideran como decisiones administrativas (por ejemplo, las decisiones acerca del calendario) se delegan a los



miembros del grupo. Los programadores trabajan en pares para diseñar un código y asumir la responsabilidad conjunta de los programas que desarrollaron.

Los grupos informales pueden ser muy exitosos, en particular cuando la mayoría de los miembros del grupo son experimentados y competentes. Tal grupo toma decisiones por consenso, lo que mejora la cohesión y el rendimiento. Sin embargo, si un grupo está compuesto principalmente por miembros inexpertos e incompetentes, la informalidad puede ser un obstáculo porque no existe autoridad definida para dirigir el trabajo, lo que causa una falta de coordinación entre los miembros del grupo y, posiblemente, una eventual falla del proyecto.

Los grupos jerárquicos son grupos que comparten una estructura jerárquica con el líder del grupo en la parte superior del escalafón. El líder tiene autoridad más formal que los miembros del grupo y así puede dirigir el trabajo. Existe una clara estructura organizacional, y las decisiones se toman hacia la parte superior de la jerarquía y se aplican por las personas que están más abajo en la jerarquía. Las comunicaciones, ante todo, son instrucciones del personal ejecutivo y existe relativamente poca comunicación ascendente, es decir, desde los niveles más bajos hacia los niveles superiores en la jerarquía.

Este enfoque funciona bien cuando un problema bien entendido puede descomponerse fácilmente en subproblemas en los que las soluciones se desarrollan en diferentes partes de la jerarquía. En dichas situaciones se requiere muy poca comunicación a través de la jerarquía. Sin embargo, tales situaciones, en proporción, son poco comunes en la ingeniería de software por las siguientes razones:

1. Los cambios al software requieren con frecuencia cambios en varias partes del sistema y esto conduce a una discusión y negociación en todos los niveles de la jerarquía.
2. Las tecnologías de software cambian tan rápido que muchas veces el personal más joven conoce más de la tecnología que el personal experimentado. Las comunicaciones descendentes pueden significar que el administrador del proyecto no vislumbra las oportunidades de usar nuevas tecnologías. El personal más joven puede frustrarse debido a que considera obsoletas las tecnologías usadas para el desarrollo.

Las organizaciones grupales democráticas y jerárquicas no reconocen formalmente que puede haber diferencias muy grandes de habilidad técnica entre los miembros del grupo. Los mejores programadores pueden ser hasta 25 veces más productivos que los peores programadores. Tiene sentido aprovechar las capacidades de los mejores elementos en la forma más efectiva y brindarles tanto apoyo como sea posible. Uno de los primeros modelos organizacionales que tenía la intención de ofrecer apoyo fue el llamado equipo programador jefe.

Para aprovechar de manera más efectiva a los programadores con mayor habilidad, Baker (1972) y otros (Aron, 1974; Brooks, 1975) sugieren que los equipos deben construirse en torno a un programador jefe individual con gran habilidad. El principio subyacente del equipo programador jefe es que el personal habilidoso y experimentado debe ser responsable de todo el desarrollo del software. Sus integrantes no deben preocuparse por cuestiones rutinarias y deben tener buen apoyo técnico y administrativo para realizar su trabajo. Deben enfocarse en el software a desarrollar y no perder mucho tiempo en reuniones externas.



### El ambiente laboral físico

El ambiente donde trabajan las personas afecta tanto las comunicaciones grupales como la productividad individual. Los espacios de trabajo individuales son mejores para la concentración en el trabajo técnico detallado, pues las personas tienen menos probabilidad de distraerse por interrupciones. Sin embargo, los espacios de trabajo compartidos son mejores para las comunicaciones. Un ambiente laboral bien diseñado toma en consideración ambas necesidades.

<http://www.SoftwareEngineering-9.com/Web/Management/workspace.html>

No obstante, la organización en equipo programador jefe es, desde la perspectiva del autor, demasiado dependiente del programador en jefe y su asistente. Otros miembros del equipo a quienes no se dé suficiente responsabilidad pueden desmotivarse porque sienten que sus habilidades son desaprovechadas. No tienen la información para hacer frente si las cosas salen mal y no se les da la oportunidad de participar en la toma de decisiones. Existen riesgos significativos para el proyecto asociados con esta organización grupal y esto podría superar cualquier beneficio que aporte este tipo de organización.

### Comunicaciones grupales

Es absolutamente esencial que los miembros del grupo se comuniquen efectiva y eficientemente entre sí y con otras partes interesadas en el proyecto. Los miembros del grupo deben intercambiar información acerca del estatus de su trabajo, las decisiones de diseño que se tomaron y los cambios a las decisiones de diseño previas. Tienen que resolver los problemas que surjan con otros interesados en el proyecto e informar a éstos sobre los cambios al sistema, grupo y planes de entrega. La buena comunicación ayuda también a fortalecer la cohesión del grupo. Los miembros del grupo llegan a entender las motivaciones, fortalezas y debilidades de otras personas en el grupo.

La efectividad y la eficiencia de las comunicaciones están influidas por:

1. *Tamaño del grupo* Conforme el grupo crece, se hace más difícil que los miembros se comuniquen de manera efectiva. El número de vínculos de comunicación de un canal es  $n^*$  ( $n - 1$ ), donde  $n$  es el tamaño del grupo, de manera que, con un grupo de ocho miembros, existen 56 posibles rutas de comunicación. Esto significa que es muy posible que algunas personas rara vez se comuniquen entre sí. Las diferencias de estatus entre los miembros del grupo significan que las comunicaciones con frecuencia son unidireccionales. Los administradores e ingenieros experimentados tienden a dominar las comunicaciones con el personal menos experimentado, quienes pueden tener reticencias para iniciar una conversación o hacer puntualizaciones críticas.

2. *Estructura del grupo* Las personas en los grupos estructurados de manera formal se comunican más efectivamente que los individuos en grupos con una estructura jerárquica formal. En los grupos jerárquicos, las comunicaciones tienden a fluir hacia arriba y abajo de la jerarquía. Las



personas en el mismo nivel tal vez no se

comuniquen entre sí. Éste es un problema particular en un proyecto grande con varios grupos de desarrollo. Si las personas que trabajan en diferentes subsistemas se comunican sólo a través de sus administradores, hay más probabilidad de demoras y malas interpretaciones.

3. *Composición del grupo* Las personas con los mismos tipos de personalidad (estudiados en la sección 22.2) pueden chocar y, como resultado, las comunicaciones se inhiben. Además, por lo regular, la comunicación es mejor en los grupos integrados por personas de uno y otro género (Marshall y Hestlin, 1975) que en los grupos formados por miembros de un solo género. Con frecuencia, las mujeres son más orientadas a la interacción que los hombres y suelen actuar como controladoras y facilitadoras de la interacción para el grupo.
4. *El ambiente laboral físico* La organización del centro de trabajo es un factor importante para facilitar o inhibir las comunicaciones. Véase la página Web del libro para más información.
5. *Los canales de comunicación disponibles* Existen muchas formas diferentes de comunicación: cara a cara, correo electrónico, documentos formales, teléfono y tecnologías Web 2.0, como las redes sociales y los wikis. Conforme los equipos de proyecto se distribuyen cada vez más, con miembros de equipo que trabajan en lugares remotos, es necesario utilizar varias tecnologías para facilitar las comunicaciones.

Los administradores de proyecto trabajan por lo general bajo plazos estrechos y, en consecuencia, tratan de usar canales de comunicación que no consuman mucho tiempo. Por lo tanto, se apoyan en reuniones y documentos formales para transmitir la información al personal del proyecto y las partes interesadas. Aunque éste tal vez sea un enfoque eficiente para la comunicación desde la perspectiva de un administrador de proyecto, comúnmente no es muy efectivo. A menudo existen buenas razones por las que las personas no pueden asistir a las reuniones y, por lo tanto, no escuchan la presentación. Los documentos extensos casi nunca se leen, debido a que los lectores no saben si los documentos son relevantes. Al producirse varias versiones del mismo documento, los lectores encuentran difícil hacer un seguimiento de los cambios.

La comunicación efectiva se logra cuando las comunicaciones son bidireccionales, y las personas implicadas pueden discutir los conflictos y la información, y establecer una comprensión común de las proposiciones y los problemas. Esto se logra mediante reuniones, aunque éstas suelen estar dominadas por las personalidades poderosas. En ocasiones no es práctico citar con escasa anticipación a reuniones. Cada vez más equipos de proyecto incluyen miembros ubicados a distancia, lo que dificulta las reuniones.

Para contrarrestar estos problemas y apoyar el intercambio de información, se puede recurrir a tecnologías Web, como wikis y blogs. Los wikis respaldan la creación y edición de documentos en colaboración, mientras que los blogs apoyan las discusiones generadas por preguntas y comentarios hechos por los miembros del grupo. Los wikis y blogs permiten a los miembros del proyecto y a los participantes externos intercambiar información, sin importar su ubicación. Ayudan a gestionar la información y a seguir la huella de los hilos de discusión, que con frecuencia se vuelven confusos cuando se realizan por correo electrónico. Para resolver conflictos que necesiten discusión, se puede recurrir a la mensajería instantánea y las teleconferencias, las cuales pueden organizarse fácilmente.



## PUNTOS CLAVE

- La buena gestión de proyectos de software es esencial si los proyectos de ingeniería de software deben desarrollarse dentro del plazo y el presupuesto establecidos.
- La gestión del software es distinta de otras administraciones de ingeniería. El software es intangible. Los proyectos pueden ser novedosos o innovadores, así que no hay un conjunto de experiencias para orientar su gestión. Los procesos de software no son tan maduros como los procesos de ingeniería tradicionales.
- La gestión del riesgo se reconoce ahora como una de las áreas más importantes de la gestión de un proyecto.
- La gestión del riesgo implica la identificación y valoración de los grandes riesgos del proyecto para establecer la probabilidad de que ocurran; también supone identificar y valorar las consecuencias para el proyecto si dichos riesgos surgen. Debe hacer planes para evitar, gestionar o enfrentar los posibles riesgos.
- Las personas sienten motivadas por la interacción con otros individuos, el reconocimiento de la gestión y sus pares, y al recibir oportunidades de desarrollo personal.
- Los grupos de desarrollo de software deben ser bastante pequeños y cohesivos. Los factores clave que influyen en la efectividad de un grupo son sus integrantes, la forma en que está organizado y la comunicación entre los miembros.
- Las comunicaciones dentro de un grupo están influenciadas por factores como el estatus de los miembros del grupo, el tamaño del grupo, la composición por género del grupo, las personalidades y los canales de comunicación disponibles.

## LECTURAS SUGERIDAS

*The Mythical Man Month (Anniversary Edition)*. Los problemas de la gestión del software siguen en gran medida invariables desde la década de 1960, y este libro es uno de los mejores sobre el tema. Una interesante y clara explicación de la gestión de un grupo de los primeros y más grandes proyectos de software: el sistema operativo OS/360 de IBM. La edición de aniversario (publicada 20 años después de la edición original de 1975) incluye otros ensayos clásicos de Brooks. (F.P. Brooks, 1995, Addison-Wesley.)

*Software Project Survival Guide*. Ésta es una explicación bastante pragmática de la gestión del software que incluye buenos consejos prácticos para los administradores de proyecto con antecedentes de ingeniería de software. Es fácil de leer y entender. (S. McConnell, 1998, Microsoft Press.)

*Peopleware: Productive Projects and Teams, 2nd edition*. Ésta es una nueva edición del libro clásico acerca de la importancia de tratar a las personas de manera adecuada cuando se administran proyectos de software. Es uno de los pocos libros que reconocen la importancia del lugar donde trabajan las personas. Es enormemente recomendable. (T. DeMarco y T. Lister, 1999, Dorset House.)

*Waltzing with Bears: Managing Risk on Software Projects*. Una introducción muy práctica y fácil de leer respecto a los riesgos y la gestión del riesgo. (T. DeMarco y T. Lister, 2003, Dorset House.)

## EJERCICIOS

Explique por qué la intangibilidad de los sistemas de software plantea problemas especiales para la gestión de proyectos de software.

Explique por qué los mejores programadores no siempre son los mejores administradores de software. Tal vez le resulte útil basar su respuesta en la lista de actividades administrativas de la sección 22.1.

Con los casos de problemas de proyecto reportados en la literatura, mencione las dificultades y los errores administrativos que ocurrieron en dichos proyectos de programación fallidos. (Se sugiere que comience con *The Mythical Man Month*, de Fred Brooks).

Además de los riesgos que se muestran en la figura 22.1, identifique al menos otros seis riesgos posibles que pudieran surgir en los proyectos de software.

Los contratos de precio fijo, donde el contratista ofrece un precio fijo para completar un desarrollo de sistema, permiten desplazar los riesgos del proyecto del cliente al contratista. Si algo sale mal, el contratista debe pagar. Sugiera cómo el uso de tales contratos puede aumentar la probabilidad de que surjan riesgos del producto.

Explique por qué mantener informados a todos los miembros de un grupo acerca del progreso y de las decisiones técnicas en un proyecto ayuda a mejorar la cohesión del grupo.

¿Qué problemas considera que surgirían en los equipos de programación extrema, donde muchas decisiones administrativas se delegan en los miembros del equipo?

Escriba un estudio de caso, con el estilo usado aquí, para ilustrar la importancia de las comunicaciones en un equipo de proyecto. Suponga que algunos miembros del equipo trabajan a distancia y no es posible reunir a todo el equipo en el corto plazo.

Su administrador le pide entregar software en un plazo que sólo podrá cumplir si pide a su equipo de proyecto trabajar tiempo extra sin remuneración. Todos los miembros del equipo tienen hijos pequeños. Discuta si debe aceptar esta demanda de su administrador o si debe persuadir a su equipo de acceder su tiempo a la organización en lugar de dedicarlo a sus familias. ¿Qué factores pueden ser significativos en su decisión?

Como programador, sele ofrece una promoción a un puesto de gestión de proyectos, pero siente que puede hacer una aportación más efectiva en un papel técnico más que administrativo. Discuta si debe aceptar la promoción.

## REFERENCIAS

Aron, J. D. (1974). *The Program Development Process*. Reading, Mass.: Addison-Wesley.

Baker, F.T. (1972). "Chief Programmer Team Management of Production Programming". *IBM Systems J.*, 11 (1), 56–73.



- Bass, B. M. y Dunteman, G. (1963). "Behaviouringroupsasafunctionofself,interactionand task orientation". *J. Abnorm. Soc. Psychology.*, **66**(4), 19–28.
- Beck, K. (2000). *Extreme Programming Explained*. Reading, Mass.: Addison-Wesley.
- Boehm, B. W. (1988). "A Spiral Model of Software Development and Enhancement". *IEEE Computer*, **21** (5), 61–72.
- Brooks, F. P. (1975). *The Mythical Man Month*. Reading, Mass.: Addison-Wesley.
- Hall, E. (1998). *Managing Risk: Methods for Software Systems Development*. Reading, Mass.: Addison-Wesley.
- Marshall, J. E. y Heslin, R. (1975). "Boys and Girls Together. Sexual composition and the effect of density on group size and cohesiveness". *J. of Personality and Social Psychology*, **35**(5), 952–61.
- Maslow, A. A. (1954). *Motivation and Personality*. Nueva York: Harper and Row.
- Ould, M. (1999). *Managing Software Quality and Business Risk*. Chichester: John Wiley & Sons.

## Lectura 8. Gestión de la calidad



---

## 24

# Gestión de la calidad

### Objetivos

El objetivo de este capítulo es introducirlo a la gestión de calidad y la medición del software. Al estudiar este capítulo:

- estará al tanto del proceso de gestión de calidad y sabrá por qué es importante la planeación de calidad;
- comprenderá que la calidad del software se ve afectada por el proceso de desarrollo del software utilizado;
- conocerá la importancia de los estándares en el proceso de gestión de calidad y aprenderá cómo se usan los estándares en el aseguramiento de calidad;
- distinguirá la forma en que se utilizan las revisiones e inspecciones como mecanismo para garantizar la calidad del software;
- identificará cómo pueden ser útiles las mediciones en la valoración de algunos atributos de calidad del software y las limitaciones actuales de medición del software.

### Contenido

Calidad del software  
Estándares de software  
Revisiones e inspecciones  
Medición y métricas del software

Los problemas de calidad del software se descubrieron inicialmente en la década de 1960 con el desarrollo de los primeros grandes sistemas de software, y han continuado invadiendo la ingeniería de software a partir de esa década. El software entregado era lento y poco fiable, difícil de mantener y de reutilizar. El descontento con esta situación condujo a la adopción de técnicas formales de gestión de calidad del software, desarrolladas a partir de métodos usados en la industria manufacturera. Estas técnicas de gestión de calidad, en conjunto con nuevas tecnologías y mejores pruebas de software, llevaron a progresos significativos en el nivel general de calidad del software.

La gestión de calidad del software para los sistemas de software tiene tres intereses fundamentales:

1. A nivel de organización, la gestión de calidad se ocupa de establecer un marco de proceso y estándares de organización que conducirán a software de mejor calidad. Esto supone que el equipo de gestión de calidad debe tener la responsabilidad de definir los procesos de desarrollo del software a usar, los estándares que deben aplicarse al software y la documentación relacionada, incluyendo los requerimientos, el diseño y el código del sistema.
2. A nivel del proyecto, la gestión de calidad implica la aplicación de procesos específicos de calidad y la verificación de que continúen dichos procesos planeados; además, se ocupa de garantizar que los resultados del proyecto estén en conformidad con los estándares aplicables a dicho proyecto.
3. A nivel del proyecto, la gestión de calidad se ocupa también de establecer un plan de calidad para un proyecto. El plan de calidad debe establecer metas de calidad para el proyecto y definir cuáles procesos y estándares se usarán.

Los términos *aseguramiento de calidad* y *control de calidad* se utilizan ampliamente en la industria manufacturera. El aseguramiento de calidad (QA, por las siglas de *quality assurance*) es la definición de procesos y estándares que deben conducir a la obtención de productos de alta calidad y, en el proceso de fabricación, a la introducción de procesos de calidad. El control de calidad es la aplicación de dichos procesos de calidad para eliminar aquellos productos que no cuentan con el nivel requerido de calidad.

En la industria de software, diversas compañías y sectores industriales interpretan de maneras diferentes el aseguramiento de calidad y el control de calidad. En ocasiones, el aseguramiento de calidad representa simplemente la definición de procedimientos, procesos y estándares cuyo objetivo es asegurar el logro de calidad del software. En otros casos, el aseguramiento de calidad incluye también todas las actividades de gestión de configuración, verificación y validación aplicadas después de que un equipo de desarrollo entrega un producto. En este capítulo se usa el término *aseguramiento de calidad* para incluir verificación y validación, y los procesos de que la comprobación de procedimientos de calidad se aplicó de manera adecuada. Se evita el término “control de calidad”, puesto que esta expresión no se usa mucho en la industria del software.

En la mayoría de las compañías, el equipo QA es el responsable de administrar el proceso de pruebas de liberación. Como se explicó en el capítulo 8, esto significa que se



aplican las pruebas del software antes de que éste se libere a los clientes. El equipo es responsable de comprobar que las pruebas del sistema cubran los requerimientos y de mantener los registros adecuados del proceso de pruebas. Como en el capítulo 8 se estudiaron las pruebas de liberación, en este apartado no se trata este aspecto del aseguramiento de calidad.

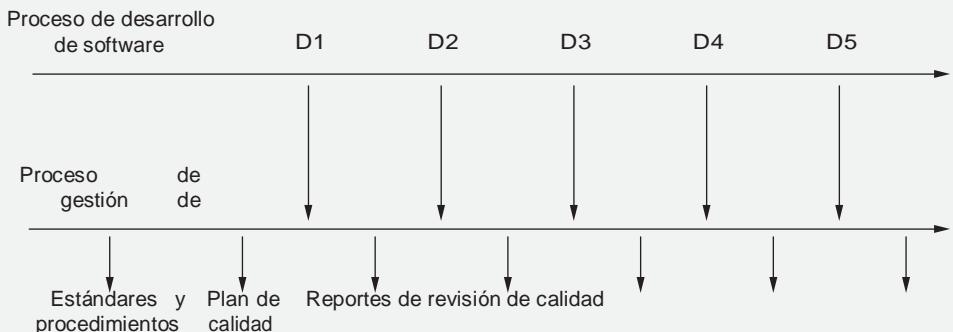


Figura 24.1  
Gestión de calidad  
y desarrollo  
de software

La gestión de calidad proporciona una comprobación independiente sobre el proceso de desarrollo de software. El proceso de gestión de calidad verifica los entregables del proyecto para garantizar que sean consistentes con los estándares y las metas de la organización (figura 24.1). El equipo QA debe ser independiente del equipo de desarrollo para que pueda tener una perspectiva objetiva del software. Esto les permite reportar la calidad del software sin estar influidos por los conflictos de desarrollo del software.

De preferencia, el equipo de gestión de calidad no debe asociarse con algún grupo de desarrollo particular; sin embargo, tiene la responsabilidad ante toda la organización por la administración de la calidad. El equipo debe ser independiente y reportarse ante la administración ubicada sobre el nivel del administrador del proyecto. La razón es que los administradores de proyecto tienen que mantener el presupuesto y el calendario del proyecto. Si surgen problemas, pueden estar tentados a comprometer la calidad del producto para cumplir con el calendario. Un equipo de gestión de calidad independiente garantiza que las metas de calidad de la organización no estén comprometidas a corto plazo por consideraciones de presupuesto y calendario. Sin embargo, en compañías más pequeñas, esto es prácticamente imposible. La gestión de calidad y el desarrollo de software están inevitablemente vinculados con las personas que tienen responsabilidades tanto de desarrollo como de calidad.

La planeación de calidad es el proceso de desarrollar un plan de calidad para un proyecto. El plan de calidad debe establecer las cualidades deseadas de software y describir cómo se valorarán. Por lo tanto, define lo que realmente significa software de "alta calidad" para un sistema particular. Sin esta definición, los ingenieros pueden hacer diferentes suposiciones, algunas veces conflictivas, sobre cuáles atributos del producto reflejan las características de calidad más importantes. La planeación de calidad formalizada es parte integral de los procesos de desarrollo basados en un plan. No obstante, los métodos ágiles adoptan un enfoque menos formal para la gestión de calidad.

Humphrey (1989), en su clásico libro referente a la gestión del software, sugiere un bosquejo de estructura para un plan de calidad. Éste incluye:

- Introducción del producto* Una descripción del producto, la pretensión de su mercado y las expectativas de calidad para el producto.
- Planes del producto* Indican las fechas de entrega críticas y las responsabilidades para el producto, junto con planes para distribución y



3. *Descripciones de procesos* Describen los procesos y estándares de desarrollo y servicio que deben usarse para diseño y gestión del producto.
4. *Metas de calidad* Las metas y los planes de calidad para el producto, incluyendo una identificación y justificación de los atributos esenciales de calidad del producto.
5. *Riesgos y gestión del riesgo* Los riesgos clave que pueden afectar la calidad del producto y las acciones a tomar para enfrentar dichos riesgos.

Los planes de calidad, que se desarrollan como parte del proceso de planeación general del proyecto, difieren en detalle dependiendo del tamaño y tipo de sistema que se desarrolló. Sin embargo, cuando escribe planes de calidad, debe tratar de mantenerlos tan breves como sea posible. Si el documento es demasiado amplio, las personas no lo leerán y, en consecuencia, se anulará el propósito de generar un plan de calidad.

Algunas personas consideran que la calidad del software puede lograrse mediante procesos establecidos basados en estándares de organización y procedimientos de calidad asociados que verifican el seguimiento de dichos estándares mediante el equipo de desarrollo de software. Su argumento es que los estándares se dirigen a la buena práctica de ingeniería de software y que seguir esta buena práctica conducirá a productos de alta calidad. No obstante, en la práctica, se considera que en la gestión de calidad hay mucho más que estándares y burocracia asociados para asegurar que éstos se sigan.

Aunque los estándares y procesos son importantes, los administradores de calidad deben enfocarse también a desarrollar una “cultura de calidad” en la que todo responsable del desarrollo del software se comprometa a lograr un alto nivel de calidad del producto. Deben exhortar a los equipos a asumir la responsabilidad de la calidad de su trabajo y desarrollar nuevos enfoques para el mejoramiento de la calidad. A pesar de que los estándares y procedimientos son la base de la gestión de calidad, los buenos administradores de calidad reconocen que existen aspectos intangibles a la calidad del software (elegancia, legibilidad, etcétera) que no pueden expresarse en estándares. Deben apoyar a la gente interesada en aspectos intangibles de calidad e impulsar el comportamiento profesional en todos los miembros del equipo.

La gestión de la calidad formalizada es particularmente importante para los equipos que diseñan grandes sistemas de larga duración, los cuales tardan varios años en desarrollarse. La documentación de la calidad es un registro de lo que cada subgrupo realizó en el proyecto. Ayuda a las personas a comprobar que no se olvidaron tareas importantes o que un grupo no hizo suposiciones incorrectas acerca de lo que hicieron otros grupos. La documentación de la calidad también es un medio de comunicación durante la vida del sistema. Permite a los grupos responsables de la evolución del sistema encontrar las pruebas y comprobaciones que el equipo de desarrollo debe implementar.

Para sistemas más pequeños, la gestión de calidad es aún importante, aunque puede adoptarse un enfoque más informal. No se necesita tanto papeleo, puesto que un equipo de desarrollo pequeño puede comunicarse de manera informal. El aspecto de calidad clave para el desarrollo de sistemas pequeños es establecer una cultura de calidad y asegurarse de que todos los miembros del equipo tienen un enfoque positivo sobre la calidad del software.

## 24.1 Calidad del software

La industria manufacturera estableció los fundamentos de la gestión de calidad para mejorar ésta en los productos que se fabricaban. Como parte de ello se desarrolló una definición de calidad, que se basa en la conformidad con una especificación de producto detallada (Crosby, 1979) y la noción de tolerancia. La suposición subyacente era que los productos podían especificarse por completo y establecerse procedimientos que comprobaran si un producto manufacturado cumplía o no con su especificación. Desde luego, los productos nunca cumplirán exactamente una especificación, pues se permite cierta tolerancia. Si el producto era “casi bueno”, se clasificaba como aceptable.

La calidad del software no es directamente comparable con la calidad en la fabricación. La idea de tolerancia no es aplicable a los sistemas digitales y es prácticamente imposible llegar a una conclusión objetiva sobre si un sistema de software cumple o no su especificación, por las siguientes razones:

1. Como se explicó en el capítulo 4, referente a la ingeniería de requerimientos, es difícil escribir especificaciones de software completas y sin ambigüedades. Los desarrolladores y clientes de software pueden interpretar los requerimientos de diferentes formas y tal vez sea imposible llegar a acuerdos acerca de si el software se desarrolló conforme a su especificación.
2. Por lo general, las especificaciones integran requerimientos de varias clases de participantes. Dichos requerimientos son un compromiso ineludible y tal vez no incluyan los requerimientos de todos los grupos de participantes. Por lo tanto, las partes interesadas excluidas quizá perciban el sistema como uno de mala calidad, a pesar de que implementa los requerimientos acordados.
3. Es imposible medir de manera directa ciertas características de calidad (por ejemplo, mantenibilidad) y, por ende, no pueden especificarse plenamente sin ambigüedades. En la sección 24.4 se estudian las dificultades de la medición.

Debido a estos problemas, la valoración de calidad del software es un proceso subjetivo en que el equipo de gestión de calidad tiene que usar su juicio para decidir si se logró un nivel aceptable de calidad. El equipo de gestión de calidad debe considerar si el software se ajusta o no a su propósito pretendido. Esto implica responder preguntas sobre las características del sistema. Por ejemplo:

1. ¿En el proceso de desarrollo se siguieron los estándares de programación y documentación?
2. ¿El software se verificó de manera adecuada?
3. ¿El software es suficientemente confiable para utilizarse?
4. ¿El rendimiento del software es aceptable para uso normal?



Protección	Comprendibilidad	Portabilidad
Seguridad	Comprobabilidad	Usabilidad
Fiabilidad	Adaptabilidad	Reusabilidad
Flexibilidad	Modularidad	Eficiencia
Robustez	Complejidad	Facilidad para que el usuario aprenda a utilizarlo

Figura 24.2 Atributos de calidad del software

5. ¿El software es utilizable?
6. ¿El software está bien estructurado y es comprensible?

Existe la suposición general en la gestión de calidad del software de que el sistema se pondrá a prueba en contra de sus requerimientos. La decisión acerca de si entregar o no la funcionalidad requerida debe basarse en los resultados de dichas pruebas. Por lo tanto, el equipo QA debe revisar las pruebas que se desarrollaron y examinar los registros de pruebas para verificar que éstas se hayan realizado de manera apropiada. En algunas organizaciones el equipo de gestión de calidad es responsable de las pruebas del sistema, pero, en ocasiones, un grupo de pruebas de sistema separado es responsable de esto.

La calidad subjetiva de un sistema de software se basa principalmente en sus características no funcionales. Esto refleja la experiencia práctica del usuario: Si la funcionalidad del software no es lo que se esperaba, entonces los usuarios con frecuencia sólo le darán la vuelta a este asunto y encontrarán otras formas de hacer lo que quieren. Sin embargo, si el software no es fiable o resulta muy lento, entonces es prácticamente imposible que los usuarios logren sus metas.

Por consiguiente, la calidad del software no sólo se trata de si la funcionalidad de éste se implementó correctamente, sino también depende de los atributos no funcionales del sistema. Boehm y sus colaboradores (1978) indican que existen 15 importantes atributos de calidad de software, los cuales se listan en la figura 24.2. Dichos atributos se relacionan con la confiabilidad, usabilidad, eficiencia y mantenibilidad del software. Como se estudió en el capítulo 11, por lo general se considera que los atributos de confiabilidad son los atributos de calidad más importantes de un sistema. Sin embargo, también es significativo el rendimiento del software. Los usuarios rechazarán el software que sea demasiado lento.

No es posible que algún sistema se optimice para todos esos atributos; por ejemplo, mejorar la robustez puede conducir a una pérdida de rendimiento. En consecuencia, el plan de calidad debe definir los atributos de calidad más importantes para el software que se desarrollará. Tal vez la eficiencia sea crítica y tengan que sacrificarse otros factores para que se logre esto. Si lo anterior se estableció en el plan de calidad, los ingenieros que trabajan en el desarrollo pueden cooperar para lograrlo. El plan debe incluir también una definición del proceso de valoración de la calidad. Ésta debe ser una forma acordada de valorar si cierto grado de calidad, como la mantenibilidad o robustez, está presente en el producto.

Una suposición que subyace en la gestión de la calidad del software es que la calidad del software se relaciona directamente con la calidad del proceso de desarrollo del

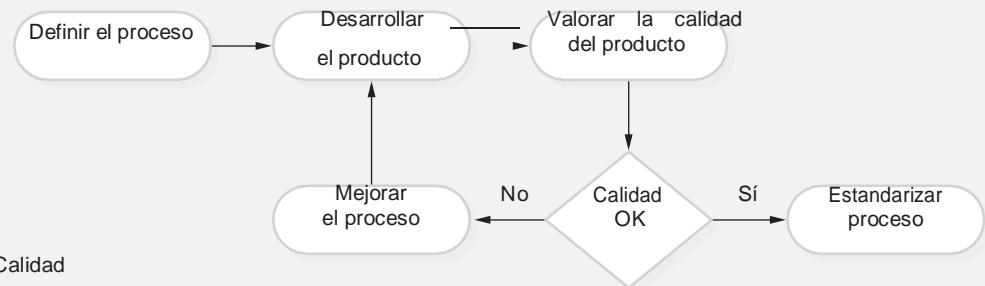


Figura 24.3 Calidad basada en el proceso

software. Esto proviene de nuevo de los sistemas fabriles, donde la calidad del producto está estrechamente relacionada con el proceso de producción. Un proceso de fabricación incluye configurar, establecer y operar las máquinas implicadas en el proceso. Una vez que las máquinas operan correctamente, se sigue de manera natural la calidad del producto. Entonces se mide la calidad del producto y el proceso se modifica hasta que se logra el nivel de calidad necesario. La figura 24.3 ilustra este enfoque basado en el proceso para obtener la calidad del producto.

En la manufactura existe un claro vínculo entre el proceso y la calidad del producto, ya que el proceso es relativamente sencillo de estandarizar y monitorizar. Una vez calibrados los sistemas de fabricación, pueden operar una y otra vez para generar productos de alta calidad; sin embargo, el software no se manufactura, se diseña. Por lo tanto, en el desarrollo del software es más compleja la relación entre calidad de proceso y calidad del producto. El diseño del software es un proceso creativo más que mecánico, pues es significativa la influencia de las habilidades y la experiencia individuales. Factores externos, como la novedad de una aplicación o la premura por el lanzamiento comercial de un producto, también afectan la calidad de éste sin importar el proceso usado.

No hay duda de que el proceso de desarrollo utilizado tiene una influencia importante sobre la calidad del software, y que los buenos procesos tienen más probabilidad de conducir a software de buena calidad. La gestión de la calidad y el mejoramiento del proceso pueden conducir a menores defectos en el software a desarrollar. Sin embargo, es difícil valorar los atributos de calidad del software, como la mantenibilidad, sin usar el software durante un largo periodo. En consecuencia, es difícil decir cómo las características del proceso influyen en dichos atributos. Más aún, debido al papel del diseño y la creatividad en el proceso de software, la estandarización del proceso en ocasiones puede extinguir la creatividad, lo cual, lejos de elevar la calidad, conducirá a un software de calidad inferior.

## Estándares 24.2 de software

Los estándares de software tienen una función muy importante en la gestión de calidad del software. Como se indicó, un aspecto importante del aseguramiento de calidad es la definición o selección de estándares que deben aplicarse al proceso de desarrollo de software o al producto de software. Como parte de este proceso QA, también pueden elegirse herramientas y métodos para apoyar el uso de dichos estándares. Una vez



seleccionados éstos



## Estándares de documentación

Los documentos del proyecto son una forma tangible de describir las diferentes representaciones de un sistema de software (requerimientos, UML, código, etcétera) y su proceso de producción. Los estándares de documentación definen la organización de diferentes tipos de documentos, así como

el formato del documento. Son importantes porque facilitan la comprobación de que no se haya omitido material importante de los documentos y garantiza que los documentos del proyecto tengan una apariencia común. Los estándares pueden desarrollarse para el proceso de escribir documentos, los documentos en sí y el intercambio de documentos.

<http://www.SoftwareEngineering-9.com/Web/QualityMan/docstandards.html>

para su uso, deben definirse procesos específicos de proyecto para monitorizar el uso de los estándares y comprobar que éstos se siguieron.

Los estándares de software son importantes por tres razones:

1. Los estándares reflejan la sabiduría que es de valor para la organización. Se basan en conocimiento sobre la mejor o más adecuada práctica para la compañía. Con frecuencia, este conocimiento se adquiere sólo después de gran cantidad de ensayo y error. Configurarla dentro de un estándar, ayuda a la compañía a reutilizar esta experiencia y a evitar errores del pasado.
2. Los estándares proporcionan un marco para definir, en un escenario particular, lo que significa el término “calidad”. Como se dijo, la calidad del software es subjetiva, y al usar estándares se establece una base para decidir si se logró un nivel de calidad requerido. Desde luego, esto depende del establecimiento de estándares que reflejen las expectativas del usuario para la confiabilidad, la usabilidad y el rendimiento del software.
3. Los estándares auxilian la continuidad cuando una persona retoma el trabajo iniciado por alguien más. Los estándares aseguran que todos los ingenieros dentro de una organización adopten las mismas prácticas. En consecuencia, se reduce el esfuerzo de aprendizaje requerido al iniciarse un nuevo trabajo.

Existen dos tipos de estándares de ingeniería de software relacionados que pueden definirse y usarse en la gestión de calidad del software:

1. *Estándares del producto* Se aplican al producto de software a desarrollar. Incluyen estándares de documentos (como la estructura de los documentos de requerimientos), estándares de documentación (como el encabezado de un comentario estándar para una definición de clase de objeto) y estándares de codificación, los cuales definen cómo debe



usarse un lenguaje de programación.

- 2 *Estándares de proceso* Establecen los procesos que deben seguirse durante el desarrollo del software. Deben especificar cómo es una buena práctica de desarrollo. Los estándares de proceso pueden incluir definiciones de especificación, procesos de diseño y validación, herramientas de soporte de proceso y una descripción de los documentos que deben escribirse durante dichos procesos.

Estándares de producto	Estándares de proceso
Formato de revisión de diseño	Realizar revisión de diseño
Estructura de documento de requerimientos	Enviar nuevo código para construcción de sistema
Formato de encabezado por método	Proceso de liberación de versión
Estilo de programación Java	Proceso de aprobación del plan del proyecto
Formato de plan de proyecto	Proceso de control de cambio
Formato de solicitud de cambio	Proceso de registro de prueba

Figura 24.4  
Estándares de producto  
y proceso

Los estándares deben entregar valor, en la forma de calidad aumentada del producto. No hay razón para definir estándares que sean costosos en términos de tiempo y esfuerzo, pues aplicarlos sólo conduce a mejoras secundarias en la calidad. Los estándares de producto deben diseñarse de forma que puedan aplicarse y comprobarse de manera efectiva en cuanto a costos, y los estándares de proceso deben incluir la definición de procesos que comprueben que se siguieron dichos estándares.

El desarrollo de estándares internacionales de ingeniería de software, por lo general, es un proceso prolongado en el que se reúnen los interesados en el estándar, elaboran borradores para comentar y, finalmente, acuerdan el estándar. Organismos nacionales e internacionales, como U.S. DoD, ANSI, BSI, OTAN y el IEEE, apoyan la determinación de estándares. Se trata de estándares generales que pueden aplicarse a través de varios proyectos. Entidades tales como la OTAN y otras organizaciones de defensa pueden requerir que sus propios estándares se usen en el desarrollo de contratos que suscriben con compañías de software.

Se han desarrollado estándares nacionales e internacionales que incluyen la terminología de ingeniería de software, lenguajes de programación como Java y C++, anotaciones como los símbolos de diagramación, procedimientos para derivar y escribir requerimientos de software, procedimientos de aseguramiento de calidad, y procesos de verificación y validación de software (IEEE, 2003). Estándares más especializados, como IEC 61508 (IEC, 1998), se desarrollaron para sistemas críticos de protección y seguridad.

Los equipos de gestión de calidad que elaboran estándares para alguna compañía, por lo general deben basar los estándares de dicha compañía en estándares nacionales e internacionales. Al usar estándares internacionales como punto de partida, el equipo de aseguramiento de calidad debe redactar un manual de estándares, el cual debe definir los estándares que necesita su organización. En la figura 24.4 se muestran ejemplos de estándares que podrían incluirse en dicho manual.

En ocasiones, los ingenieros de software consideran los estándares como demasiado prescriptivos y realmente poco relevantes para la actividad técnica del desarrollo de software. Esto es probable sobre todo cuando los estándares de proyecto requieren documentación y registro del trabajo tediosos. Aunque en general están de acuerdo con la necesidad de los estándares, los ingenieros encuentran a menudo razones para señalar que los estándares no necesariamente son adecuados para su proyecto particular. Para



minimizar el descontento y alentar la participación en los estándares, los administradores de calidad que establezcan los estándares deben dar los siguientes pasos:

1. *Involucrar a los ingenieros de software en la selección de estándares de producto* Si los desarrolladores comprenden por qué se seleccionaron los estándares, tienen más probabilidad de comprometerse con éstos. De preferencia, los documentos de estándares no deben establecer sólo el estándar a seguir, sino también deben incluir comentarios que expliquen por qué se tomaron las decisiones de estandarización.
2. *Revisar y modificar regularmente los estándares para reflejar las tecnologías cambiantes* Los estándares son costosos de desarrollar y tienden a guardarse como reliquias en un manual de estándares de una compañía. Debido a los costos y la discusión requeridos, muchas veces hay reticencia para cambiarlos. Aunque un manual de estándares es esencial, debe evolucionar para reflejar las circunstancias y la tecnología cambiantes.
3. *Ofrecer herramientas de software para dar soporte a los estándares* Los desarrolladores encuentran con frecuencia que los estándares son una pesadilla cuando la adhesión a ellos incluye un tedioso trabajo manual que podría hacerse mediante una herramienta de software. Si está disponible el soporte para herramientas, se requiere muy poco esfuerzo para seguir los estándares de desarrollo de software. Por ejemplo, los estándares de documento pueden implementarse mediante estilos de procesador de texto.

Diferentes tipos de software necesitan distintos procesos de desarrollo, puesto que los estándares deben ser adaptables. No hay razón para prescribir una forma particular de trabajar si es inadecuada para un proyecto o equipo de proyecto. Cada administrador de proyecto debe tener la autoridad de modificar los estándares de proceso de acuerdo con las circunstancias individuales. Sin embargo, cuando se hacen cambios, es importante garantizar que dichos cambios no conduzcan a una pérdida de calidad del producto. Esto afectará la relación de una empresa con sus clientes y conducirá probablemente a un aumento en los costos del proyecto.

El administrador del proyecto y el administrador de calidad pueden evitar problemas en los estándares al planear cuidadosamente la calidad oportuna en el proyecto. Deben decidir cuál de los estándares de la organización debe usarse sin cambio, cuáles deben modificarse y cuáles ignorarse. Es posible que deban crearse nuevos estándares en respuesta a requerimientos del cliente o del proyecto. Por ejemplo, tal vez se requieran estándares para especificaciones formales si no se han usado en proyectos anteriores.

#### 24.2.1 El marco de estándares ISO 9001

Existe un conjunto internacional de estándares que pueden utilizarse en el desarrollo de los sistemas de administración de calidad en todas las industrias, llamado ISO 9000. Los estándares ISO 9000 pueden aplicarse a varias organizaciones, desde las industrias

manufactureras hasta las de servicios. ISO 9001, el más general de dichos estándares, se aplica a organizaciones que diseñan, desarrollan y mantienen productos, incluido software. El estándar ISO 9001 se desarrolló originalmente en 1987, y su revisión más reciente fue en 2008.





Figura 24.5 Procesos centrales ISO 9001

El estándar ISO 9001 no es en sí mismo un estándar para el desarrollo de software, sino un marco para elaborar estándares de software. Establece principios de calidad total, describe en general el proceso de calidad, y explica los estándares y procedimientos organizacionales que deben determinarse. Éstos tienen que documentarse en un manual de calidad de la organización.

La revisión principal del estándar ISO 9001 reorientó en 2000 el estándar hacia nueve procesos centrales (figura 24.5). Si una organización quiere estar conforme con el estándar ISO 9001, debe documentar cómo se relacionan sus procesos con dichos procesos centrales. También deberá definir y mantener registros que demuestren que se siguieron los procesos organizacionales establecidos. El manual de calidad de la compañía tiene que describir los procesos relevantes y los datos de proceso que deben recopilarse y conservarse.

El estándar ISO 9001 no define ni prescribe los procesos de calidad específicos que deben usarse en una compañía. Para estar de conformidad con ISO 9001, una compañía debe especificar los tipos de proceso que se muestran en la figura 24.5 y tener procedimientos que demuestren que se siguen sus procesos de calidad. Esto permite flexibilidad a través de sectores industriales y diversos tamaños de compañías. Pueden definirse estándares de calidad que sean adecuados para el tipo de software a desarrollar. Las compañías pequeñas pueden tener procesos no burocráticos y estar en conformidad con ISO 9001. Sin embargo, esta flexibilidad significa que no es posible hacer suposiciones sobre las similitudes o diferencias entre los procesos en distintas compañías que acatan ISO 9001. Algunas compañías tienen procesos de calidad muy rígidos con registros detallados, mientras que otras son mucho menos formales, con poca documentación adicional.

En la figura 24.6 se muestran las relaciones entre ISO 9001, manuales de calidad organizacional y planes de calidad de proyecto individuales. Este diagrama se derivó de un modelo presentado por Ince (1994), quien explica cómo puede usarse el estándar general ISO 9001 como base para procesos de gestión de calidad de software. Bamford y Dielbler (2003) explican cómo puede aplicarse el más reciente estándar ISO 9001:2000 en las compañías de software.

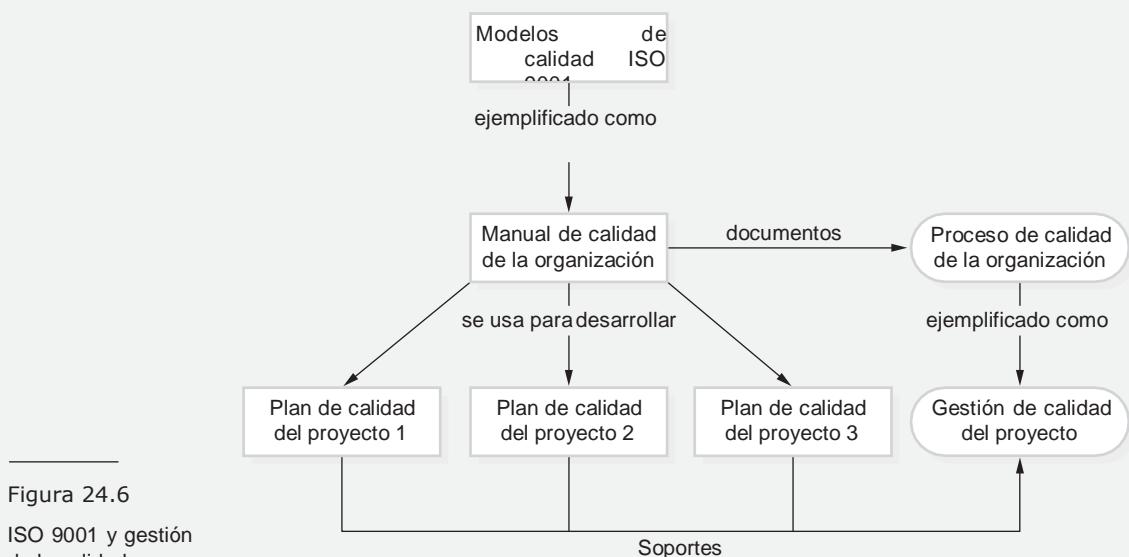


Figura 24.6

ISO 9001 y gestión de la calidad

Algunos clientes de software demandan que sus proveedores tengan la certificación ISO 9001. Así, los clientes podrán estar seguros de que la compañía que desarrolla el software tiene un sistema de gestión de calidad aprobado. Autoridades de acreditación independiente examinan los procesos de gestión de calidad y la documentación de proceso, y deciden si dichos procesos abarcan todas las áreas especificadas en ISO 9001. Si es así, certifican que los procesos de calidad de una compañía, definidos en el manual de calidad, concuerdan con el estándar ISO 9001.

Algunas personas consideran que la certificación ISO 9001 significa que la calidad del software producido por compañías certificadas será mejor que el derivado de compañías no certificadas. Esto no precisamente es cierto. El estándar ISO 9001 se enfoca en garantizar que la organización tenga procedimientos de gestión de calidad y que siga dichos procedimientos. No hay seguridad de que las compañías con certificación ISO 9001 empleen las mejores prácticas de desarrollo de software o que sus procesos conduzcan a software de alta calidad.

Por ejemplo, una compañía podría definir estándares de cobertura de pruebas que especifiquen que todos los métodos en los objetos deben llamarse al menos una vez. Lamentablemente, este estándar puede cumplirse mediante pruebas de software incompletas, que no incluyen pruebas con diferentes parámetros de métodos. En tanto se sigan los procedimientos de prueba definidos y se conserven registros de las pruebas realizadas, la compañía podría tener la certificación ISO 9001. Esta certificación define la calidad como la conformidad con estándares, y toma en cuenta la calidad como la advierten los usuarios del software.

Los métodos ágiles, que evitan la documentación y se enfocan en el código a desarrollar, tienen poco en común con los procesos de calidad formal que se examinan en ISO 9001. Se ha hecho cierto trabajo para reconciliar estos enfoques (Stalhane y Hanssen, 2008), pero la comunidad de desarrollo ágil en general se opone a lo que considera una carga burocrática de la conformidad con los estándares. Por esta razón, las compañías



que usan métodos de desarrollo ágil se preocupan pocas veces por la certificación ISO 9001.

## Revisiones e inspecciones

Las revisiones e inspecciones son actividades QA que comprueban la calidad de los entregables del proyecto. Esto incluye examinar el software, su documentación y los registros del proceso para descubrir errores y omisiones, así como observar que se siguieron los estándares de calidad. Como se estudió en los capítulos 8 y 15, revisiones e inspecciones se usan junto con las pruebas del programa como parte del proceso general de verificación y validación del software.

Durante una revisión, un grupo de personas examinan el software y su documentación asociada en busca de problemas potenciales y la falta de conformidad con los estándares. El equipo de revisión realiza juicios informados sobre el nivel de calidad de un entregable de sistema o de proyecto. Entonces los administradores de proyecto pueden usar dichas valoraciones para tomar decisiones de planeación y asignar recursos al proceso de desarrollo.

Las revisiones de calidad se basan en documentos que se elaboraron durante el proceso de desarrollo del software. Al igual que las especificaciones, el diseño o el código del software, también pueden revisarse los modelos de proceso, planes de prueba, procedimientos de gestión de configuración, estándares de proceso y manuales de usuario. La revisión debe comprobar la coherencia e integridad de los documentos o el código objeto de prueba, y asegurarse de que se han seguido las normas de calidad.

Sin embargo, la revisión no sólo es acerca de la comprobación de conformidad con las normas, sino también se utiliza para ayudar a descubrir problemas y omisiones en la documentación del software o proyecto. Las conclusiones de la revisión deben registrarse formalmente como parte del proceso de gestión de calidad. Si se descubren problemas, los comentarios de los revisores deben pasar al autor del software o a quien resulte responsable de corregir los errores u omisiones.

El propósito de las revisiones e inspecciones es mejorar la calidad del software, no de valorar el rendimiento de los miembros del equipo de desarrollo. La revisión es un proceso público de detección de errores, comparado con el proceso más privado de prueba de componentes. Es necesario que los errores cometidos por los individuos se revelen a todo el equipo de programación. Para garantizar que todos los desarrolladores participen constructivamente con el proceso de revisión, los administradores de proyecto tienen que ser sensibles a las preocupaciones individuales. Deben desarrollar una cultura de trabajo que brinde apoyo y no culpar cuando se descubran errores.

Aunque una revisión de calidad ofrece a la administración datos sobre el software a desarrollar, las revisiones de calidad no son lo mismo que las revisiones de avance administrativo. Como se expuso en el capítulo 23, las revisiones de avance comparan el avance real en un proyecto de software frente al avance planeado. Su principal preocupación es si el proyecto entregará o no el software útil a tiempo y dentro del presupuesto. Las revisiones de progreso toman en cuenta factores externos, y circunstancias cambiantes pueden significar que el software en fase de desarrollo ya no se requiera o que tenga

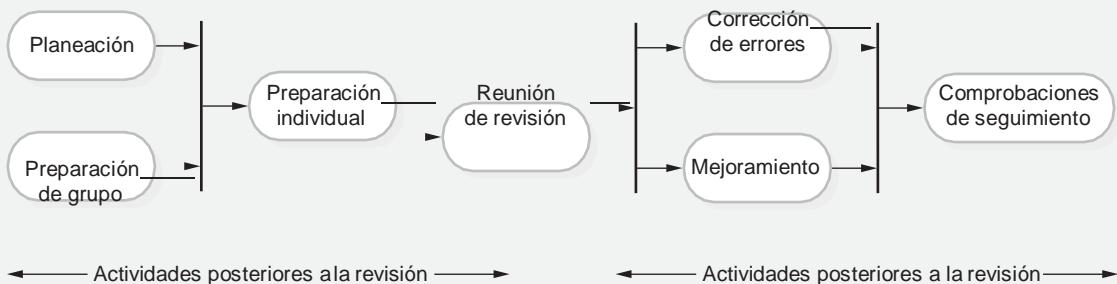


Figura 24.7

El proceso de revisión de software

que cambiarse radicalmente. Tal vez se cancelen los proyectos que desarrollaron software de alta calidad debido a cambios en la empresa o su entorno operacional.

## El proceso de revisión

Aunque existen numerosas variaciones en los detalles de las revisiones, el proceso de revisión (figura 24.7) se estructura por lo general en tres fases:

- 1. Actividades previas a la revisión** Se trata de actividades preparatorias esenciales para que sea efectiva la revisión. Por lo general, las actividades previas a la revisión se ocupan de la planeación y preparación de la revisión. La planeación de la revisión incluye establecer un equipo de revisión, organizar un tiempo, destinar un lugar para la revisión y distribuir los documentos a revisar. Durante la preparación de la revisión, el equipo puede reunirse para obtener un panorama del software a revisar. Miembros del equipo de revisión leen y entienden el software o los documentos y estándares relevantes. Trabajan de manera independiente para encontrar errores, omisiones y distanciamiento de los estándares. Si los revisores no pueden asistir a la reunión de revisión, pueden hacer sus comentarios por escrito acerca del software.
- 2. La reunión de revisión** Durante la reunión de revisión un autor del documento o programa a revisar debe repasar el documento con el equipo de revisión. La revisión en sí debe ser relativamente corta, dos horas a lo sumo. Un miembro del equipo debe dirigir la revisión y otro registrar formalmente todas las decisiones y acciones de revisión a tomar. Durante la revisión, quien dirige es responsable de garantizar que se consideren todos los comentarios escritos. La dirección de la revisión debe firmar un registro de comentarios y acciones acordados durante la revisión.
- 3. Actividades posteriores a la revisión** Despues de terminada una reunión de revisión, deben tratarse los conflictos y problemas surgidos durante la revisión. Esto puede implicar corregir bugs de software, refactorizar el software de modo que esté conforme con los estándares de calidad, o reescribir los documentos. Algunas veces, los problemas descubiertos en



una revisión de calidad son tales que es necesaria también una revisión administrativa con la finalidad de decidir si deben disponerse más recursos para corregirlos. Después de efectuar los cambios, la dirección de la revisión deberá comprobar que se hayan considerado todos los comentarios de la revisión. En ocasiones se requerirá una revisión ulterior para comprobar que los cambios realizados comprenden todos los comentarios de revisión anteriores.



## Roles en el proceso de inspección

Cuando se estableció por primera vez una inspección del programa en IBM (Fagan, 1976; Fagan, 1986), se definieron algunos roles formales para los miembros del equipo de inspección. Éstos incluían: moderador, lector de código y secretario. Otros usuarios de la inspección modificaron dichos roles, pero generalmente se acepta que una inspección debe incluir al autor del código, un inspector, un secretario y un moderador que la dirija.

<http://www.SoftwareEngineering-9.com/Web/QualityMan/roles.html>

Con frecuencia, los equipos de revisión tienen un eje de tres o cuatro personas seleccionadas como revisores principales. Un miembro debe ser un diseñador ejecutivo, quien tendrá la responsabilidad de tomar decisiones técnicas significativas. Los revisores principales pueden invitar a otros miembros del proyecto, como los diseñadores de subsistemas relacionados, para contribuir a la revisión. Tal vez no participen en la revisión de todo el documento, pero deben concentrarse en aquellas secciones que afecten su trabajo. Como alternativa, el equipo de revisión puede hacer circular el documento y pedir comentarios por escrito de un amplio número de miembros del proyecto. El administrador del proyecto necesita participar en la revisión, a menos que se anticipen problemas que requieran cambios al plan del proyecto.

El proceso de revisión anterior se apoya en todos los miembros de un equipo de desarrollo asignado y disponible para una reunión de equipo. Sin embargo, ahora es común que los equipos de proyecto estén distribuidos, a veces a lo largo del país o en distintos continentes, así que muchas veces no es práctico que los miembros del equipo se reúnan en el mismo lugar. Ante tales situaciones, pueden usarse herramientas de edición de documentos para apoyar el proceso de revisión. Los miembros del equipo usan éstos para anotar comentarios en el documento o código fuente de software. Tales comentarios son visibles para otros miembros del equipo, quienes entonces podrán aprobarlos o rechazarlos. Sólo se requeriría una conversación telefónica cuando deban resolverse desacuerdos entre los revisores.

Por lo general, el proceso de revisión en el desarrollo de software ágil es informal. En Scrum, por ejemplo, hay una junta de revisión después de completar cada iteración del software (una revisión rápida), en la que pueden exponerse los conflictos y problemas de calidad. En la programación extrema, como se estudiará en la siguiente sección, la programación en grupos de dos personas garantiza que el código se examine y revise constantemente por otro miembro del equipo. Los conflictos de calidad general también se consideran en las reuniones diarias del equipo, pero XP se apoya en individuos que toman la iniciativa para mejorar y refactorizar el código. Por lo general, los enfoques ágiles no son dirigidos por estándares, de manera que no se consideran los asuntos de cumplimiento de estándares.

La falta de procedimientos de calidad formal en los métodos ágiles supone que puede haber problemas en el uso de enfoques ágiles en compañías que desarrollaron procedimientos de gestión de calidad detallados. Las revisiones de calidad en ocasiones aplazan el ritmo del desarrollo del software, así que éstas se emplean mejor dentro de un proceso de desarrollo dirigido por un plan. En este tipo de proceso, las revisiones pueden efectuarse mientras otros trabajos se realizan paralelamente. Esto no es práctico en enfoques ágiles que se centran de manera exclusiva en el desarrollo del código.



## Inspecciones del programa

Las inspecciones del programa son “revisiones de pares” en las que los miembros del equipo colaboran para encontrar bugs en el programa en desarrollo. Como se explicó en el capítulo 8, las inspecciones pueden ser parte de los procesos de verificación y validación del software. Complementan las pruebas, puesto que no requieren la ejecución del programa. Esto quiere decir que es posible verificar versiones incompletas del sistema y comprobar representaciones como los modelos UML. Gilb y Graham (1993) sugieren que una de las formas más efectivas de usar las inspecciones es revisar los casos de prueba para un sistema. Las inspecciones permiten identificar problemas con las pruebas y, así, mejorar la efectividad de dichas pruebas en la detección de bugs de programa.

Las inspecciones del programa incluyen a miembros del equipo con diferentes antecedentes que realizan una cuidadosa revisión, línea por línea, del código fuente del programa. Buscan defectos y problemas, y los informan en una reunión de inspección. Los defectos pueden ser errores lógicos, anomalías en el código que indican una condición errónea o ciertas características que se hayan omitido del código. El equipo de revisión examina a detalle los modelos de diseño o el código del programa y destaca las anomalías y problemas a reparar.

Durante una inspección, con frecuencia se usa una lista de verificación de errores comunes de programación para enfocar la búsqueda de bugs. Esta lista de verificación se basa en ejemplos de libros, o bien, en el conocimiento de defectos normales en un dominio de aplicación común. Para diferentes lenguajes de programación se usan distintas listas de verificación, puesto que cada lenguaje tiene sus errores característicos. Humphrey (1989), en un amplio debate sobre inspecciones, ofrece algunos ejemplos de listas de verificación de inspección.

En la figura 24.8 se muestran las posibles comprobaciones que pueden hacerse durante el proceso de inspección. Gilb y Graham (1993) enfatizan que cada organización debe desarrollar su lista de verificación de inspección con base en estándares y prácticas locales. Dichas listas de verificación deben actualizarse regularmente, conforme se encuentren nuevos tipos de defectos. Los ítems en la lista de verificación varían según el lenguaje de programación, debido a diferentes niveles de comprobación posibles en el tiempo de compilación. Por ejemplo, un compilador Java comprueba que las funciones tengan el número correcto de parámetros; un compilador C no lo hace.

La mayoría de compañías que introdujeron las inspecciones descubrieron que éstas son muy efectivas para encontrar bugs. Fagan (1986) reportó que es posible detectar más del 60% de los errores en un programa mediante inspecciones informales de programa. Mills y sus colaboradores (1987) sugieren que un enfoque más formal a la inspección, con base en argumentos de exactitud, permite detectar más del 90% de los errores en un programa. McConnell (2004) compara las pruebas de unidad, en las que la tasa de detección de defectos es de alrededor del 25%, con las inspecciones, en las que la tasa de detección de defectos fue del 60%. También describe diversos estudios de caso, incluido un ejemplo en que la introducción de revisiones de pares condujo a un aumento en la productividad del 14% y una reducción en los defectos en el programa del 90 por ciento.

A pesar de su reconocida efectividad en términos de costos, muchas compañías de desarrollo de software se resisten a usar inspecciones o revisiones de pares. Los ingenieros de software con experiencia en pruebas de programa en ocasiones son reacios a aceptar que las inspecciones son más efectivas que las pruebas para la detección de defectos. Los administradores tal vez se muestren recelosos porque las inspecciones requieren

Clase de falla	Comprobación de inspección
Fallas de datos	<ul style="list-style-type: none"> <li>• ¿Todas las variables del programa seinizianantesdeusarsusvalores?</li> <li>• ¿Todas las constantes tienen nombre?</li> <li>• ¿La cota superior de los arreglos es igual al tamaño del arreglo o Valor – 1?</li> <li>• Si se usan cadenas de caracteres, ¿se asigna explícitamente un delimitador?</li> <li>• ¿Existe alguna posibilidad de desbordamiento de buffer?</li> </ul>
Fallas de control	<ul style="list-style-type: none"> <li>• Para cada enunciado condicional, ¿la condición es correcta?</li> <li>• ¿Hay certeza de que termine cada ciclo?</li> <li>• ¿Los enunciados compuestos están correctamente colocados entre paréntesis?</li> <li>• En caso de enunciados, ¿se justifican todos los casos posibles?</li> <li>• Si después de cada caso en los enunciados se requiere un paréntesis, ¿éste se incluyó?</li> </ul>
Fallas de entrada/salida	<ul style="list-style-type: none"> <li>• ¿Se usan todas las variables de entrada?</li> <li>• ¿A todas las variables de salida se les asigna un valor antes de que se produzcan?</li> <li>• ¿Entradas inesperadas pueden causar corrupción?</li> </ul>
Fallas de interfaz	<ul style="list-style-type: none"> <li>• ¿Todas las llamadas a función y método tienen el número correcto de parámetros?</li> <li>• ¿Los tipos de parámetro formal y real coinciden?</li> <li>• ¿Los parámetros están en el orden correcto?</li> <li>• Si los componentes acceden a memoria compartida, ¿tienen el mismo modelo de estructura de memoria compartida?</li> </ul>
Fallas de gestión de almacenamiento	<ul style="list-style-type: none"> <li>• Si se modifica una estructura vinculada, ¿todos los vínculos se reasignan correctamente?</li> <li>• Si se usa almacenamiento dinámico, ¿el espacio se asignó correctamente?</li> <li>• ¿El espacio se cancela explícitamente después de que ya no se requiere?</li> </ul>
Fallas de gestión de excepción	<ul style="list-style-type: none"> <li>• ¿Se tomaron en cuenta todas las posibles condiciones de error?</li> </ul>

Figura 24.8 Lista de verificación de una inspección

costos adicionales durante el diseño y desarrollo. Quizá no quieran aceptar el riesgo de que no haya ahorros correspondientes en los costos de prueba del programa.

Los procesos ágiles pocas veces usan procesos de inspección formal o revisión de pares. En vez de ello, se apoyan en los miembros del equipo que cooperan para comprobar mutuamente el código y en lineamientos informales, tales como “comprobar antes de ingresar”, lo que sugiere que los programadores deben comprobar su propio código. Los profesionales de la programación extrema argumentan que la programación en parejas es un sustituto efectivo de la inspección, ya que, en efecto, se trata de un proceso de inspección continuo. Dos personas observan cada línea de código y la comprueban antes de aceptarla.

La programación en grupos de dos conduce a un conocimiento profundo de un programa, pues ambos programadores deben entender su funcionamiento a detalle para continuar el desarrollo. En ocasiones es difícil lograr esta profundidad de conocimiento en otros procesos de inspección y, por lo tanto, la programación en grupos de dos permite



encontrar bugs que a veces no se descubrirían en inspecciones formales. Sin embargo, la programación en grupos de dos también puede conducir a malas interpretaciones de los requerimientos, en las que ambos miembros del par cometan el mismo error. Más aún, las parejas pueden tener reticencias para buscar errores, pues uno de los dos no quiere frenar el avance del proyecto. En ocasiones, las personas que participan no son tan objetivas como un equipo de inspección externo, y es probable que su habilidad para descubrir defectos esté comprometida por su cercana relación laboral.

## 24.4 Medición y métricas del software

La medición del software se ocupa de derivar un valor numérico o perfil para un atributo de un componente, sistema o proceso de software. Al comparar dichos valores unos con otros, y con los estándares que se aplican a través de una organización, es posible extraer conclusiones sobre la calidad del software, o valorar la efectividad de los procesos, las herramientas y los métodos de software.

Por ejemplo, suponga que una organización pretende introducir una nueva herramienta de prueba de software. Antes de introducir la herramienta, hay que registrar el número de defectos descubiertos de software en un tiempo determinado. Ésta es una línea de referencia para valorar la efectividad de la herramienta. Después de usar la herramienta durante algún tiempo, se repite este proceso. Si se descubren más defectos en el mismo lapso, después de introducida la herramienta, usted tal vez determine que ofrece apoyo útil para el proceso de validación del software.

La meta a largo plazo de la medición del software es usar la medición en lugar de revisiones para realizar juicios de la calidad del software. Al usar medición de software, un sistema podría valorarse preferentemente mediante un rango de métricas y, a partir de dichas mediciones, se podría inferir un valor de calidad del sistema. Si el software alcanzó un umbral de calidad requerido, entonces podría aprobarse sin revisión. Cuando es adecuado, las herramientas de medición pueden destacar también áreas del software susceptibles de mejora. Sin embargo, aún se está lejos de esta situación ideal y no hay señales de que la valoración automatizada de calidad será en el futuro una realidad previsible.

Una métrica de software es una característica de un sistema de software, documentación de sistema o proceso de desarrollo que puede medirse de manera objetiva. Los ejemplos de métricas incluyen el tamaño de un producto en líneas de código; el índice Fog (Gunning, 1962), que es una medida de la legibilidad de un pasaje de texto escrito; el número de fallas reportadas en un producto de software entregado, y el número de días-hombre requerido para desarrollar un componente de sistema.

Las métricas de software pueden ser métricas de control o de predicción. Como el nombre lo dice, las métricas de control apoyan la gestión del proceso, y las métricas de predicción ayudan a predecir las características del software. Las métricas de control se asocian por lo general con procesos de software. Ejemplos de las métricas de control o de proceso son el esfuerzo promedio y el tiempo requerido para reparar los defectos reportados. Las métricas de predicción se asocian con el software en sí y a veces se conocen como métricas de producto. Ejemplos de métricas de predicción son la complejidad ciclomática de un módulo (estudiado en el capítulo 8), la longitud promedio de los

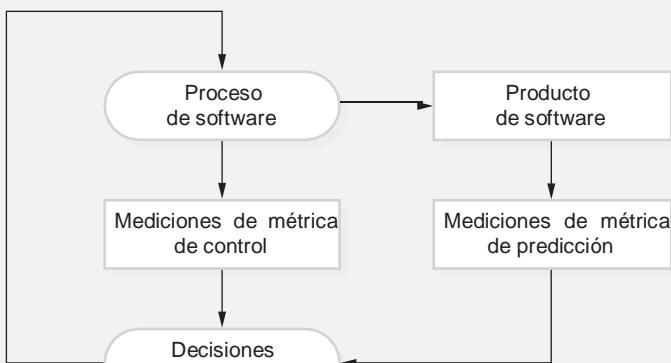


Figura 24.9 Mediciones de predicción y de control

identificadores de un programa, y el número de atributos y operaciones asociados con las clases de objetos en un diseño.

Tanto las métricas de control como las de predicción pueden influir en la toma de decisiones administrativas, como se muestra en la figura 24.9. Los administradores usan mediciones de proceso para decidir si deben hacerse cambios al proceso, y las métricas de predicción ayudan a estimar el esfuerzo requerido para hacer cambios al software. En este capítulo se estudian principalmente las métricas de predicción, cuyos valores se evalúan al analizar el código de un sistema de software. En el capítulo 26 se estudian las métricas de control y cómo se usan en el mejoramiento de procesos.

Existen dos formas en que pueden usarse las mediciones de un sistema de software:

1. *Para asignar un valor a los atributos de calidad del sistema* Al medir las características de los componentes del sistema, como su complejidad ciclomática, y luego agregar dichas mediciones, es posible valorar los atributos de calidad del sistema, tales como la mantenibilidad.
2. *Para identificar los componentes del sistema cuya calidad está por debajo de un estándar* Las mediciones pueden identificar componentes individuales con características que se desvían de la norma. Por ejemplo, es posible medir componentes para descubrir aquéllos con la complejidad más alta. Éstos tienen más probabilidad de tener bugs porque la complejidad los hace más difíciles de entender.

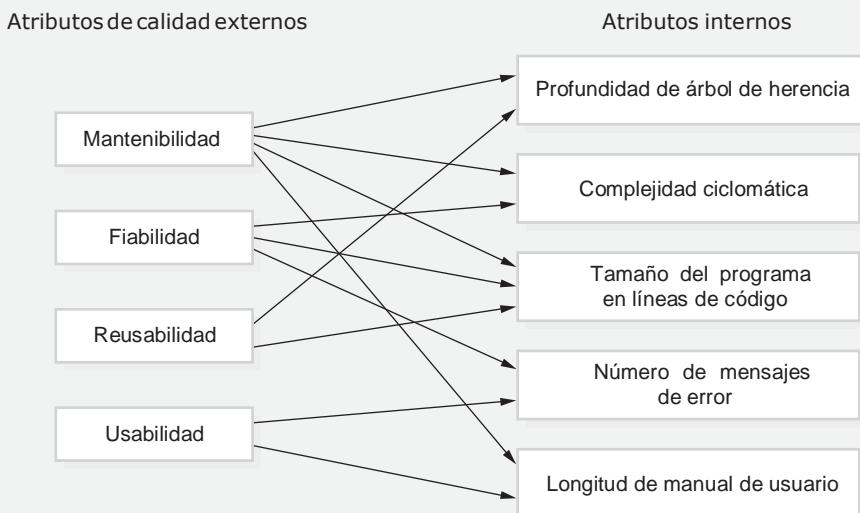
Lamentablemente, es difícil hacer mediciones directas de muchos de los atributos de calidad del software que se muestran en la figura 24.2. Los atributos de calidad, como mantenibilidad, comprensibilidad y usabilidad, son atributos externos que se refieren a cómo los desarrolladores y usuarios experimentan el software. Se ven afectados por factores subjetivos, como la experiencia y educación del usuario, y, por lo tanto, no pueden medirse de manera objetiva. Para hacer un juicio sobre estos atributos, hay que medir algunos atributos internos del software (como tamaño, complejidad, etcétera) y suponer que éstos se relacionan con las características de calidad por las que uno se interesa.

La figura 24.10 muestra algunos atributos externos de calidad del software y atributos internos que podrían, intuitivamente, relacionarse con ellos. Aunque el diagrama



sugiere que pueden existir relaciones entre atributos externos e internos, no dice cómo se

Figura 24.10  
Relaciones entre software interno y externo



relacionan dichos atributos. Si la medida del atributo interno debe ser un factor de predicción útil de la característica externa del software, deben sostenerse tres condiciones (Kitchenham, 1990):

1. El atributo interno debe medirse con exactitud. Esto no siempre es un proceso directo y tal vez se requiera de herramientas de propósito especial para hacer las mediciones.
2. Debe existir una relación entre el atributo que pueda medirse y el atributo de calidad externo que es de interés. Esto es, el valor del atributo de calidad debe relacionarse, en alguna forma, con el valor del atributo que puede medirse.
3. Esta relación entre los atributos interno y externo debe comprenderse, validarse y expresarse en términos de una fórmula o un modelo. La formulación de un modelo implica identificar la manera funcional del modelo (lineal, exponencial, etcétera) mediante el análisis de datos recopilados, identificar los parámetros que se incluirán en el modelo, y calibrar dichos parámetros usando los datos existentes.

Los atributos de software internos, como la complejidad ciclomática de un componente, se miden usando herramientas de software que analizan el código fuente del software. Hay herramientas disponibles de fuente abierta que pueden utilizarse para hacer dichas mediciones. Aunque la intuición sugiere que podría existir una relación entre la complejidad de un componente de software y el número de fallas observadas en el uso, es difícil demostrar objetivamente que éste es el caso. Para probar esta hipótesis, se requieren datos de falla para un gran número de componentes y acceso al código fuente del componente para su análisis. Muy pocas compañías han establecido un compromiso a largo plazo para la recopilación de datos sobre su software, de manera que pocas veces están disponibles datos de fallas para el análisis.

En la década de 1990, numerosas y grandes compañías, como Hewlett-Packard (Grady,



1993), AT&T (Barnard y Price, 1994) y Nokia (Kilpi, 2001) introdujeron programas de métricas. Hicieron mediciones de sus productos y procesos y las usaron durante sus

procesos de gestión de calidad. La mayor parte de la atención se centró en la recolección de métricas sobre los defectos de programa y los procesos de verificación y validación. Offen y Jeffrey (1997) y Hall y Fenton (1997) tratan con más detalle la introducción en la industria de programas de métricas.

Existe escasa información disponible al público concerniente al uso actual en la industria de la medición sistemática del software. Muchas compañías reúnen información referente a su software, como el número de peticiones de cambio de requerimientos o el número de defectos descubiertos en las pruebas. Sin embargo, no es claro si usan entonces dichas mediciones de manera sistemática para comparar productos y procesos de software o para valorar el efecto de los cambios sobre los procesos y las herramientas de software. Existen algunas razones por las que esto se dificulta:

1. Es imposible cuantificar la rentabilidad de la inversión de introducir un programa de métricas organizacional. En años pasados existieron significativas mejoras en la calidad del software sin el uso de métricas, así que es difícil justificar los costos iniciales de introducir medición y valoración sistemáticas del software.
2. No hay estándares para las métricas de software o para los procesos estandarizados para medición y análisis. Muchas compañías son renuentes a introducir programas de medición hasta que se hallan disponibles tales estándares y herramientas de apoyo.
3. En gran parte de las compañías, los procesos de software no están estandarizados y se encuentran mal definidos y controlados. Por lo tanto, hay demasiada variabilidad de procesos dentro de la misma compañía para que las mediciones se usen en una forma significativa.
4. Buena parte de la investigación en la medición y métricas del software se enfoca en métricas basadas en códigos y procesos de desarrollo basados en un plan. Sin embargo, ahora cada vez más se desarrolla software mediante la configuración de sistemas ERP o COTS, o el uso de métodos ágiles. Por consiguiente, no se sabe si la investigación previa es aplicable a dichas técnicas de desarrollo de software.
5. La introducción de medición representa una carga adicional a los procesos. Esto contradice las metas de los métodos ágiles, los cuales recomiendan la eliminación de actividades de proceso que no están directamente relacionadas con el desarrollo de programas. En consecuencia, es improbable que las compañías que adoptaron los métodos ágiles aprueben un programa de métricas.

La medición y las métricas de software son la base de la ingeniería de software empírica (Endres y Rombach, 2003). Ésta es un área de investigación en la que se han usado experimentos respecto a los sistemas de software, y la recolección de datos referente a proyectos reales para formar y validar hipótesis sobre métodos y técnicas de ingeniería de software. Los investigadores que trabajan en esta área argumentan que sólo es posible confiar en el valor de los métodos y las técnicas de la ingeniería de software si se encuentra evidencia concreta de que en realidad ofrecen los beneficios que sugieren sus inventores.



Resulta lamentable que aun cuando es posible hacer mediciones objetivas y extraer conclusiones a partir de ellas, esto no necesariamente convence a quienes toman las decisiones. En vez de ello, la toma de decisiones está influida con frecuencia por factores

subjetivos, como la novedad, o la medida en que las técnicas son de interés para los profesionales. Por lo tanto, se considera que transcurrirán muchos años antes de que los resultados de la ingeniería de software empírica presenten un efecto significativo sobre la práctica de la ingeniería de software.

## Métricas del producto

Las métricas del producto son métricas de predicción usadas para medir los atributos internos de un sistema de software. Los ejemplos de las métricas de productos incluyen el tamaño del sistema, la medida en líneas de código o el número de métodos asociados con cada clase de objeto. Por desgracia, como se explicó anteriormente en esta sección, las características del software que pueden medirse fácilmente, como el tamaño y la complejidad ciclomática, no tienen una relación clara y consistente con los atributos de calidad como comprensibilidad y mantenibilidad. Las relaciones varían dependiendo de los procesos de desarrollo, la tecnología empleada y el tipo de sistema a diseñar.

Las métricas del producto se dividen en dos clases:

1. Métricas dinámicas, que se recopilan mediante mediciones hechas de un programa en ejecución. Dichas métricas pueden recopilarse durante las pruebas del sistema o después de que el sistema está en uso. Un ejemplo es el número de reportes de bugs o el tiempo necesario para completar un cálculo.
2. Métricas estáticas, las cuales se recopilan mediante mediciones hechas de representaciones del sistema, como el diseño, el programa o la documentación. Ejemplos de mediciones estáticas son el tamaño del código y la longitud promedio de los identificadores que se usaron.

Estos tipos de métrica se relacionan con diferentes atributos de calidad. Las métricas dinámicas ayudan a valorar la eficiencia y fiabilidad de un programa. Las métricas estáticas ayudan a valorar la complejidad, comprensibilidad y mantenibilidad de un sistema de software o de los componentes del sistema.

Por lo general, existe una relación clara entre métricas dinámicas y características de calidad del software. Es muy sencillo medir el tiempo de ejecución requerido para funciones particulares y valorar el tiempo requerido con la finalidad de iniciar un sistema. Éstos se relacionan directamente con la eficiencia del sistema. De igual modo, el número de fallas del sistema y el tipo de fallas pueden registrarse y relacionarse directamente con la fiabilidad del software, que se estudió en el capítulo 15.

Como se comentó, las métricas estáticas, como las que se muestran en la figura 24.11, tienen una relación indirecta con los atributos de calidad. Se ha propuesto una gran cantidad de diferentes métricas y se han intentado muchos experimentos para derivar y validar las relaciones entre dichas métricas y atributos como complejidad y mantenibilidad. Ninguno de tales experimentos ha sido concluyente, pero el tamaño del programa y la complejidad del control parecen ser los factores de predicción más fiables de la comprensibilidad, la complejidad del sistema y la mantenibilidad.

Las métricas de la figura 24.11 son aplicables a cualquier programa, pero también se han propuesto métricas más específicas orientadas a objetos (OO). La figura 24.12 resume la suite de Chidamber y Kemerer (en ocasiones llamada suite CK) de seis métricas.



Métrica de software	Descripción
Fan-in/Fan-out	Fan-in (abanicado de entrada) es una medida del número de funciones o métodos que llaman a otra función o método (por ejemplo, X). Fan-out (abanicado de salida) es el número de funciones a las que llama la función X. Un valor alto para fan-in significa que X está estrechamente acoplado con el resto del diseño y que los cambios a X tendrán extensos efectos dominó. Un valor alto de fan-out sugiere que la complejidad global de X puede ser alta debido a la complejidad de la lógica de control necesaria para coordinar los componentes llamados.
Longitud de código	Ésta es una medida del tamaño de un programa. Por lo general, cuanto más grande sea el tamaño del código de un componente, más probable será que el componente sea complejo y proclive a errores. Se ha demostrado que la longitud del código es una de las métricas más fiables para predecir la proclividad al error en los componentes.
Complejidad ciclomática	Ésta es una medida de la complejidad del control de un programa. Tal complejidad del control puede relacionarse con la comprensibilidad del programa. En el capítulo 8 se estudia la complejidad ciclomática.
Longitud de identificadores	Ésta es una medida de la longitud promedio de los identificadores (nombres para variables, clases, métodos, etcétera) en un programa. Cuanto más largos sean los identificadores, es más probable que sean significativos y, por ende, más comprensible será el programa.
Profundidad de anidado condicional	Ésta es una medida de la profundidad de anidado de los enunciados if en un programa. Los enunciados if profundamente anidados son difíciles de entender y proclives potencialmente a errores.
Índice Fog	Ésta es una medida de la longitud promedio de las palabras y oraciones en los documentos. Cuanto más alto sea el valor del índice Fog de un documento, más difícil será entender el documento.

Figura 24.11  
Métricas estáticas  
de productos de  
software

orientadas a objetos (1994). Aunque se propusieron originalmente a principio de la década de 1990, aún son las métricas OO de más amplio uso. Algunas herramientas de diseño UML recopilan automáticamente valores para dichas métricas conforme se crean los diagramas UML.

El-Amam (2001) hace una excelente revisión de las métricas orientadas a objetos, analiza las métricas CK y otras métricas OO, y concluye que todavía no se tiene suficiente evidencia para comprender cómo estas y otras métricas orientadas a objetos se relacionan con cualidades externas de software. Esta situación no ha cambiado realmente desde su análisis en 2001. Todavía no se sabe cómo usar las mediciones de los programas orientados a objetos para extraer conclusiones fiables acerca de su calidad.

### Análisis de componentes de software

En la figura 24.13 se ilustra un proceso de medición que puede ser parte de un proceso de valoración de calidad del software. Cada componente del sistema puede analizarse por separado mediante un rango de métricas. Los valores de dichas métricas pueden compararse entonces para diferentes componentes y, tal vez, con datos de medición históricos.

### Métrica orientada a objetos Descripción

Métodos ponderados por clase ( <i>weighted methods per class</i> , WMC)	Éste es el número de métodos en cada clase, ponderado por la complejidad de cada método. Por lo tanto, un método simple puede tener una complejidad de 1, y un método grande y complejo tendrá un valor mucho mayor. Cuanto más grande sea el valor para esta métrica, más compleja será la clase de objeto. Es más probable que los objetos complejos sean más difíciles de entender. Tal vez no sean lógicamente cohesivos, por lo que no pueden reutilizarse de manera efectiva como superclases en un árbol de herencia.
Profundidad de árbol de herencia ( <i>depth of inheritance tree</i> , DIT)	Esto representa el número de niveles discretos en el árbol de herencia en que las subclases heredan atributos y operaciones (métodos) de las superclases. Cuanto más profundo sea el árbol de herencia, más complejo será el diseño. Es posible que tengan que comprenderse muchas clases de objetos para entender las clases de objetos en las hojas del árbol.
Número de hijos ( <i>number of children</i> , NOC)	Ésta es una medida del número de subclases inmediatas en una clase. Mide la amplitud de una jerarquía de clase, mientras que DIT mide su profundidad. Un valor alto de NOC puede indicar mayor reutilización. Podría significar que debe realizarse más esfuerzo para validar las clases base, debido al número de subclases que dependen de ellas.
Acoplamiento entre clases de objetos ( <i>coupling between object classes</i> , CBO)	Las clases están acopladas cuando los métodos en una clase usan los métodos o variables de instancia definidos en una clase diferente. CBO es una medida de cuánto acoplamiento existe. Un valor alto para CBO significa que las clases son estrechamente dependientes y, por lo tanto, es más probable que el hecho de cambiar una clase afecte a otras clases en el programa.
Respuesta por clase ( <i>response for a class</i> , RFC)	RFC es una medida del número de métodos que potencialmente podrían ejecutarse en respuesta a un mensaje recibido por un objeto de dicha clase. Nuevamente, RFC se relaciona con la complejidad. Cuanto más alto sea el valor para RFC, más compleja será una clase y, por ende, es más probable que incluya errores.
Falta de cohesión en métodos ( <i>lack of cohesion in methods</i> , LCOM)	LCOM se calcula al considerar pares de métodos en una clase. LCOM es la diferencia entre el número de pares de método sin compartir atributos y el número de pares de método con atributos compartidos. El valor de esta métrica se debate ampliamente y existe en muchas variaciones. No es claro si realmente agrega alguna información útil además de la proporcionada por otras métricas.

Figura 24.12 Suite de métricas CK orientadas a objetos

recopilados en proyectos anteriores. Las mediciones anómalas, que se desvían significativamente de la norma, pueden implicar que existen problemas con la calidad de dichos componentes.

Las etapas clave en este proceso de medición de componentes son:

1. *Elegir las mediciones a realizar* Deben formularse las preguntas que la medición busca responder, y definir las mediciones requeridas para responder a tales preguntas. Deben recopilarse las mediciones que no son directamente relevantes para dichas preguntas. El paradigma GQM (por las siglas de *Goal-Question-Metric*, es decir, Meta-Pregunta-Métrica)



de  
Basili  
(Basili  
y  
Romb  
ach,  
1988),  
que  
se  
estudi  
a

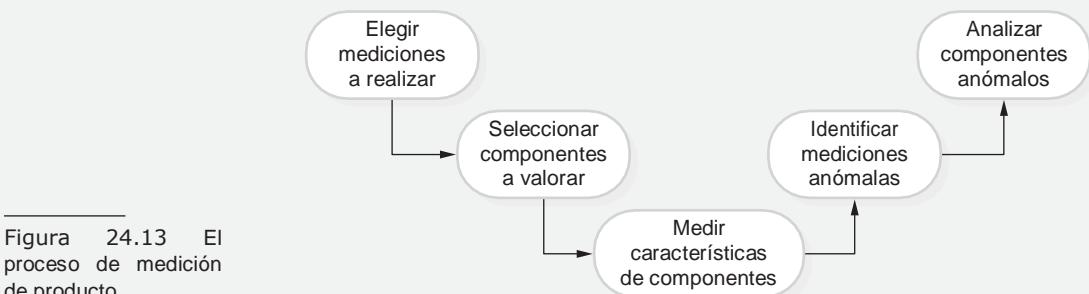


Figura 24.13 El proceso de medición de producto

en el capítulo 26, es un enfoque adecuado cuando se decide cuáles datos hay que recopilar.

2. *Seleccionar componentes a valorar* Probablemente usted no necesite estimar valores métricos para todos los componentes en un sistema de software, ya que en ocasiones podrá seleccionar una muestra representativa de componentes para medición, que le permitirá realizar una valoración global de la calidad del sistema. En otras circunstancias, tal vez desee enfocarse en los componentes centrales del sistema que están casi en uso constante. La calidad de dichos componentes es más importante que la de aquellos componentes que sólo se usan muy pocas veces.
3. *Medir las características de los componentes* Se miden los componentes seleccionados y se calculan los valores de métrica asociados. Por lo general, esto implica procesar la representación de los componentes (diseño, código, etcétera) mediante una herramienta de recolección automatizada de datos. Esta herramienta puede escribirse especialmente o ser una característica de las herramientas de diseño que ya están en uso.
4. *Identificar mediciones anómalas* Despues de hacer las mediciones de componentes, se comparan entonces unas con otras y con mediciones anteriores que se hayan registrado en una base de datos de mediciones. Hay que observar los valores inusualmente altos o bajos para cada métrica, pues éstos sugieren que podría haber problemas con el componente que muestra dichos valores.
5. *Analizar componentes anómalos* Cuando identifique los componentes con valores anómalos para sus métricas seleccionadas, debe examinarlos para decidir si dichos valores de métrica anómalo significan que la calidad del componente se encuentra o no comprometida. Un valor de métrica anómalo para la complejidad (al parecer) no necesariamente significa un componente de mala calidad. Podría haber alguna otra razón para el valor alto, por lo que no necesariamente significa que haya problemas con la calidad del componente.



organizacional, así como registros históricos de todos los proyectos aun cuando no se hayan usado durante un proyecto particular. Una vez establecida una base suficientemente grande de datos de medición, será posible hacer comparaciones de calidad de software a través de proyectos, además de validar las relaciones entre atributos de componentes internos y características de calidad.

## Ambigüedad de mediciones

Cuando reúna datos cuantitativos relativos al software y los procesos de software, deberá analizar dichos datos para entender su significado. Es fácil malinterpretar los datos y hacer inferencias incorrectas. No basta con observar los datos por sí mismos, sino que también hay que considerar el contexto donde se recaban los datos.

Para ilustrar cómo pueden interpretarse los datos recopilados en diferentes formas, considere el siguiente escenario, que se ocupa del número de peticiones de cambio hechas por los usuarios de un sistema:

*Una administradora decide monitorizar el número de peticiones de cambio enviadas por los clientes, con base en una suposición de que existe una relación entre dichas peticiones de cambio y la usabilidad y conveniencia del producto. Ella supone que cuanto más alto sea el número de peticiones de cambio, menos cumple el software las necesidades del cliente.*

*Es costoso manejar las peticiones de cambio y modificar el software. Por lo tanto, la organización decide cambiar su proceso con la intención de mejorar la satisfacción del cliente y, al mismo tiempo, reducir los costos de hacer cambios. La intención es que el cambio de proceso dará como resultado mejores productos y menos peticiones de cambio.*

*Los cambios de proceso se inician para aumentar la inclusión del cliente en el proceso de diseño del software. Se introducen pruebas beta de todos los productos; además, se incorporan en el producto entregado las modificaciones solicitadas por el cliente. Se entregan las nuevas versiones de los productos, que se desarrollan mediante este proceso modificado. En algunos casos se reduce el número de peticiones de cambio, aunque en otros aumenta. La administradora está confundida y descubre que es imposible valorar los efectos de los cambios de proceso sobre la calidad del producto.*

Para comprender por qué puede ocurrir este tipo de ambigüedad, hay que conocer las razones por las que los usuarios pueden hacer peticiones de cambio:

1. El software no es lo bastante bueno y no hace lo que quieren los clientes. Por lo tanto, solicitan cambios para obtener la funcionalidad que ellos requieren.
2. Como alternativa, el software puede ser muy bueno y, por consiguiente, se usa amplia e intensamente. Las peticiones de cambio pueden generarse porque existen muchos usuarios de software que piensan creativamente en nuevas ideas que podrían hacer con el software.

Por ende, aumentar la participación del cliente en el proceso puede reducir el número de peticiones de cambio para los productos con los que los clientes están descontentos. Los cambios de proceso han sido efectivos y han hecho al software más útil y adecuado. Sin embargo, alternativamente, los cambios al proceso pueden no

**EDITORIAL:**

**guiaceneval.mx**



haber funcionado, y los clientes tal vez decidieron buscar un sistema opcional. El número de peticiones de cambio disminuye porque el producto perdió participación en el mercado frente a un producto rival y, en consecuencia, hay menos usuarios del producto.

Por otra parte, los cambios al proceso pueden conducir a muchos nuevos clientes satisfechos que deseen participar en el proceso de desarrollo del producto. Por lo tanto, generan más peticiones de cambio. Los cambios al proceso de manejar las peticiones de cambio contribuyen a este aumento. Si la compañía tiene mayor capacidad de respuesta con los clientes, ellos generarán más peticiones de cambio porque saben que éstas se tomarán con seriedad. Creen que sus sugerencias se incorporarán quizás en versiones posteriores del software. O bien, el número de peticiones de cambio puede aumentar porque los sitios de prueba beta no eran los típicos del mayor uso del programa.

Para analizar los datos de petición de cambio, no basta con conocer el número de peticiones de cambio, sino que se necesita conocer quién hizo la petición, cómo usa el software y por qué hizo la petición. También se requiere información sobre los factores externos, como modificaciones al procedimiento de petición de cambio o cambios al mercado que puedan tener un efecto. Con esta información, es posible averiguar si los cambios al proceso fueron efectivos para aumentar la calidad del producto.

Esto ilustra las dificultades de entender los efectos de los cambios, y el enfoque “científico” a este problema es reducir el número de factores que tiendan a afectar las mediciones hechas. Sin embargo, los procesos y productos que se miden no están aislados de su entorno. El ambiente empresarial cambia constantemente y es imposible evitar los cambios a la práctica laboral sólo porque pueden hacerse comparaciones de datos inválidos. Como tales, los datos cuantitativos sobre las actividades humanas no siempre deben tomarse en serio. Las razones por las que cambia un valor medido con frecuencia son ambiguas. Dichas razones deben investigarse a profundidad antes de extraer conclusiones de cualquier medición que se haya realizado.

## PUNTOS CLAVE



- La gestión de calidad del software se ocupa de garantizar que el software tenga un número menor de defectos y que alcance los estándares requeridos de mantenibilidad, fiabilidad, portabilidad, etcétera. Incluye definir estándares para procesos y productos, y establecer procesos para comprobar que se siguieron dichos estándares.
- Los estándares de software son importantes para el aseguramiento de la calidad, pues representan una identificación de las “mejores prácticas”. Al desarrollar el software, los estándares proporcionan un cimiento sólidopara diseñar software de buena calidad.
- Es necesario documentar un conjunto de procedimientos de aseguramiento de la calidad en un manual de calidad organizacional. Esto puede basarse en el modelo genérico para un manual de calidad sugerido en el estándar ISO 9001.
- Las revisiones de los entregables del proceso de software incluyen a un equipo de personas que verifican que se siguieron los estándares de calidad. Las revisiones son la técnica usada más ampliamente para valorar la calidad.
- En una inspección de programa o revisión de pares, un reducido equipo comprueba sistemáticamente el código. Ellos leen el código a detalle y buscan posibles errores y omisiones. Entonces los problemas detectados se discuten en una reunión de revisión del código.

- La medición del software puede usarse para separar y copiar datos cuantitativos de software como el modelo de procesos de software. Se usan los valores de las métricas de software recopilados para hacer inferencias referentes a la calidad del producto y el proceso.
- Las métricas de calidad del producto son particularmente útiles para resaltar los componentes anómalos que pudieran tener problemas de calidad. Dichos componentes deben entonces analizarse con más detalle.

## LECTURAS SUGERIDAS

*Metrics and Models for Software Quality Engineering, 2nd edition.* Éste es un análisis muy completo de las métricas del software que incluyen métricas de proceso, de producto y orientadas a objetos. También contiene cierto conocimiento matemático requerido para desarrollar y comprender modelos basados en medición de software. (S. H. Kan, Addison-Wesley, 2003.)

*Software Quality Assurance: From Theory to Implementation.* Una excelente vista actualizada a los principios y la práctica de la aseguramiento de la calidad del software. Incluye un análisis de los estándares, como el ISO 9001. (D. Galin, Addison-Wesley, 2004.)

“A Practical Approach for Quality-Driven Inspections”. En la actualidad muchos artículos concernientes a las inspecciones son más bien anticuados, ya que no consideran la práctica moderna del desarrollo de software. Este texto relativamente reciente describe un método de inspección que se ocupa de algunos de los problemas al utilizarla en inspección y sugiere cómo pueden usarse las inspecciones en un entorno moderno de desarrollo. (C. Denger, F. Shull, *IEEE Software*, 24(2), marzo-abril de 2007.)

<http://dx.doi.org/10.1109/MS.2007.31>

“Misleading Metrics and Unsound Analyses”. Un excelente artículo de los principales investigadores de métricas, quienes analizan las dificultades de comprensión de lo que significan realmente las métricas. (B. Kitchenham, R. Jeffrey y C. Connaughton, *IEEE Software*, 24(2), marzo- abril de 2007.)

<http://dx.doi.org/10.1109/MS.2007.49>.

“The Case for Quantitative Project Management”. Ésta es una introducción a una sección especial de la revista que incluye otros dos artículos sobre administración cuantitativa de proyectos.

Plantea razones para una mayor investigación en métricas y medición con la finalidad de mejorar la administración de proyectos de software. (B. Curtis et al., *IEEE Software*, 25 (3), mayo-junio de 2008.)

<http://dx.doi.org/10.1109/MS.2008.80>.

## EJERCICIOS



Explique por qué un proceso de software de alta calidad debe ser conducir a productos de alta calidad de software. Discuta los posibles problemas con este sistema de gestión de calidad.

Exponga cómo pueden usarse los estándares para obtener conocimiento de la organización sobre los métodos efectivos de desarrollo de software. Sugiera cuatro tipos de conocimiento que puedan reflejarse en los estándares de la organización.

Discuta la valoración de calidad del software según los atributos de calidad mostrados en la figura 24.2. Debe considerar a la vez cada atributo y explicar cómo puede valorarse.

Diseñe un formato electrónico que pueda usar para registrar comentarios de revisión para enviar comentarios por correo electrónico a los revisores.

Describa brevemente posibles estándares que podría utilizar para:

- El uso de sentencias de control en C, C# o Java;
- enviar reportes para un proyecto final en una universidad;
- el proceso de hacer y aprobar cambios al programa (véase el capítulo 26);
- el proceso de comprar e instalar una nueva computadora.

Suponga que trabaja en una organización que desarrolla productos de bases de datos para individuos y empresas pequeñas. Esta organización está interesada en cuantificar su desarrollo de software. Escriba un reporte que sugiera métricas adecuadas y mencione cómo pueden recopilarse.

Explique por qué las inspecciones de programas son una técnica efectiva para descubrir errores en un programa. ¿Qué tipos de errores tienen en cuenta la probabilidad de descubrirse mediante inspecciones?

Diga por qué las métricas de diseño son, por sí mismas, un método inadecuado para predecir la calidad del diseño.

Explique por qué es difícil validar las relaciones entre atributos de producto internos (como la complejidad ciclomática) y los atributos externos (como la mantenibilidad).

Un colega que es muy buen programador labora software con un bajo número de defectos, pero siempre pasa por alto los estándares de calidad de la organización. ¿Cómo deberían reaccionar sus administradores ante este comportamiento?

## REFERENCIAS

Bamford, R. y Deibler, W. J. (eds.) (2003). "ISO9001:2000 for Software and Systems Providers: An Engineering Approach". Boca Raton, Fla.: CRC Press.

Barnard, J. y Price, A. (1994). "Managing Code Inspection Information". *IEEE Software*, 11(2), 59–69.

Basili, V. R. y Rombach, H. D. (1988). "The TAME project: Towards Improvement-Oriented Software Environments". *IEEE Trans. on Software Eng.*, 14(6), 758–773.

Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., Macleod, G. y Merrit, M. (1978). *Characteristics of Software Quality*. Amsterdam: North-Holland.

Chidamber, S. y Kemerer, C. (1994). "A Metrics Suite for Object-Oriented Design". *IEEE Trans. on Software Eng.*, 20(6), 476–93.



- Crosby, P. (1979). *Quality is Free*. Nueva York: McGraw-Hill.
- El-Amam, K. 2001. "Object-oriented Metrics: A Review of Theory and Practice". National Research Council of Canada. <http://seg.iit.nrc.ca/English/abstracts/NRC44190.html>.
- Endres, A. y Rombach, D. (2003). *Empirical Software Engineering: A Handbook of Observations, Laws and Theories*. Harlow, UK: Addison-Wesley.
- Fagan, M. E. (1976). "Design and code inspections to reduce errors in program development". *IBM Systems J.*, **15** (3), 182–211.
- Fagan, M. E. (1986). "Advances in Software Inspections". *IEEE Trans. on Software Eng.*, **SE-12** (7), 744–51.
- Gilb, T. y Graham, D. (1993). *Software Inspection*. Wokingham: Addison-Wesley.
- Grady, R. B. (1993). "Practical Results from Measuring Software Quality". *Comm. ACM*, **36** (11), 62–8. Gunning, R. (1962). *Techniques of Clear Writing*. Nueva York: McGraw-Hill.
- Hall, T. y Fenton, N. (1997). "Implementing Effective Software Metrics Programs". *IEEE Software*, **14** (2), 55–64.
- Humphrey, W. (1989). *Managing the Software Process*. Reading, Mass.: Addison-Wesley.
- IEC. 1998. "Standard IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems". International Electrotechnical Commission: Ginebra.
- IEEE. (2003). *IEEE Software Engineering Standards Collection on CD-ROM*. Los Alamitos, Ca.: IEEE Computer Society Press.
- Ince, D. (1994). *ISO 9001 and Software Quality Assurance*. Londres: McGraw-Hill.
- Kilpi, T. (2001). "Implementing a Software Metrics Program at Nokia". *IEEE Software*, **18** (6), 72–7. Kitchenham, B. (1990). "Measuring Software Development". En *Software Reliability Handbook*. Rook, P. (ed.). Amsterdam: Elsevier, 303–31.
- McConnell, S. (2004). *Code Complete: A Practical Handbook of Software Construction, 2nd edition*. Seattle: Microsoft Press.
- Mills, H. D., Dyer, M. y Linger, R. (1987). "Cleanroom Software Engineering". *IEEE Software*, **4** (5), 19–25.
- Offen, R. J. y Jeffrey, R. (1997). "Establishing Software Measurement Programs". *IEEE Software*, **14** (2), 45–54.
- Stalhane, T. y Hanssen, G. K. (2008). "The application of ISO 9001 to agile software development". *9th International Conference on Product Focused Software Process Improvement, PROFES 2008*, Monte Porzio Catone, Italy: Springer.

## Lectura 9. Arquitectura de redes de información



## Arquitectura de redes de información. Principios y

**conceptos *Architecture of information networks.***

***Principles and concepts Arquitetura de redes de***

***informação. Princípios e Conceitos***

Verónica P. Tintín-Perdomo<sup>1</sup>

[vptintin@espe.edu.ec](mailto:vptintin@espe.edu.ec)

José R. Caiza-Caizabuano<sup>2</sup>

[jrcaiza@espe.edu.ec](mailto:jrcaiza@espe.edu.ec)

Fernando S. Caicedo-Altamirano<sup>3</sup>

[fscaicedo@espe.edu.ec](mailto:fscaicedo@espe.edu.ec)

**Recibido:** 15 de septiembre de 2017 \* **Corregido:** 20 de noviembre de 2017 \* **Aceptado:** 22 de febrero de 2018

<sup>1</sup> Ingeniero en Sistemas, Supervisora del Área de Digitalización del Ministerio Coordinador de Desarrollo Social, Docente de la Unidad de Gestión de Tecnologías de la Universidad de las Fuerzas Armadas ESPE.

<sup>2</sup> Ingeniero en Sistemas e Informática, Docente de la Unidad de Gestión de Tecnologías de la Universidad de las Fuerzas Armadas ESPE.

<sup>3</sup> Ingeniero en Electrónica y Comunicaciones, Docente de la Unidad de Gestión de Tecnologías de la Universidad de las Fuerzas Armadas ESPE.



## **Resumen**

Los grandes avances que se han dado en los últimos años en los campos de tecnologías de la información y las comunicaciones, conducen hacia un mundo cada vez más global e interconectado. La llamada sociedad de la información ha traído consigo un enorme incremento en el número de interacciones entre individuos, así como entre individuos y empresas. La infraestructura que hace posible estas interacciones, son las redes informáticas y su arquitectura. La presente investigación tiene como objetivo estudiar la arquitectura de redes, sus principios y conceptos, con la finalidad de develar que en los entornos de trabajo académico, empresarial, comercial o de cualquier otra índole, el trabajo en red minimiza las redundancias, reduce el tiempo de dedicación y esfuerzo; además de facilitar tanto la difusión como el intercambio de información. La metodología de trabajo fue de tipo bibliográfico, permitiendo una panorámica general de la arquitectura de las redes informáticas, enfocada en los conceptos básicos de los sistemas informáticos, tipos de arquitectura de red informática y tipos de redes. Es de resaltar que en el conocimiento del modelo OSI, en cada una de sus capas, aun cuando tiene dos únicas capas con las cuales interactúa el usuario la primera o capa física y la última o capa de aplicación, existen otras cinco capas donde cada una tiene una función determinada dentro del proceso global de la transmisión de datos entre equipos informáticos interconectados a través de protocolos.

**Palabras clave:** tecnologías de comunicación e información; arquitectura de redes informáticas; principios; conceptos.

## **Abstract**

The great advances that have occurred in recent years in the fields of information technology and communications, lead to an increasingly global and interconnected world. The so-called information society has brought with it a huge increase in the number of interactions between individuals, as well as between individuals and companies. The infrastructure that makes these interactions possible is the computer networks and their architecture. The objective of this research is to study the architecture of networks, their principles and concepts, with the purpose of revealing that in academic, business, commercial or any other work environment, networking minimizes redundancies, reduces time of

dedication and effort; in addition to facilitating both the dissemination and the exchange of information. The methodology of work was of bibliographic type, allowing a general panoramic of the architecture of the computer networks, focused on the basic concepts of computer systems, types of computer network architecture and types of networks. It is noteworthy that in the knowledge of the OSI model, in each of its layers, even though it has two unique layers with which the user interacts with the first or physical layer and the last layer or application layer, there are five other layers where each one it has a determined function within the global process of the transmission of data between computer equipment interconnected through protocols.

**Keywords:** communication and information technologies; computer network architecture; beginning; concepts.

## Resumo

Os grandes avanços que ocorreram nos últimos anos nas áreas de tecnologia da informação e comunicações, levam a um mundo cada vez mais global e interconectado. A chamada sociedade da informação trouxe consigo um enorme aumento no número de interações entre indivíduos, assim como entre indivíduos e empresas. A infraestrutura que possibilita essas interações é a rede de computadores e sua arquitetura. O objetivo desta pesquisa é estudar a arquitetura de redes, seus princípios e conceitos, com o objetivo de revelar que, em ambientes acadêmicos, empresariais, comerciais ou qualquer outro ambiente de trabalho, a rede minimiza redundâncias, reduz o tempo de dedicação e esforço; além de facilitar tanto a divulgação quanto o intercâmbio de informações. A metodologia de trabalho foi do tipo bibliográfica, permitindo uma panorâmica geral da arquitetura das redes de computadores, focada nos conceitos básicos de sistemas computacionais, tipos de arquitetura de redes de computadores e tipos de redes. Vale ressaltar que, no conhecimento do modelo OSI, em cada uma de suas camadas, embora tenha duas camadas únicas com as quais o usuário interage com a primeira ou camada física e a última camada ou camada de aplicação, existem outras cinco camadas onde cada uma tem uma função determinada dentro do processo global de transmissão de dados entre equipamentos de informática interconectados através de protocolos.



**Palavras chave:** tecnologias de comunicação e informação; arquitetura de redes de computadores; princípios; conceitos.

## Introducción

La humanidad, desde épocas prehistóricas, ha evidenciado su gran necesidad de comunicarse, de transmitir información e ideas, con los recursos materiales y tecnológicos con los que ha contado en cada época. Desde las pinturas rupestres y la escritura cuneiforme hasta los modernos y sofisticados sistemas de información y comunicación actuales, la humanidad ha avanzado en su propósito de estar en comunicación con su entorno personal, familiar, laboral, de negocios o simplemente de ocio.

Los grandes avances que se han dado en los últimos años en los campos de tecnologías de información y de comunicaciones (TIC), conducen hacia un mundo cada vez más global e interconectado. La llamada sociedad de la información ha traído consigo un enorme incremento en el número de interacciones entre individuos, así como entre individuos y empresas. La infraestructura que hace posible estas interacciones, son las redes informáticas y su arquitectura (Incera y col, 2007).

Las redes informáticas pueden definirse como un conjunto de equipos (computadoras, periféricos, etc.) que están interconectados y que comparten diversos recursos (Mancilla, 2018). Este tipo de redes implica la interconexión de los equipos a través de ciertos dispositivos que permiten el envío y la recepción de ondas, las cuales llevan los datos que se desea compartir. En las redes informáticas, por lo tanto, hay emisores y receptores que intercambian mensajes.

Por su parte, la arquitectura de red representa la conceptualización o visualización de cómo deben diseñarse y funcionar las redes informáticas en relación con sus propósitos y medios tecnológicos disponibles.

En este trabajo de investigación bibliográfica se presenta una panorámica general de la arquitectura las redes informáticas, se enfoca en los conceptos básicos de sistemas informáticos.

## Desarrollo

Según Rivera (2016), una red es la interconexión entre dos o más equipos que se comunican entre sí para transmitir o recibir información en distintos instantes de tiempo además de compartir recursos. Dicha alternancia de tiempo se establece a través de mensajes (comando o señales) que determinan un intercambio de roles, bien sea para emitir o para recibir tales mensajes.

Esto, se materializa de un equipo a otro a través de un dispositivo denominado Tarjeta para Interfaz de Red del inglés Network Interface Card (NIC) que maneja un protocolo de comunicación para la emisión/recepción de datos, el cual se denomina nodo de comunicación. A su vez, se comunica a través de un medio de conexión que puede ser cableado o inalámbrico y otro dispositivo que se encarga de conmutar, similar a una central telefónica, las recepciones y emisiones de comunicación llamado switch o comutador.

Para Peña (2013), los componentes de una red, de manera general los dispositivos conectados a una red informática pueden clasificarse en dos tipos: los que gestionan el acceso y las comunicaciones en una red (dispositivos de red), como módem, router, switch, access point, bridge, entre otros; y los que se conectan para utilizarla (dispositivos de usuario final), como computadora, notebook, tablet, teléfono celular, impresora, televisor inteligente, consola de videojuegos y afines.

En cuanto a la arquitectura de red es el medio más efectivo en cuanto a costos para desarrollar e implementar un conjunto coordinado de productos que se puedan interconectar. La arquitectura es el “plan” con el que se conectan los protocolos y otros programas de software. Esto es benéfico tanto para los usuarios de la red como para los proveedores de hardware y software.

Entre las características de una arquitectura de red, se tiene:

- La separación de funciones. Partiendo de que las redes separan los usuarios y los productos ofertados que evolucionan por la naturaleza de la tecnología con el tipo, las funciones mejoradas se puedan adaptan a la última, a través de la arquitectura de red se concibe el



sistema de manera modular, para evitar perturbaciones por los cambios mínimos que mantengan una alta disponibilidad.

- Conectividad amplia. Por definición una red debe proveer una conexión óptima entre cualquier cantidad de nodos, considerando los niveles de seguridad requeridos.
- Recursos compartidos. Por medio de las arquitecturas de red puede compartirse recursos tales como impresoras, bases de datos, unidades de disco, haciendo que la operatividad de la red sea económica y eficiente.
- Administración de la red. En la arquitectura se establece la permisología correspondiente para que el usuario defina, opere, cambie, proteja y ejecute mantenimientos en dicha red.
- Facilidad de uso. Mediante la arquitectura de red el diseñador puede enfocarse en las interfaces primarias de la red, logrando hacerlas amigables al usuario.
- Administración de datos. Se considera este aspecto unido a la necesidad de interconectar los diferentes sistemas de administración de base de datos, de unidades de almacenamiento y otros servicios.
- Interfaces. Se define las interfaces de persona a red, de persona y de programa a programa. Así la arquitectura combina los protocolos apropiados de software para generar una red funcional.
- Aplicaciones. Aquí son separadas las funciones que se requieren para operar la red de acuerdo con las aplicaciones comerciales de la empresa.

### **Tipos de arquitectura de red informática**

Como parte de la evolución de las redes informáticas han surgido en el mundo científico y empresarial diversos tipos de arquitectura de redes, algunos de los cuales se describen a continuación.

- **Arquitectura SRA.** Fue desarrollada por IBM y describe una estructura integral que provee todos los modos de comunicación de datos y con base en la cual se pueden planear e implementar nuevas redes de comunicación de datos. La ASR se construyó en torno a cuatro principios básicos:
  - Comprende las funciones distribuidas con base en las cuales muchas responsabilidades de la red se pueden mover de la computadora central a otros componentes de la red como son los concentradores remotos.
  - Define trayectorias ante los usuarios finales (programas, dispositivos u operadores) de la red de comunicación de datos en forma separada de los usuarios mismos, lo cual permite hacer extensiones o modificaciones a la configuración de la red sin afectar al usuario final.
  - Se utiliza el principio de la independencia de dispositivo, lo cual permite la comunicación de un programa con un dispositivo de entrada/salida sin importar los requerimientos de cualquier dispositivo único. Esto también permite añadir o modificar programas de aplicación y equipo de comunicación sin afectar a otros elementos de la red de comunicación.
  - Se utilizan funciones y protocolos tanto lógicos como físicos normalizados para la comunicación de información entre dos puntos cualesquiera, es decir, que se puede tener una arquitectura de propósito general y terminales industriales de muchas variedades y un solo protocolo de red.

La organización lógica de una red ASR, sin importar su configuración física, se divide en dos grandes categorías de componentes: unidades direccionables de red y red de control de trayectoria.

Las unidades de direccionables de red son grupos de componentes de ASR, que proporcionan los servicios mediante los cuales el usuario final puede enviar datos a través de la red y ayudan a los operadores de la red a realizar el control de esta y las funciones de administración.

La red de control de trayectoria provee el control de enrutamiento y flujo; el principal servicio que proporciona la capa de control del enlace de datos dentro de la red de control de trayectoria es la transmisión de datos por enlaces individuales. La



red de control de trayectoria tiene dos capas: la capa de control de trayectoria y la capa de control de enlace de datos. El control de enrutamiento y de flujo

son los principales servicios proporcionados por la capa de control de trayectoria, mientras que la transmisión de datos por enlaces individuales es el principal servicio que proporciona la capa de control de enlace de datos.

Una red de comunicación de datos construida con base en los conceptos ASR consta de lo siguiente: Computadora principal, Procesador de comunicación de entrada (nodo intermedio), Controlador remoto inteligente (nodo intermedio o nodo de frontera), Diversos terminales de propósito general y orientadas a la industria (nodo terminal o nodo de grupo) y posiblemente redes de área local o enlaces de microcomputadora o macrocomputadora.

- **Arquitectura de Red Digital (DRA).** Esta es una arquitectura de red distribuida de la Digital Equipment Corporation. Se le llama DECnet y consta de cinco capas. Las capas físicas, de control de enlace de datos, de transporte y de servicios de la red corresponden casi exactamente a las cuatro capas inferiores del modelo OSI. La quinta capa, la de aplicación, es una mezcla de las capas de presentación y aplicación del modelo OSI. La DECnet no cuenta con una capa de sesión separada.

La DECnet, al igual que la ASR de IBM, define un marco general tanto para la red de comunicación de datos como para el procesamiento distribuido de datos. El objetivo de la DECnet es permitir la interconexión generalizada de diferentes computadoras principales y redes punto a punto, multipunto o conmutadas de manera tal que los usuarios puedan compartir programas, archivos de datos y dispositivos de terminal remotos.

La DECnet soporta la norma del protocolo internacional X.25 y cuenta con capacidades para conmutación de paquetes. Se ofrece un emulador mediante el cual los sistemas de la Digital Equipment Corporation se pueden interconectar con las macrocomputadoras de IBM y correr en un ambiente ASR. El protocolo de mensaje para comunicación digital de datos (PMDD) de la DECnet es un protocolo orientado a los bytes cuya estructura es similar a la del protocolo de Comunicación Binaria Síncrona (CBS) de IBM.

- **Arcnet.** La Red de computación de recursos conectadas o del inglés Attached Resource Computing Network (ARCNET) es un sistema de red banda base, con paso de testigo (token) que



ofrece topologías flexibles en estrella y bus a un precio bajo. Las velocidades de transmisión son de 2.5 Mbits/seg. ARCNET usa un protocolo de paso de testigo en una topología de red en bus con testigo, pero ARCNET en sí misma no es una norma IEEE. En 1977, Datapoint desarrolló ARCNET y autorizó a otras compañías. En 1981, Standard Microsystems Corporation (SMC) desarrolló el primer controlador LAN en un solo chip basado en el protocolo de paso de testigo de ARCNET. En 1986 se introdujo una nueva tecnología de configuración de chip.

ARCNET tiene un bajo rendimiento, soporta longitudes de cables de hasta 2000 pies cuando se usan concentradores activos. Es adecuada para entornos de oficina que usan aplicaciones basadas en texto y donde los usuarios no acceden frecuentemente al servidor de archivos. Las versiones más nuevas de ARCNET soportan cable de fibra óptica y de par trenzado. Debido a que su esquema de cableado flexible permite de conexión largas y como se pueden tener configuraciones en estrella en la misma red de área local (LAN Local Área Network).

ARCNET es una buena elección cuando la velocidad no es un factor determinante pero el precio sí. Además, el cable es del mismo tipo del que se utiliza para la conexión de terminales IBM 3270 a computadoras centrales de IBM y puede que ya esté colocado en algunos edificios. ARCNET proporciona una red robusta que no es tan susceptible a fallos como Ethernet de cable coaxial si el cable se suelta o se desconecta. Esto se debe particularmente a su topología y a su baja velocidad de transferencia. Si el cable que une una estación de trabajo a un concentrador se desconecta o corta, solo dicha estación de trabajo se va a abajo, no la red entera.

El protocolo de paso de testigo requiere que cada transacción sea reconocida, de modo no hay cambios virtuales de errores, aunque el rendimiento es mucho más bajo que en otros esquemas de conexión de red. ARCNET Plus, una versión de 20 Mbits/seg que es compatible con ARCNET a 2.5 Mbits/seg. Ambas versiones pueden estar en la misma LAN. Fundamentalmente, cada nodo advierte de sus capacidades de transmisión a otros nodos, de este modo si un modo rápido necesita comunicarse con uno lento, reduce su velocidad a la más baja durante esa sesión. ARCNET Plus soporta tamaños de paquetes más grandes y ocho veces más estaciones.

Otra nueva característica en la capacidad de conectar con redes Ethernet, anillo con testigo y Protocolo de Control de Transmisión/Protocolo Internet (TCP/IP, Transmission Control Protocol/Internet Protocol) mediante el uso de puentes (bridges) y encaminadores (routers). Esto es posible porque la versión nueva soporta la norma de control de enlace lógico IEEE 802.2.

El método de acceso a la ARCnet es mediante un protocolo de bus de token que considera a la red como un anillo lógico. El permiso para transmitir un token se tiene que turnar en el anillo lógico, de acuerdo con la dirección de la tarjeta de interfaz de red de la estación de trabajo, la cual debe fijarse entre 1 y 255 mediante un conmutador DIP de 8 posiciones. Cada tarjeta de interfaz de red conoce su propio modo con la dirección de la estación de trabajo a la cual le tiene que pasar la ficha. El moso con la dirección mayor cierra el anillo pasando la ficha al modo con la dirección menor.

- **Ethernet.** Desarrollado por la compañía XERTOX y adoptado por la DEC (Digital Equipment Corporation), y la Intel, Ethernet fue uno de los primeros estándares de bajo nivel. Actualmente es el estándar más ampliamente usado. Ethernet esta principalmente orientado para automatización de oficinas, procesamiento de datos distribuido, y acceso de terminal que requieran de una conexión económica a un medio de comunicación local transportando tráfico a altas velocidades. Este protocolo está basado sobre una topología bus de cable coaxial, usando CSMA/CD para acceso al medio y transmisión en banda base a 10 MBPS. Además de cable coaxial soporta pares trenzados. También es posible usar Fibra Óptica haciendo uso de los adaptadores correspondientes.

Además de especificar el tipo de datos que pueden incluirse en un paquete y el tipo de cable que se puede usar para enviar esta información, el comité especifica también la máxima longitud de un solo cable (500 metros) y las normas en que podrían usarse repetidores para reforzar la señal en toda la red.

## Funciones de la Arquitectura Ethernet

- Encapsulación de datos. Formación de la trama estableciendo la delimitación correspondiente, direccionamiento del nodo fuente y destino, así como la detección de errores en el canal de transmisión.



- Manejo de Enlace. Asignación de canal, resolución de contención, manejando colisiones.
- Codificación de los Datos. Generación y extracción del preámbulo para fines de sincronización Codificación y decodificación de bits.
- Acceso al Canal. Transmisión/Recepción de los bits codificados. Sensibilidad de portadora, indicando tráfico sobre el canal y detección de colisiones, indicando contención sobre el canal.
- Formato de Trama. En una red ethernet cada elemento del sistema tiene una dirección única de 48 bits, y la información es transmitida serialmente en grupos de bits denominados tramas. Las tramas incluyen los datos a ser enviados, la dirección de la estación que debe recibirlas y la dirección de la estación que las transmite. Cada interface ethernet monitorea el medio de transmisión antes de una transmisión para asegurar que no esté en uso y durante la transmisión para detectar cualquier interferencia. En caso de alguna interferencia durante la transmisión, las tramas son enviadas nuevamente cuando el medio esté disponible. Para recibir los datos, cada estación reconoce su propia dirección y acepta las tramas con esa dirección mientras ignora las demás. El tamaño de trama permitido sin incluir el preámbulo puede ser desde 64 a 1518 octetos. Las tramas fuera de este rango son consideradas invalidas.

#### **Campos que componen la Trama.**

- El preámbulo. Inicia o encabeza la trama con ocho octetos formando un patrón de 1010, que termina en 10101011. Este campo provee sincronización y marca el límite de trama.
- Dirección destino. Sigue al preámbulo o identifica la estación destino que debe recibir la trama, mediante seis octetos que pueden definir una dirección de nivel

físico o múltiples direcciones, lo cual es determinado mediante el bit de menos significación del primer byte de este campo. Para una dirección de nivel físico este es puesto en 0 lógico, y la misma es única a través de toda la red ethernet. Una dirección múltiple puede ser dirigida a un grupo de estaciones o a todas las estaciones y tiene el bit de menos significación en 1 lógico. Para direccionar todas



las estaciones de la red, todos los bits del campo de dirección destino se ponen en 1, lo cual ofrece la combinación FFFFFFFFFFFFH.

- Dirección fuente. Este campo sigue al anterior, compuesto también por seis octetos, que identifican la estación que origina la trama.
- Los campos de dirección son además subdivididos: Los primeros tres octetos son asignados a un fabricante, y los tres octetos siguientes son asignados por el fabricante. La tarjeta de red podría venir defectuosa, pero la dirección del nodo debe permanecer consistente. El chip de memoria ROM que contiene la dirección original puede ser removido de una tarjeta vieja para ser insertado en una nueva tarjeta, o la dirección puede ser puesta en un registro mediante el disco de diagnóstico de la tarjeta de interfaces de red (NIC). Cualquiera que sea el método utilizado se deberá ser cuidadoso para evitar alteración alguna en la administración de la red.
- Tipo. Este es un campo de dos octetos que siguen al campo de dirección fuente, y especifican el protocolo de alto nivel utilizado en el campo de datos. Algunos tipos serían 0800H para TCP/IP, y 0600H para XNS.
- Campo de dato. Contiene los datos de información y es el único que tiene una longitud de bytes variable que puede oscilar de un mínimo de 46 bytes a un máximo de 1500. El contenido de ese campo es completamente arbitrario y es determinado por el protocolo de alto nivel usado.
- Frame Check Sequence. Este viene a ser el último campo de la trama, compuesto por 32 bits que son usados por la verificación de errores en la transmisión mediante el método CRC, considerando los campos de dirección tipo y de dato.
- **Modelo SNA.** El modelo SNA tiene las siguientes características: Permite compartir recursos, reconoce perdida de datos durante la transmisión, usa procedimientos de control de flujo, evade sobrecarga y la congestión, reconoce fallos y hace corrección de

errores. Provee interfaces abiertas documentadas. Simplifica la determinación de problemas gracias a los servicios de administración de



la red. Mantiene una arquitectura abierta. Provee facilidad de interconexión de redes. Provee seguridad a través de rutinas de logon y facilidades de encryptamiento. Usa Synchronous Data Link Control (SDLC).

### **Niveles del Modelo SNA.**

- Niveles de Control del Enlace Físico. El enlace físico de control de capas es la capa o nivel más baja en la arquitectura. Este permite el uso de una variedad realística de medios físicos para la interconexión de procedimientos de control. Procedimientos de protocolos típicos para esta capa o nivel son conexiones físicas provistas por líneas de comunicación, módem y la interface EIA RS-232C. Esta capa o nivel no tan solo permite variar tipos de circuitos punto a punto o multipunto, sino que provee los protocolos físicos para establecer, controlar y liberar los circuitos de datos comutados.
- Nivel de Enlace de Datos. Los medios de comunicación físicos (ej.: Línea telefónica) requieren técnicas específicas para ser usadas con el fin de transmitir dato entre sistemas a pesar de la naturaleza de tendencia de error de las facilidades físicas. Estas técnicas específicas son usadas en los procedimientos de control de enlace de dato. Las características primarias de la capa o nivel de enlace de Data de IBM SNA es que esta usa Control de Enlace de Data Síncrono (Synchronous Data Link Control - SDLC) como el protocolo de línea de comunicación.
- Nivel de Control de Ruta. Este nivel provee rutas virtualmente libres de errores entre los últimos orígenes y destinos conectados a la red. Sobre todo, el control de la red abarca o agrupa el establecimiento y manejo de estas rutas a través de la red. El control de sendas o rutas (paths) por lo tanto tiene dos funciones primarias: Enrutar mensajes a través de la red desde el origen hacia las localidades de destino. Segmentar grandes mensajes o combinar pequeños mensajes, llamado segmentar en bloques (blocking), con el propósito de un caudal de transferencia más eficiente a través de la red.

- Nivel de Control de Transmisión. Provee un control básico de los recursos de transmisión de la red. Las funciones que provee son: Número de verificación de secuencia cuando se recibe un mensaje. Encriptamiento de datos. Administración de la rapidez en que los requerimientos enviados de una unidad lógica son recibidos en otra unidad lógica. Soporte para las funciones de frontera para nodos periféricos
- Nivel de Control de Flujo de Datos El flujo de datos en una sesión LU-LU necesita ser controlado de acuerdo con los protocolos de sesión usados y este nivel provee ese control. Las funciones que provee este nivel son: Asignación de números de secuencia de flujo de datos Correlación de la petición y respuesta Soporte para protocolos encadenados gracias a que hace agrupamiento en cadenas de las unidades relacionadas de petición Soporte y refuerzo de la petición de sesión y protocolos de modo de respuesta Soporte y coordinación de los modos de transmisión y recepción de los protocolos de sesión
- Nivel de Servicio de Presentación. Los programas de transacciones se comunican unos con otros, de acuerdo con lo bien definidos protocolos de conversación, usando verbos de conversación. Este nivel define estos protocolos para comunicaciones de programa a programa de comunicación. También, controla el uso del nivel de verbos de los programas de transacciones. Controla la carga y el inicio de los programas de transacción, Mantiene y soporta los modos de transmisión y recepción de protocolos de conversación, Supervisa el uso de los parámetros de los verbos de los programas de transacción, Refuerza las restricciones de los protocolos de secuencia y Procesa verbos de programas de transacciones.
- Nivel de Servicios de Transacción. Es el nivel en el que los programas de servicios de transacción son implementados. Provee los siguientes servicios de usuario final:



Control operativo del límite de sesión LU-LU Arquitectura de Intercambio de Documentos (DIA) para distribución de documentos entre sistemas de información de oficina basados en SNA Servicios Distribuidos SNA (SNADS) para comunicación asincrónica de datos.

- **El modelo OSI.** Según Dordogne (2015) cada uno de los componentes que conforman la arquitectura de la red se deben gestionar de manera eficiente. En respuesta a ello y otros aspectos, surgió el modelo OSI que da cabida a la compleja cadena de eventos, que se generan con el movimiento de datos en una red.

Su origen se remonta en la década de los setenta donde la Organización Internacional para la Normalización o International Standards Organization (ISO) donde inicio el desarrollo de un modelo conceptual para la conexión en red llamado Modelo de Referencia de Interconexión de Sistemas Abiertos del Inglés Open Systems Interconnection Reference Model. En 1984, dicho modelo paso a ser un estándar internacional para las comunicaciones en red OSI para especificar como se desplazan los datos dentro una red.

Por su parte Rivera (2016) enfatiza que las especificaciones del modelo OSI son un estándar abierto, es decir, que se encuentra disponible para todos los interesados. También, este modelo contempla siete capas independientes siendo cada una responsable de un grupo de servicios para el proceso de transmisión de la información entre equipos informáticos interconectados a través de protocolos de un nodo a otro, donde cada capa tiene determinada función como parte del proceso global de la transmisión de datos. Cabe destacar que las dos únicas capas con las cuales interactúa el usuario son la primera, capa física y la última, capa de aplicación.

Es importante destacar que cada capa se comunica solo con la siguiente capa inferior y superior de manera estandarizada, permitiendo la implementación independiente de los servicios en cada una de ellas. Las capas actúan como si estuviesen comunicando con su capa homologa en el otro equipo. Ello permite que un proveedor pueda especializarse en un servicio de una capa e integrarla fácilmente con los servicios de las otras capas formando la solución que se requiere.

A continuación, se describe cada capa del modelo OSI (Microsoft, 2017):

- **Capa Física.** Contempla los elementos físicos de la red, donde se definen las especificaciones eléctricas, mecánicas y funcionales para activar, mantener y desactivar la conexión física. Se



determinan las características físicas del cableado, codificación de señales, conectores, además de las limitaciones de distancia y velocidad.

- Capa de Enlace. Es la interfaz entre la capa física y de red. Se transforman los paquetes de datos en tramas (unidades de datos), colocando la cabecera de enlace y viceversa, que vienen definidas por la arquitectura de red empleada y el protocolo utilizado.
- Capa de Red. Se encarga del tráfico de la red, direccionando los paquetes de datos además de ocuparse de su entrega. Aquí se convierten las direcciones lógicas en direcciones físicas.
- Capa de Transporte. Se encarga de controlar el flujo de datos entre nodos de manera eficiente para su entrega, además de asegurar que los datos entregados están libres de errores, en secuencia y sin pérdidas o duplicación. También el tamaño de los paquetes está determinado por la arquitectura de la red empleada.
- Capa de Sesión. Establece, administra y termina las sesiones de comunicación, por peticiones y respuestas del servicio entre aplicaciones ubicadas en dos o más computadores conectados en red. Aquí se ubican puntos de control en la secuencia de datos, proporcionando cierta tolerancia a fallos dentro de la sesión de comunicación.
- Capa de Presentación. También se le conoce como el traductor del modelo OSI, porque toma los paquetes de la capa de aplicación, para darles formato a los datos, el tipo de codificación, de conversión de datos incluyendo la compresión/descompresión y el cifrado/descifrado.
- Capa de Aplicación. Es la que habilita la interfaz que emplea el usuario en su computadora, por ejemplo, navegar por la red por el protocolo http (hypertext protocol), enviar un correo electrónico por el protocolo smtp (simple mail transfer protocol) o ubicar un archivo para su descarga desde la red por el protocolo ftp

(file transfer protocol).



## **Tipos de redes**

Entre los distintos tipos de Redes, se encuentran los siguientes, diferenciados lógicamente por el tamaño y la cantidad de terminales que abarcan:

- LAN: del inglés Local Área Network o Red de Área Local, que se trata de redes pequeñas (hogareñas o empresariales) en donde cada equipo está conectado al resto.
- WAN: del inglés Wide Área Network o Red de Área Extensa, en este caso las redes se dan entre países enteros o inclusive pueden alcanzar una extensión continental.
- MAN: del inglés Metropolitan Área Network o Red de Área Metropolitana, en este tipo de redes la extensión es mucho mayor, abarcando una ciudad o una pequeña población determinada.

También, según el medio físico utilizado las redes se catalogan en:

- Redes alámbricas: utilizan cables para transmitir datos.
- Redes inalámbricas: utilizan ondas electromagnéticas para enviar y recibir información
- Redes mixtas: unas áreas están comunicadas por cable y otras comunicadas de forma inalámbrica.

## **La topología de red**

Se define como el mapa físico o lógico de una red para intercambiar datos. En otras palabras, es la forma en que está diseñada la red, sea en el plano físico o lógico. El concepto de red puede definirse como “conjunto de nodos interconectados”. Un nodo es el punto en el que una curva se intercepta a sí misma. Lo que un nodo es concretamente depende del tipo de red en cuestión.

El éxito de una red está definido por su potencia, rendimiento y alta disponibilidad, además de la libertad de conexión desde el punto de vista del plano de control, sin importar los elementos físicos y lógicos, del plano de datos para los mismos fluyan por el canal de comunicación cumpliendo con los estándares de comunicación sin importar el fabricante.

Para Banks, Krivan y Linthicum (2015) sobre la topología de red, es importante destacar que no existe una mejor y única topología de red para una empresa. Solo cuando se entienden las principales opciones de topologías, se puede identificar cual es la que funcionara mejor según el tráfico de red o se puede obtener ideas para solucionar problemas en una red existente. A continuación, se describen las topologías de red:

- Red en bus: (bus o “conductor común”) o Red lineal (line) se caracteriza por tener un único canal de comunicaciones (denominado bus, troncal o backbone) al cual se conectan los diferentes dispositivos. Su principal problema son un posible fallo en el cable central y la acumulación de tráfico.
- Red en anillo: (ring) o Red circular, cada estación está conectada a la siguiente y la última está conectada a la primera. Además, puede compararse con la Red en cadena margarita, el cual es un esquema de cableado realizado en cadenas sucesivas entrelazadas (Daisy chain).
- Red en estrella: (star), las estaciones están conectadas directamente a un punto central (concentrador) y todas las comunicaciones se han de hacer necesariamente a través de éste, por cuanto no están conectados entre sí.
- Red en malla: (mesh), cada nodo está conectado a todos los otros, haciéndola muy segura ante un fallo, pero de instalaciones complejas.
- Red en árbol: (tree) o Red jerárquica donde los nodos están colocados en forma de árbol. Desde una visión topológica, la conexión en árbol es parecida a una serie de redes en estrella interconectadas salvo en que no tiene un nodo central. En otras palabras, se trata de una combinación de redes en estrella en la que cada concentrador o switch se conecta a un servidor o a un switch principal.

En la actualidad, las redes emplean principalmente las topologías de tres capas, una que comprende un núcleo de switches que se conectan entre sí y con el proveedor (o los proveedores) de redes



externo(s), otra capa de usuario o de acceso y una capa de agregación entre estos dos que mueve información.

## Conclusiones

- Se deduce que en un entorno de trabajo académico, empresarial, comercial o de cualquier otra índole, el trabajo en red minimiza las redundancias, reduce el tiempo de dedicación y esfuerzo además de facilitar tanto la difusión como el intercambio de información.
- Es de resaltar y profundizar en el conocimiento del modelo OSI, en cada una de sus capas, aun cuando tiene dos únicas capas con las cuales interactúa el usuario que son la primera, capa física y la última, capa de aplicación, existen otras cinco capas donde cada una tiene una función determinada dentro del proceso global de la transmisión de datos entre equipos informáticos interconectados a través de protocolos.
- El éxito del modelo radica en la estandarización de los protocolos para la transmisión de datos que facilitan la interoperabilidad de sistemas heterogéneos de diversos fabricantes tanto de hardware como de software.

## Referencias Bibliográficas

Banks, E., Krivan, P y Linthicum D. (2015). Cómo cumplir con los requisitos de una buena red. Editorial SearchDataCenter.Es. España.

Dordogne, J. (2015). Redes informáticas, nociones fundamentales. Edición quinta. Editorial Eni. España.

Incera J., Cartas R., y Cairó O. (2007). Redes Digitales: Presente y Futuro. Instituto Tecnológico Autónomo de México. Visible en: <http://allman.rhon.itam.mx/~jincera/IntroRedesDigitales.pdf>. Recuperado el: 02/08/2017

Mancilla CM. (2017). REDES DE COMPUTADORAS. Universidad Nacional del Litoral. Visible en:  
<http://www.fca.unl.edu.ar/informaticabasica/Redes.pdf>. Recuperado el 21/08/17

Microsoft. (2017). Definición de las siete capas del modelo OSI y explicación de las funciones. Soporte Técnico. Visible en:  
<https://support.microsoft.com/es-es/help/103884/the-osi-model-s-seven-layers-defined-and-functions-explained>. Recuperado el 17/09/2017.

Peña, C. (2013). Redes la guía definitiva. Editorial REDUSERS. Argentina.

Rivera J. (2016). Fundamentos de redes informáticas. Edición Segunda. Editorial IT Academy Campus. España.



## **Lectura 10. Modelos de gestión de red**

## **Tema 3: Modelos de gestión de red**

1. SNMP.

2. OSI.

# **SNMP**

## **Historia**

- Desde el origen de TCP/IP(1969) se utiliza para gestión herramientas basadas en el protocolo ICMP (Internet-Control Message Protocol) .
- La principal herramienta es PING (Packet INternet Groper), que permite comprobar la comunicación entre dos máquinas, calcular tiempos medios de respuesta y pérdidas de paquetes.
- Al ser parte de la familia de protocolos TCP/IP, todas las máquinas disponen de este protocolo.
- Con el crecimiento exponencial de Internet a partir de los años 80, surge la necesidad de herramientas estándar de gestión más potentes.

## **Historia**

- En los años 80 surgen tres propuestas de estándar de protocolo de gestión para TCP/IP:
  - **HEMS** (High-level entity-management system), que es una generalización del que fue quizás el primer protocolo de gestión usado en Internet (**HMP**)
  - **SNMP** (Simple network management protocol), que es una versión mejorada de SGMP (Simple gateway-monitoring protocol)
  - **CMOT** (CMIP over TCP/IP), que intenta incorporar, hasta donde sea posible, el protocolo (common management information protocol), servicios y estructura de base de dato que se están estandarizando por ISO
- A principios de 1988, el **IAB** recibe las propuestas y decide el desarrollo de SNMP como solución a corto plazo y CMOT a largo plazo (ya que supone que con el tiempo las redes TCP/IP evolucionarán a redes OSI). HEMS es más potente que SNMP, pero como se supone que se va a producir la transición a OSI, se elige SNMP por ser más simple y necesitar menos esfuerzo para desarrollarse (ya que se supone que va a desaparecer con el tiempo).



# Historia

- SNMP se estandariza en los años 90/91, y aunque era una solución simple a corto plazo, el retraso en la aparición de redes OSI y la gran cantidad de redes TCP/IP, le augura una larga vida, mientras que los trabajos en CMOT se ralentizan.
- Actualmente es un estándar utilizado universalmente y se está ampliando a todo tipo de redes (no sólo TCP/IP) incluido OSI.
- Durante su historia ha ido evolucionado desde el estándar simple original. Las principales extensiones aumentan su funcionalidad y cubren problemas de seguridad detectados en el protocolo original.

## Evoluciones más importantes

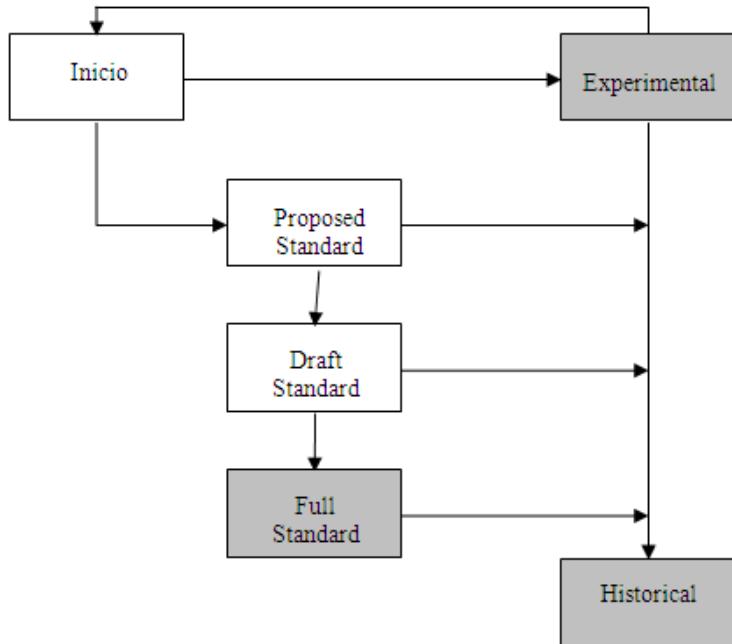
- Extensiones de la MIB:
  - RMON (remote-monitoring), que permite monitorizar subredes como un todo, además de equipos individuales. Aunque es reciente, ya es ampliamente utilizado.
  - Otras extensiones:
    - Independientes de vendedores: principalmente para soportar interfaces de red como Token-Ring o FDDI
    - Dependientes del vendedor para soportar características específicas de sus productos.
- Otras extensiones:
  - De funcionalidad
  - De seguridad

# Evoluciones más importantes

- Extensiones de seguridad y funcionalidad:
  - En Julio de 1992 se proponen 3 documentos. No son compatibles con SNMP original ya que cambia el formato de las cabeceras aunque no las PDU (packet data unit) contenidas en los mensajes, ni el número de PDUs
  - También en Julio de 1992, cuatro organismos proponen una extensión de SNMP llamada SMP (Simple Management Protocol). Al mismo tiempo aparecen 4 implementaciones (dos comerciales y dos públicas). SMP añade tanto nuevas funcionalidades como mejoras de seguridad, por tanto añade nuevas PDUs y los cambios de cabeceras comentados antes.
  - El IETF acepta SMP como base para la versión 2 de SNMP (SNMPv2), creándose dos grupos de trabajo, uno centrado en la seguridad y otro en los demás aspectos.
  - El resultado fueron 12 documentos publicados como *proposed standards* a principios de 1993. Aunque no son estándares finales ya son soportados por multitud de fabricantes.

## Estándares en Internet

- Los estándares son publicados por el IAB (Internet Activities Board) a propuesta del IETF (Internet Engineering Task Force). (El IAB tiene otro grupo denominado IRTF- Internet Research Task Force) como RFCs y Estándares Finales.



# Estándares de SNMP

- **3 Full Standards:**
  - RFC 1155(STD 16): Estructura e identificación de la información de gestión para Internets basadas en TCP/IP. Mayo de 1990.
    - Define como se definen en la MIB los objetos gestionados.
  - RFC 1157 (STD 15): A Simple Network Management Protocol (SNMP). Mayo de 1990.
    - Define el protocolo para gestionar los objetos.
  - RFC 1213 (STD 17): Management Information Base para gestión de red en Internets basadas en TCP/IP:Ñ MIB-II. Marzo de 1991
    - describe los objetos almacenados en la MIB.
- **1 Draft Standard**
- **20 Proposed Standards**
- **4 Experimental Standards:** entre ellos SNMP sobre OSI
- **2 Informational**

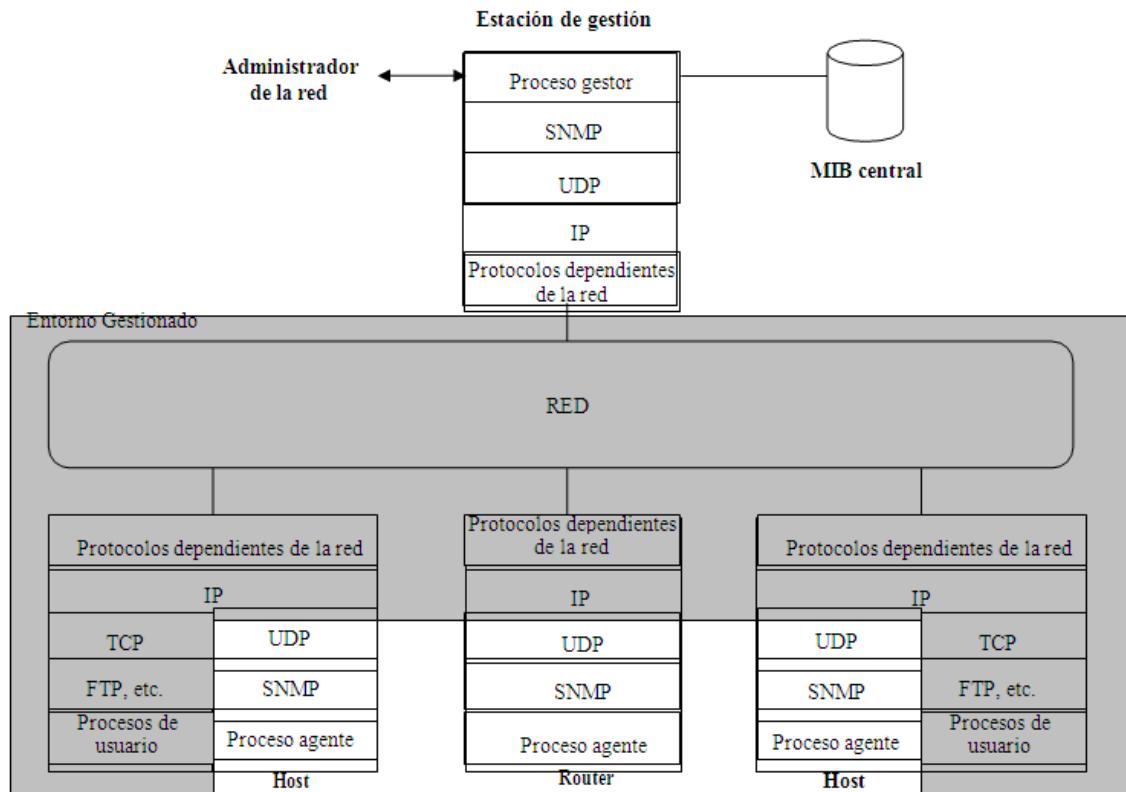
## Arquitectura

- Estructura clásica ya vista:
  - Estación de gestión
  - Agentes de gestión (incluidos agentes proxy)
  - Base de información de gestión (MIB)
  - Protocolo de gestión de red
- De los elementos de la estación de gestión: aplicaciones (para análisis de datos, etc.), intefaz de usuario, capacidad de convertir las solicitudes del usuario apeticiones demonitorización y control a los elementos remotos y base de datos con información de las MIBS de los elementos de la red gestionados, sólo los dos últimos son cubieros por SNMP.
- Los agentes mantendrán una MIB local, atenderán solicitudes de la estación de gestión y podrán enviar de manera asíncrona informes de eventos importantes (*event reporting*). Soporta por tanto los dos mecanismos de comunicación agente-gestor que conocemos.

# Arquitectura

- La MIB local de cada agente mantiene información sobre objetos del recurso que gestiona almacenada en forma de pares atributo-valor. Los objetos están estandarizados para recursos del mismo tipo (p.e., todos los concentradores tendrán los mismos objetos).
- El protocolo (SNMP) enlaza la estación de gestión y los agentes. El protocolo es muy simple, proporcionando las siguientes posibilidades:
  - **Get:** permite a la estación gestora obtener valores de objetos de agentes.
  - **Set:** permite a la estación gestora modificar valores de objetos de agentes.
  - **Trap:** permite a un agente enviar de manera asíncrona la notificación de un evento importante a la estación de gestión.
- En el estándar no se indica nada acerca del número de estaciones gestoras o del ratio gestores/agentes, aunque lo normal es tener dos estaciones gestoras (una de *backup*) y al ser el protocolo simple, el número de agentes por gestor puede ser bastante alto (centenares).

## Entorno de gestión



## MIB

- Almacena una serie de valores relacionados con los elementos gestionados. Cada recurso gestionado se representa por un **objeto**.
- Independientemente del protocolo, una MIB debe cumplir:
  - El objeto u objetos usados para representar un recurso concreto deben ser los mismos en cada nodo (p.e., el número de conexiones TCP abiertas durante un período de tiempo está formado por sesione activas y pasivas. Guardando dos de los tres posibles valores (sesiones activas, pasivas o totales) podemos obtener el tercero. Pero si almacenamos en cada nodo dos distintos, es difícil diseñar un protocolo simple para acceder a esa información).
  - Se debe utilizar un esquema común de representación de la información para permitir la interoperatividad. Esto se consigue en SNMP mediante la definición SMI (structure of management informacion) (RFC 1155).

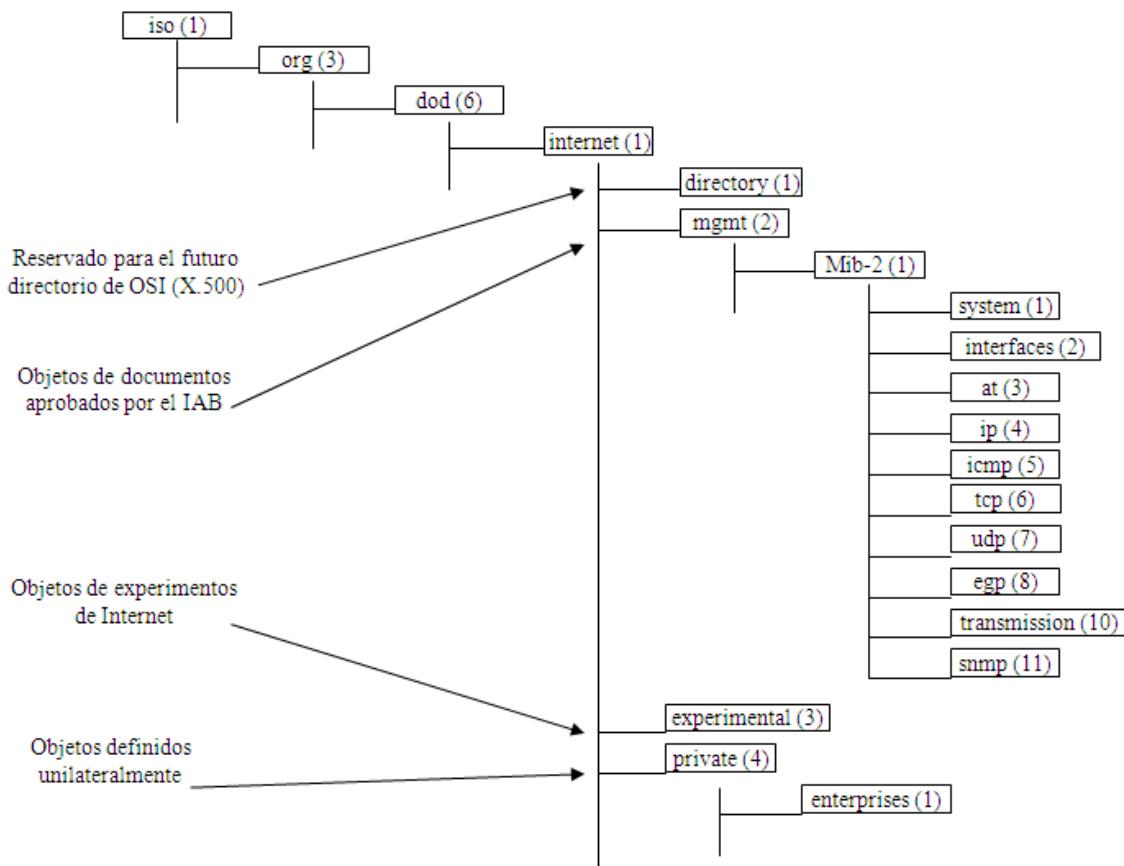
## Estructura de la MIB (SMI)

- Identifica los tipos de datos que se pueden utilizar y cómo los recursos se representan y nombran en la MIB.
- La filosofía de SMI es:
  - **Simplicidad:** sólo tipos de datos simples: escalares y arrays de dos dimensiones de escalares. El protocolo sólo puede acceder a escalares, incluyendo elementos individuales de una tabla.
  - **Posibilidad de extensión:** para poder introducir nuevos objetos, dependientes o independientes del fabricante. La introducción de objetos dependientes del fabricante afectará a la interoperatividad.
- El SMI define:
  - La estructura de la MIB (en ASN.1).
  - Sintaxis y tipos de valores para objetos individuales (en ASN.1).
  - Codificación de los valores de los objetos (en ASN.1).

## Estructura de la MIB (SMI)

- La describiremos sin entrar en profundidad en la definición en ASN.1
- Cada tipo concreto de objeto tiene un identificador único que sirve para nombrarlo. Además como el valor asociado a cada identificador es jerárquico (una secuencia de enteros), dichos identificadores también definen la estructura de la MIB (es una estructura en forma de árbol).
- Empezando por la raíz, existen tres nodos de primer nivel: **iso**, **ccitt** y **join-iso-ccitt**. Como ejemplo, bajo **iso**, un subárbol se reserva para uso de otras organizaciones, y una de ellas es el departamento de defensa de EEUU (**dod**). El RFC 1155 reserva un subárbol de **dod** para la Internet Activities Board (IAB) de la siguiente manera:
  - internet OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
- Es decir, el nodo **internet** tiene el valor de identificador de objeto **1.3.6.1**, que valdrá como prefijo para nodos a niveles más bajos del árbol

## Estructura de la MIB (SMI)



## Estructura de la MIB (SMI)



- El **mgmt** contiene definiciones de información aprobada por el IAB. Actualmente existen dos versiones de la MIB: **mib-1** y **mib-2**. La segunda es una extensión de la primera. Como tienen el mismo identificador sólo uno puede estar presente.
- Se pueden definir objetos adicionales en la MIB de tres maneras:
  - Expandiendo la **mib-2** (por ejemplo se ha introducido RMON) o reemplazandola por una nueva (futura mib-3).
  - Construyendo una MIB experimental para una aplicación que después puede pasar al subárbol **mgmt** (p.e., MIBs de varios medios de transmisión que se han definido, como token-ring (RFC 1231), etc.)
  - Extensiones privadas en el subárbol **private**. Por ejemplo el RFC 1227 define el **MUX** (para multiplexores)
- El objeto **private** actualmente sólo tiene un subárbol (**enterprises**) donde los fabricantes pueden almacenar extensiones propias. Cada fabricante registrado tiene su propio subárbol bajo **enterprises**.

## Estructura de la MIB (SMI)

- Sintaxis y tipos de valores para objetos individuales:
  - Se definen como tipos de ASN.1.
  - Sólo se pueden utilizar los tipos universales más simples de ASN.1 y algunos definidos.
- Codificación de los valores de los objetos:
  - Se utilizan las reglas básicas de ASN.1 (**BER** - basic encoding rules).
  - No son las reglas más compactas o eficientes posibles pero es un esquema estandarizado y ampliamente utilizado.

## Contenidos de la MIB

- La **MIB-II** (RFC 1213) es la segunda versión estandarizada. Es un superconjunto de la **MIB-I** (RFC 1156) con objetos y grupos adicionales
- Criterios especificados en la RFC para incluir un objeto en la MIB-II:
  - Ser esencial para la gestión de configuración o fallos
  - Sólo se permiten objetos cuya modificación provoque daños limitados (refleja la falta de mecanismos de seguridad en SNMP)
  - Tiene uso actual y utilidad
  - MIB-I intentaba mantener los objetos por debajo de 100, en MIB-II se elimina ese límite ya que cada vez hay mas tecnologías que gestionar
  - No es redundante. Se eliminan todos los que se pueden derivar de otros
  - Se eliminan objetos dependientes de implementaciones concretas (p.e., para BSD UNIX)
  - Se minimizan las secciones críticas de código

## Contenidos de la MIB

- Hay 10 grupos:
  - **system**: información general del sistema (7 objetos)
  - **interfaces**: “ de cada uno de los interfaces de red (2 subárboles: 1 y 22 obj.)
  - **at** (address-translation, despreciado): mapeo internet-subred (3 obj.)
  - **ip**: información sobre el protocolo IP (60 objetos con 4 subárboles)
  - **icmp**: 26 objetos
  - **tcp**: 18 objetos y 2 subárboles
  - **udp**: 6 objetos y 2 subárboles
  - **egp**: 20 objetos y 2 subárboles
  - **transmission**: información sobre los esquemas de transmisión y protocolos de acceso (el número de objetos es variable)
  - **snmp**: 30 objetos.

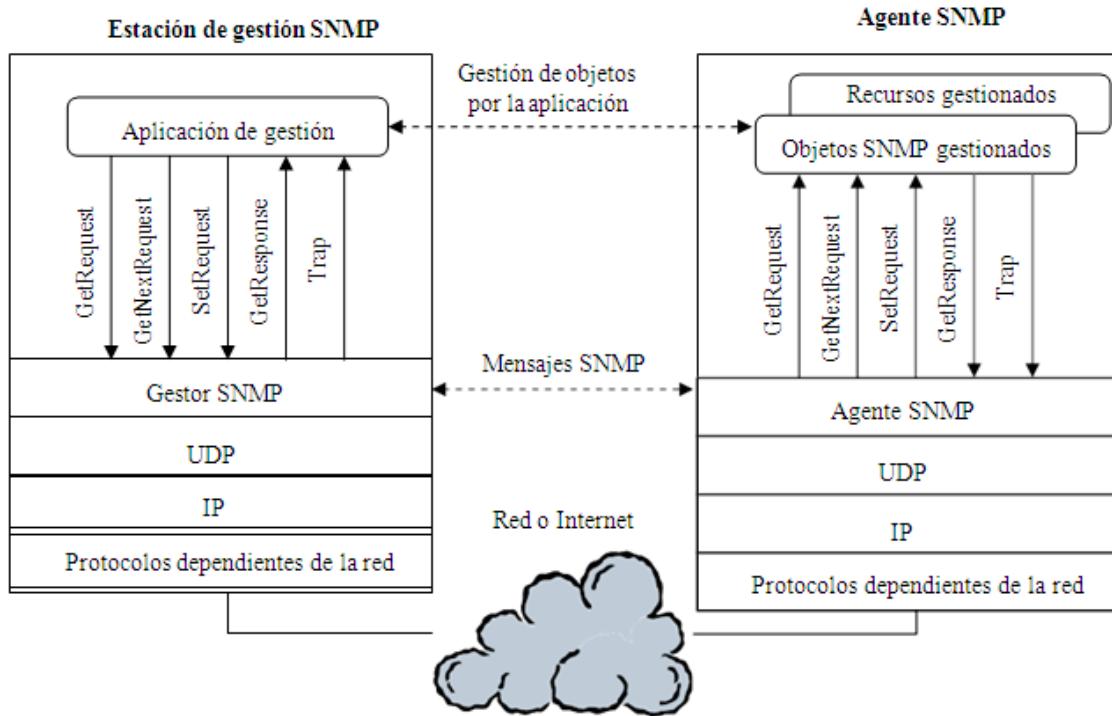


# Protocolo SNMP

- El protocolo (SNMP) enlaza la estación de gestión y los agentes. El protocolo es muy simple, proporcionando las siguientes posibilidades:
  - **Get**: permite a la estación gestora obtener valores de objetos de agentes.
  - **Set**: permite a la estación gestora modificar valores de objetos de agentes.
  - **Trap**: permite a un agente enviar de manera asíncrona la notificación de un evento importante a la estación de gestión.
- No permite modificar la estructura de la MIB añadiendo o eliminando objetos. Por tanto sólo permite acceder a las hojas del árbol.
- Tampoco permite comandos para realizar acciones.
- La seguridad se hace mediante comunidades (un nombre con un conjunto de operaciones permitidas) definidas en los agentes (un agente puede tener varias comunidades definidas). El nombre va en las peticiones y es el único mecanismo de autenticación soportado (seguridad casi nula).
- Estas características hacen simple el protocolo, pero limitan las posibilidades de gestión.
- No vamos a ver la especificación de las 5 primitivas definidas.

## Protocolo

- El protocolo es un protocolo de nivel de aplicación de tipo datagrama (UDP), por tanto no existe sesión, cada intercambio es una transacción independiente entre el gestor y el agente.



## Limitaciones de SNMP

- El protocolo no es muy adecuado para leer grandes cantidades de datos (como tablas de encaminamiento).
- No hay asentimiento de las *Traps*, no sabiendo el agente si ha llegado a la estación gestora.
- Mecanismo trivial de autenticación.
- No soporta comando imperativos directos. Se puede hacer indirectamente modificando un valor que implique una acción (más limitado que comandos directos).
- El modelo de la MIB es limitado y no soporta operaciones complejas como consultas basadas en valores o tipos de objetos.
- No soporta comunicaciones gestor-gestor, de tal manera que por ejemplo una máquina gestora no puede obtener información sobre dispositivos o redes de otras máquinas gestoras.
- Como ya dijimos, algunas de estas limitaciones se han mejorado con extensiones como RMON o SNMPv2.

## SNMPv2

- No lo vamos a ver.
- Mejora SNMP en:
  - Estructura de la Información gestionada (SMI)
  - Primitivas del protocolo
  - Capacidad de comunicación gestor a gestor
  - Seguridad
- Las dos primeras permiten la definición y acceso a tipos de datos complejos no soportados por SNMP. También permite un acceso más eficiente a los datos (tablas, etc.) que sí soportaba SNMP.
- En seguridad se introduce cifrado con clave pública y firma digital.



# OSI

## Introducción

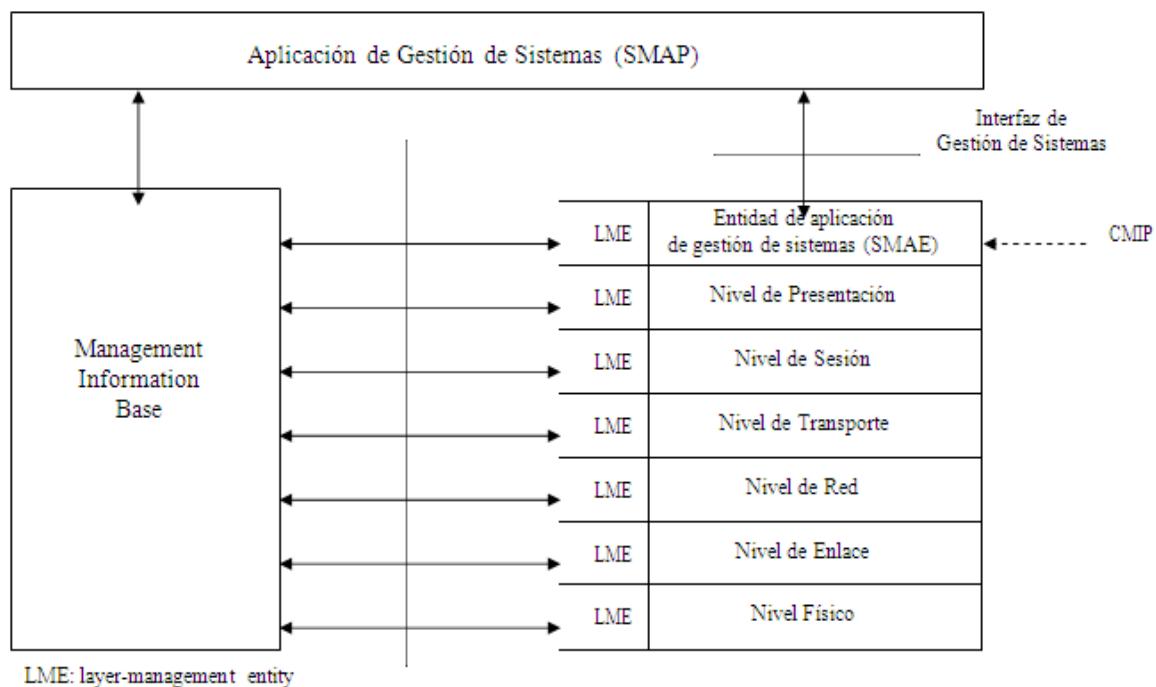
- Conjunto de muchos estándares (es el más amplio y complejo de todos los desarrollados por ISO) desarrollados conjuntamente por ISO y CCITT para gestión de redes OSI.
- CCITT tiene reservada la serie de números X.700 para este conjunto de estándares.
- El conjunto de estándares se denomina **Gestión de sistemas OSI** (*OSI systems management*) e incluye la definición de:
  - Un servicio de gestión (CMIS - Common Management Information Service).
  - Un protocolo de gestión (CMIP - Common Management Information Protocol).
  - Una base de datos.
  - Otros conceptos relacionados.
- En OSI se utiliza el término **gestión de sistemas** en vez de gestión de red.

# Estándares

- Actualmente existen 37 estándares.
- El primero fue el ISO 7498-4 (X.700) que especifica el entorno de gestión para el modelo OSI (introducción general a los conceptos básicos de gestión: arquitectura, terminología utilizada, etc.).
- Los podemos agrupar en los siguientes apartados:
  - **Entorno de gestión e Introducción** (2 documentos): El ISO 7498-4, y el ISO 10040 (X.701) que introduce y describe los demás documentos.
  - **CMIS/CMIP** (6 documentos): especifican el CMIS que proporciona servicios a las aplicaciones de gestión y el CMIP que soporta el CMIS.
  - **Funciones de gestión** (22 documentos): especifican las funciones específicas que se definen para el sistema de gestión OSI.
  - **Modelo de información de gestión** (5 documentos): especifica la MIB.
  - **Gestión de niveles OSI** (2 documentos): especifican información, servicios y funciones relacionadas con niveles específicos de la torre OSI.

## Modelo de arquitectura (CMIS)

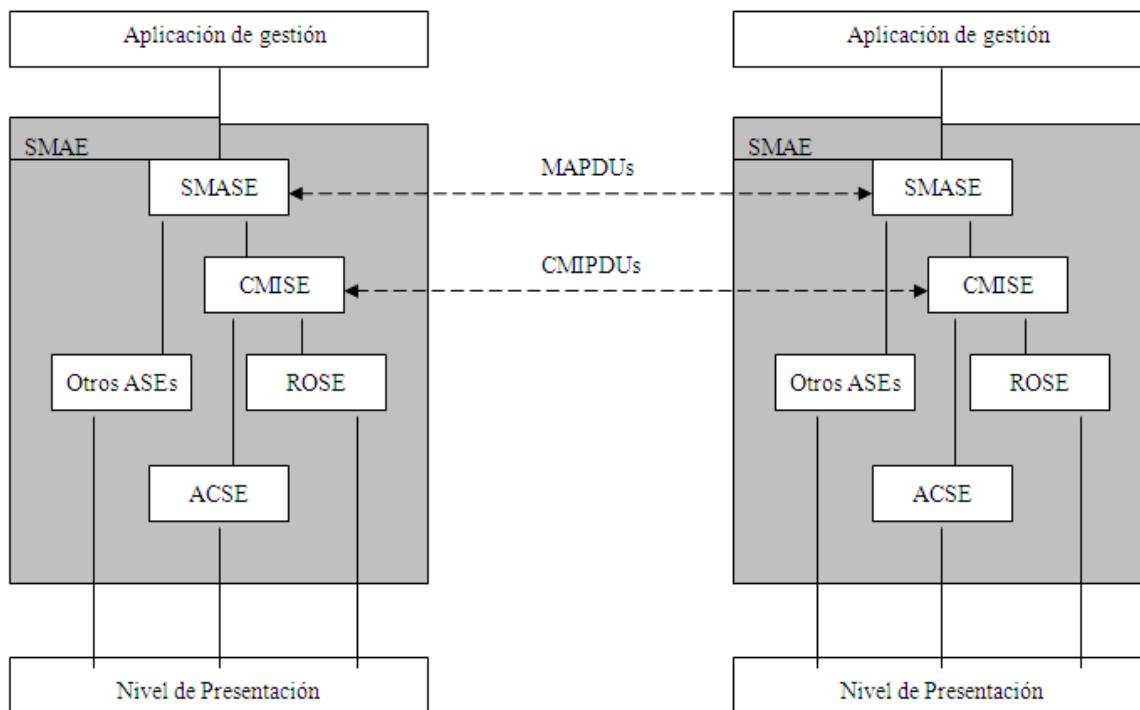
- Un equipo gestionado dentro del entorno OSI seguirá el siguiente modelo de arquitectura:



## Modelo de arquitectura (CMIS)

- Los elementos clave de este modelo de arquitectura son:
  - Aplicación de gestión de sistemas (SMAP - systems-management application process):** Software local de un equipo (sistema) gestionado que implementa las funciones de gestión para ese sistema (host, router, etc.). Tiene acceso a los parámetros del sistema y puede, por tanto, gestionar todos los aspectos del sistema y coordinarse con SMAPs de otros sistemas.
  - Entidad de aplicación de gestión de sistemas (SMAE - systems- management application entity):** Entidad de nivel de aplicación responsable del intercambio de información de gestión con SMAEs de otros nodos, especialmente con el sistema que hace las funciones de centro de control de red. Para esta función se utiliza un protocolo normalizado (CMIP)
  - Entidad de gestión de nivel (LME - layer-management entity):** Proporciona funciones de gestión específicas de cada capa de la torre OSI.
  - Base de información de gestión (MIB).**

## Estructura de la entidad SMAE



MAPDU = management-application protocol data unit.

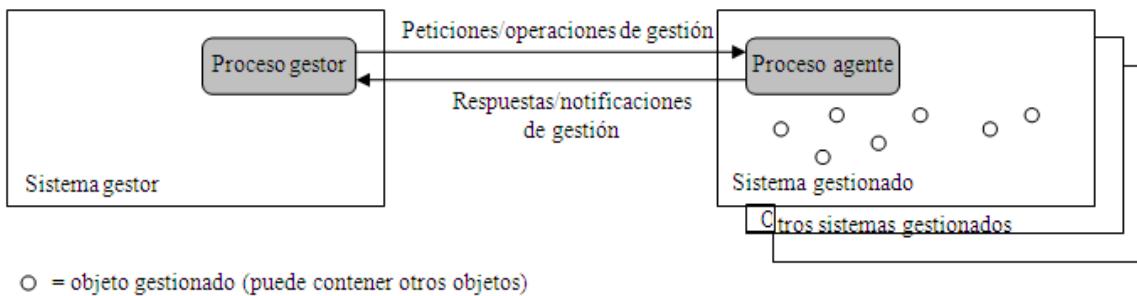
CMIPDU = common management-information protocol data unit.

# Estructura de la entidad SMAE

- Como cualquier entidad de nivel de aplicación se define a nivel lógico como un conjunto de elementos de servicio de aplicación (**ASE - application-service element**). En concreto en la figura destacamos cuatro:
  - De propósito general (diseñadas para ser útiles en muchas aplicaciones):
    - ACSE (*association-control-service element*).
    - ROSE (*remote-operations-service element*).
  - Específicas para gestión de sistemas:
    - **SMASE (system-management application-service element)**: implementa funciones básicas de gestión en las áreas de gestión de fallos, costes, configuración, prestaciones y seguridad. Proporciona servicios al gestor de red y a las aplicaciones de gestión (p.e., SMAP).
    - **CMISE (common management information service element)**: proporciona funciones básicas de gestión que soportan las 5 áreas funcionales. El SMASE delega en este elemento aquellas funciones que requieren comunicación con otros sistemas.

## Arquitectura

- Todos los elementos descritos deben existir en cada uno de los elementos a gestionar en el sistema (red).
- El SMAP puede tomar el papel de **agente** o de **gestor**. El papel de **gestor** corresponde a lo(s) centro(s) de control de red, y el de **agente** a los sistemas gestionados.
- Un **gestor** solicita información o solicita la ejecución de comandos a los sistemas gestionados. El **agente** interacciona con el **gestor** y es responsable de administrar los objetos de su sistema.



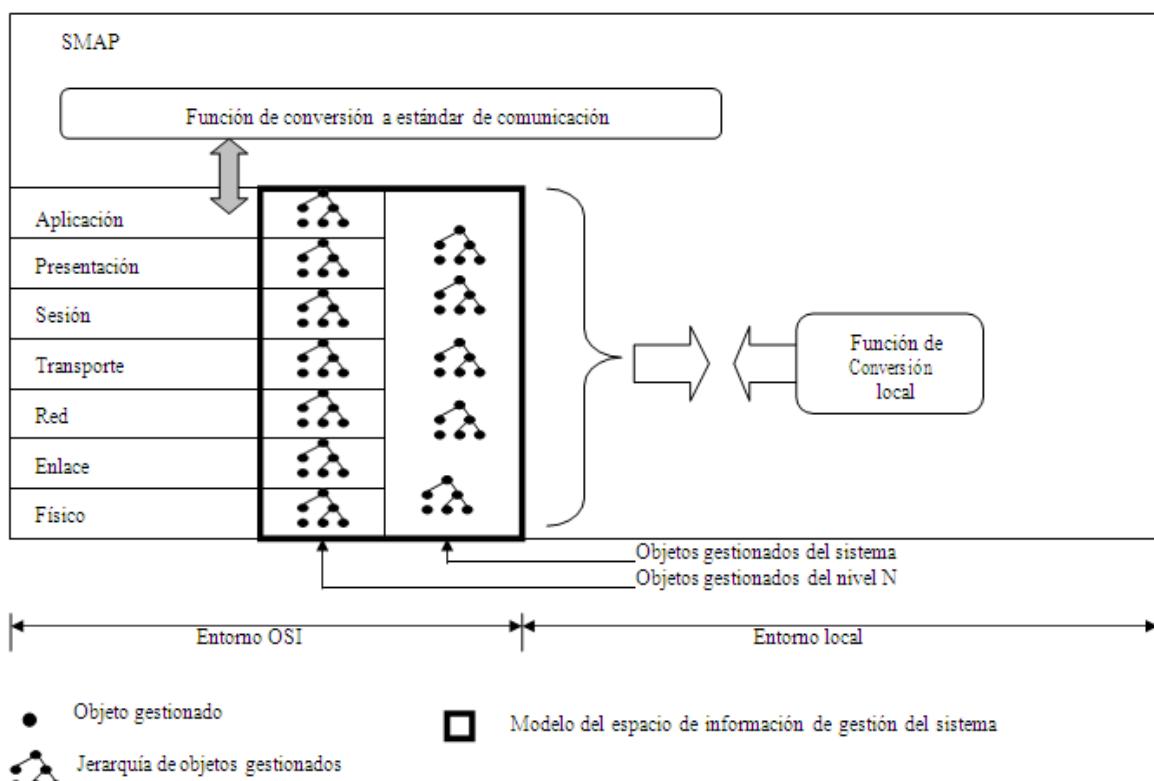
○ = objeto gestionado (puede contener otros objetos)



## Almacenamiento de la información

- La forma de representación y almacenamiento de los datos en un agente, a partir de la cual se deriva la información de gestión no se estandariza.
- Una función local se utiliza para convertir la información relacionada con los objetos gestionados a un formato en que se pueda almacenar localmente y ser usada por el software de gestión local.
- Puesto que para que exista interacción con otros sistemas es necesaria una forma estándar de representación, otra función se encarga de realizar las conversiones necesarias.
- La información se almacena en **objetos** definidos por atributos, operaciones que se pueden realizar sobre él, notificaciones que puede emitir y sus relaciones con otros objetos (es una representación orientada a objetos).
- Los objetos representan recursos, y las operaciones se realizan sobre los objetos directamente, no sobre los recursos que representan.
- Aunque un recurso soporte más operaciones, sólo aquellas definidas para el objeto correspondiente están disponibles para la gestión del sistema.

## Almacenamiento de la información

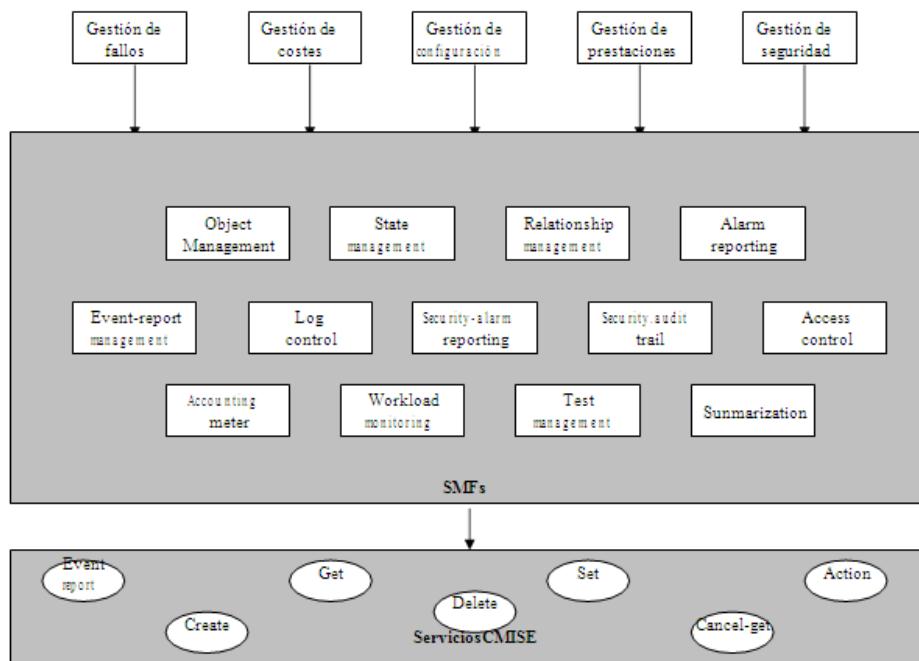




## Funciones de gestión de sistemas

- Las áreas funcionales de gestión (**SMFA - system-management functional area**) que define OSI son muy amplias y se definen a un nivel descriptivo. De hecho estas áreas no están estandarizadas como tales.
- Cada una de estas áreas supone el uso de funciones específicas, y hay un considerable solapamiento entre estas funciones de soporte.
- Se estandarizan un conjunto de funciones específicas, llamadas **SMFs (system management functions)**. Cada estándar de SMF define la funcionalidad para soportar los requisitos de las SMFAs. Una SMF determinada puede dar soporte a varias SMFAs y cada SMFA requiere varias SMFs.
- Cada estándar de SMF define su funcionalidad y proporciona una correspondencia entre los servicios de la SMF y de la CMISE. Cada SMF puede utilizar los servicios de otras SMFs y de la CMISE.

## Funciones de gestión de sistemas



## Funciones de gestión de sistemas

- Están estandarizadas 13 SMFs:
  - **Object management:** soporta la creación y borrado de objetos gestionados y la lectura y cambio de atributos de objetos. También especifica las notificaciones que se deben enviar cuando cambia el valor de un atributo.
  - **State management:** especifica el modelo de representación del estado de gestión de un objeto. Proporciona servicios para soportar ese modelo.
  - **Relationship management:** especifica el modelo de representación y gestión de relaciones entre objetos. Proporciona servicios para soportar ese modelo.
  - **Alarm reporting:** soporta la definición de alarmas de fallos y las notificaciones utilizadas para comunicarlas.
  - **Event-report management:** soporta el control de informe de eventos (notificaciones), incluyendo la especificación de los receptores de la notificación, la definición de notificaciones, y la especificación de criterios para generar y distribuir notificaciones.

## Funciones de gestión de sistemas

- **Log control:** soporta la creación de históricos, la creación y almacenamiento de registros en históricos, y la especificación de criterios para realizar históricos.
- **Security-alarm reporting:** soporta la definición de alarmas de seguridad y las notificaciones utilizadas para comunicarlas.
- **Security-audit trail:** especifica los tipos de informe de eventos que debería contener un histórico utilizado para evaluación de seguridad.
- **Access control:** soporta el control de acceso a la información y operaciones de gestión.
- **Accounting meter:** proporciona informes de la utilización de los recursos del sistema y un mecanismo para poner límites.
- **Workload monitoring:** soporta la monitorización de atributos de los objetos relacionados con las prestaciones del recurso.
- **Test management:** soporta la gestión de procedimientos de prueba y diagnóstico.
- **Summarization:** soporta la definición de medidas estadísticas para los atributos y la comunicación de la información resumida.



## MIB

- El estándar ISO 10165-1 (X.720) presenta un modelo general de la MIB en sistemas OSI. En concreto:
  - Define el modelo de información de los objetos gestionados y sus atributos.
  - Define los principios para nombrar los objetos y sus atributos, de manera que puedan ser identificados y accedidos por los protocolos de gestión.
  - Define la estructura lógica de la información de gestión (*SMI -structure of management information*).
  - Describe el concepto de clases de objetos gestionados y las relaciones en las que pueden participar, incluyendo herencia, especialización, y polimorfismo.

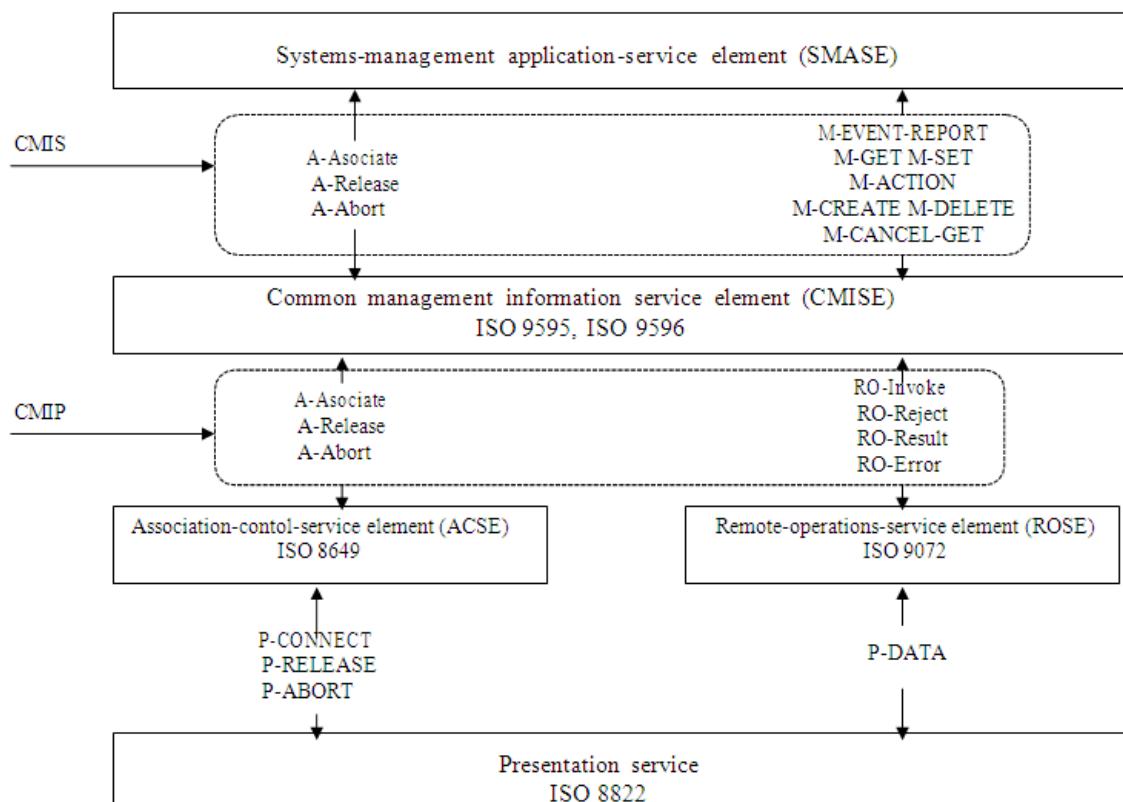
## Estructura de la MIB (SMI)

- Está definido en ASN.1.
- La unidad básica de información es el **objeto**. Un objeto puede incluir:
  - **Atributos**: variables que representan características de los recursos gestionados
  - **Comportamientos**: acciones que pueden ser disparadas por un **gestor**.
  - **Notificaciones**: informe de eventos que pueden ser disparados por determinados eventos.
- La definición ASN.1 sigue el formato de **clases de objetos**. Para cada clase, la información actual se representa mediante **instancias de objeto**. Es decir, puede haber varias instancias de un objeto para una determinada clase, cada una con valores de atributos diferentes.
- Existen dos formas de crear jerarquías para estructurar la MIB:
  - Definición de subclases, o clases derivadas de otras clases que heredan sus características.
  - Una instancia de objeto puede estar contenida en otra instancia de objeto.

## Protocolo (CMIS/CMIP)

- El intercambio de información entre dos entidades (**gestor, agente**) es una de las funciones básicas del sistema de gestión OSI. Esta función de intercambio de información en el sistema de gestión OSI se conoce como CMISE (*common management information service element*). Como la mayoría de las áreas de funcionalidad en OSI, CMISE se especifica en dos partes:
  - La interfaz con el usuario, especificando los servicios proporcionados. Se denomina CMIS (*common management information service*).
  - El protocolo, especificando el formato de las PDUs (*packet data unit*) y los procedimientos asociados. Se denomina CMIP (*common management information protocol*).
- El CMIS proporciona 7 servicios para realizar operaciones de gestión mediante primitivas de servicio. Además los usuarios necesitan establecer asociaciones entre CMISEs para poder realizar operaciones de gestión. Para ello existen 3 primitivas que proporciona el ACSE (*association-control-service element*) y que pasan de manera transparente el CMISE.
- Para los servicios de gestión el CMISE emplea el CMIP para intercambiar PDUs, para los servicios de asociación, no interviene el CMIP.

## Protocolo (CMIS/CMIP)



## Protocolo (CMIS/CMIP)

- Primitivas de servicio CMIS:
  - *M-EVENT-REPORT*: usado por un agente para notificar la ocurrencia de un evento a un **gestor**.
  - *M-GET*: Usado por un **gestor** para obtener información de un **agente**.
  - *M-SET*: Usado por un **gestor** para modificar información de un **agente**.
  - *M-ACTION*: Usado por un **gestor** para invocar un procedimiento predefinido especificado como parte de un objeto de un **agente**. La petición indica el tipo de acción y los parámetros de entrada.
  - *M-CREATE*: Usado para crear una nueva instancia de una clase de objetos.
  - *M-DELETE*: Usado para eliminar uno o más objetos.
  - *M-CANCEL-GET*: Usado para finalizar una operación *GET* larga.
- No vamos a ver las PDUs del CMIP. Las primitivas de servicio de CMIS se convierten en las correspondientes PDUs que utilizan los servicios del ASCE, del ROSE y del servicio de presentación.

## Lectura 11. Gestión y planificación de redes



**SEP**  
SECRETARÍA DE  
EDUCACIÓN PÚBLICA



CENTRO NACIONAL  
DE EVALUACIÓN PARA  
LA EDUCACIÓN SUPERIOR, A.C.



**EDITORIAL:**  
**guiaceneval.mx**



## Tema 1: Introducción a la gestión y planificación de redes

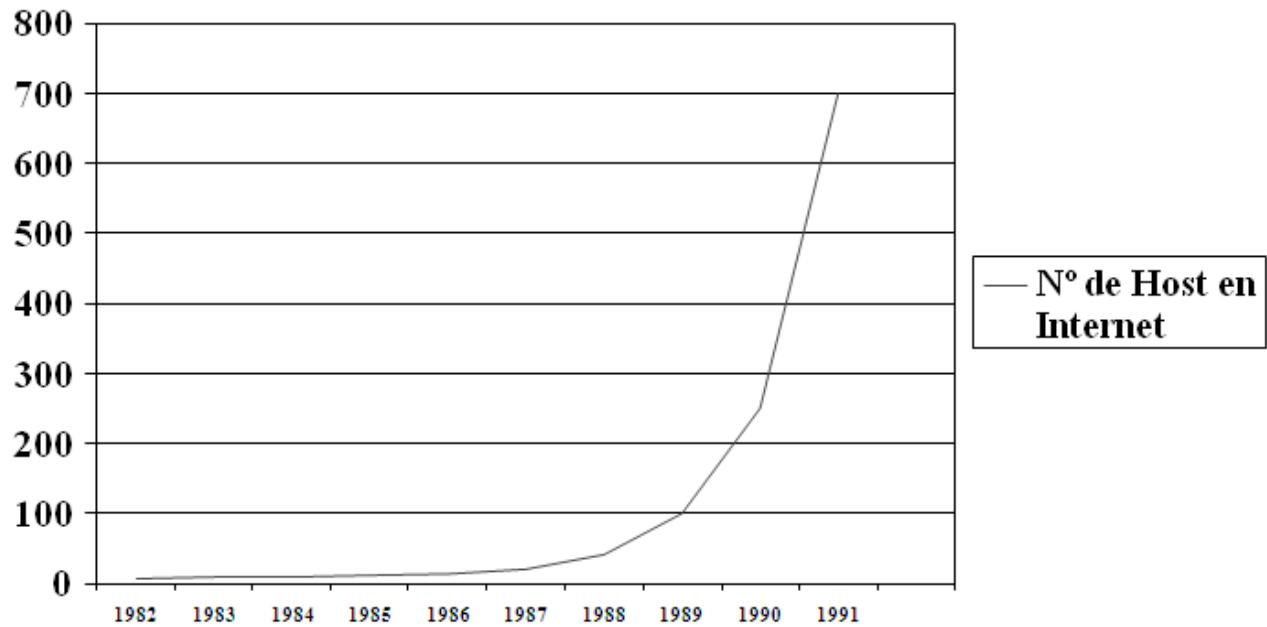
1. Introducción general
2. Objetivos de la gestión de redes
3. Objetivos de la planificación de redes
4. Sistemas de gestión de red

## Gestión de redes

- Disciplina reciente (finales de los 80)
- Definición, desarrollo y utilización de herramientas para gestionar diversos aspectos de las redes de comunicaciones.
- Nos centramos en redes TCP/IP y OSI.

## Complejidad de las redes

(hasta finales de los 80 con ICMP, PING, etc., era suficiente)



## Necesidad de herramientas de gestión de red

- Redes cada vez más complejas
- Redes y recursos se hacen indispensables para las organizaciones
- Componentes diversos y de fabricantes diversos



Imposibilidad de gestión no automatizada



# Necesidad de estándares

- Elementos diversos: sistemas finales, bridges, routers, etc.
- Elementos de diversos fabricantes.



Dos estándares: Familia SNMP y OSI

## **SNMP (Simple Network Management Protocol)**

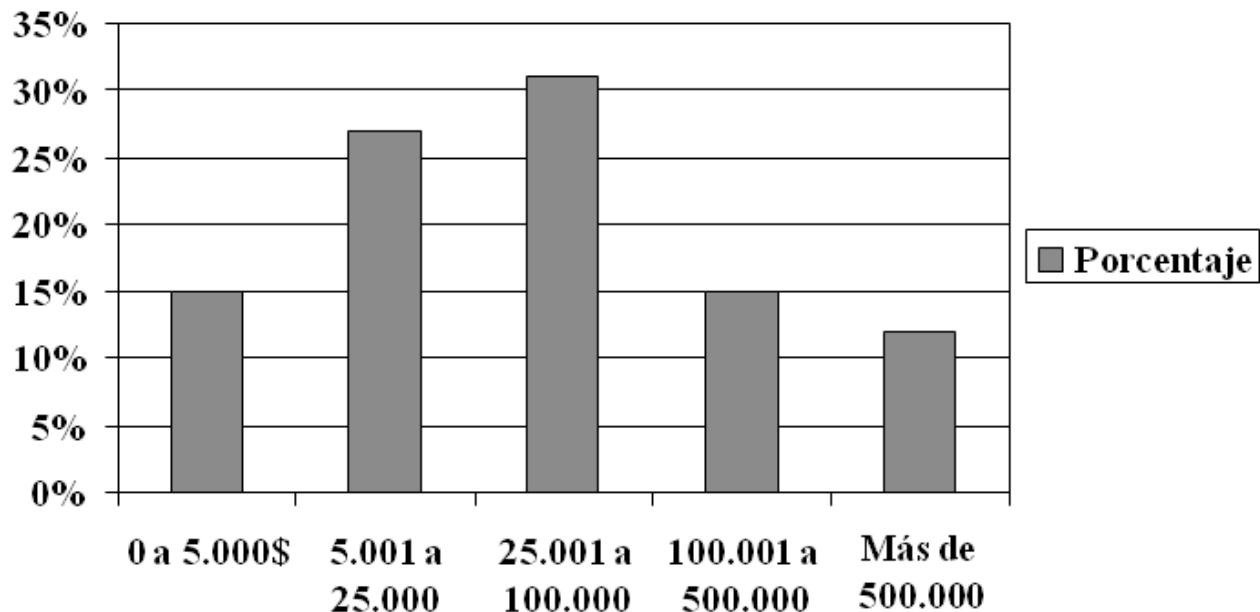
- Conjunto de estándares incluyendo un protocolo, una especificación de estructura de base de datos y un conjunto de objetos
- Adoptado como estándar para TCP/IP en 1989
- Actualizado con mejoras de seguridad y otras en 1993: SNMPv2
- Actualmente en estudio su utilización en redes OSI además de TCP/IP
- El más utilizado hoy en día

## OSI

- Conjunto de estándares (grande y complejo) incluyendo un conjunto de aplicaciones de propósito general, un servicio de gestión, un protocolo, una especificación de estructura de base de datos y un conjunto de objetos
- Actualmente en desarrollo: algunas partes ya estandarizadas y otras en desarrollo.
- Actualmente poco usado (poco a poco se va empezando a utilizar) debido a su complejidad y su no total estandarización.

## Gastos en gestión de red

(1000 empresas más importantes de EEUU)

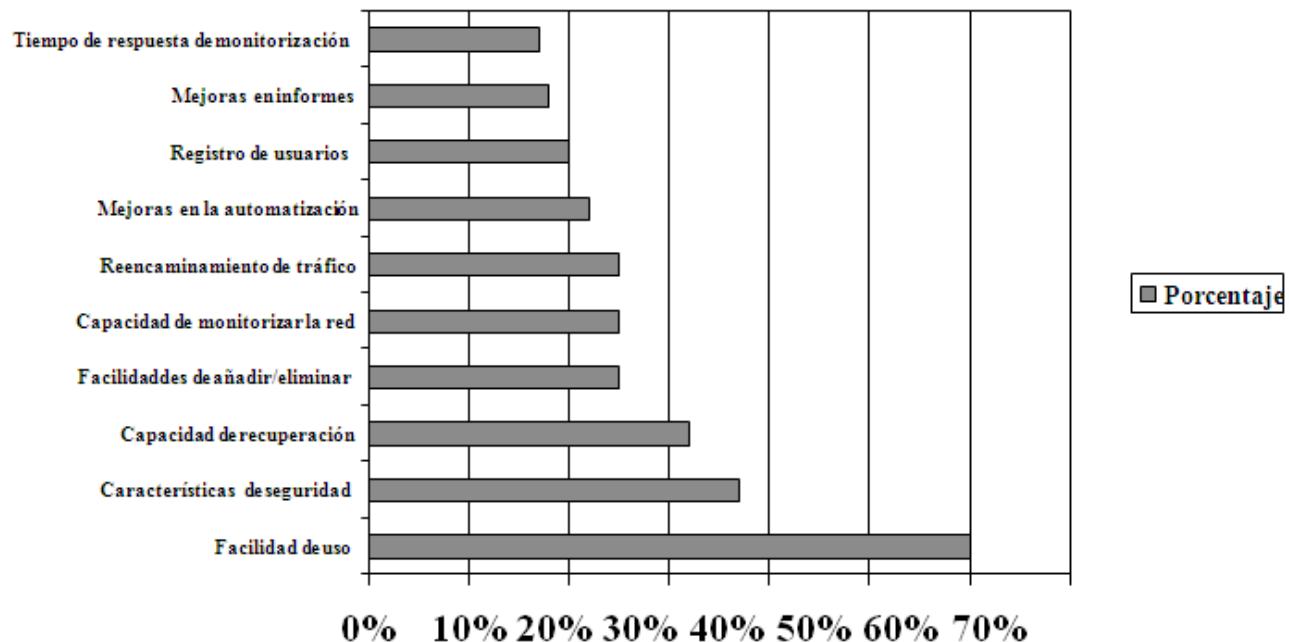


# Gastos en gestión de red

(100 empresas más importantes de EEUU)

- El 15% de los gastos totales en sistemas de información en gestión
- Gasto medio anual por empresa: 1,3 millones de dólares

## Requisitos de los gestores de red



# Justificaciones de inversión en gestión de red

Terplan (1992)

- **Control de recursos estratégicos de la empresa:** Las redes y servicios se han vuelto vitales para las empresas. Sin un control efectivo o se consiguen los resultados esperados de ellas.
- **Control de la complejidad:** El crecimiento de las redes, usuarios, interfaces, protocolos y vendedores complican la gestión.
- **Mejor servicio:** Los usuarios esperan igual o mejor servicio a medida que la información y los recursos informáticos crecen y se distribuyen.
- **Balance de necesidades:** Las organizaciones tienen diversos usuarios con diferentes necesidades y niveles de soporte, con requisitos específicos de prestaciones, disponibilidad y seguridad.
- **Reducción de tiempos de no funcionamiento:** Cuanto más vital se vuelve la red, más debe aproximarse su disponibilidad al 100% del tiempo
- **Control de costes:** Se debe controlar la red para cubrir las necesidades de los usuarios a un coste razonable.

## Áreas funcionales de la Gestión de Redes

- Los análisis cualitativos anteriores nos dan una idea de las necesidades a cubrir por la gestión de red.
- OSI define un división en áreas funcionales para cubrir esas necesidades (es parte del estándar OSI de gestión).
- Esta división ha sido aceptada como útil para cualquier sistema de gestión (OSI o



no)

# Áreas funcionales definidas por OSI

- Gestión de fallos
- Gestión de costes
- Gestión de configuración e identificación
- Gestión de prestaciones
- Gestión de seguridad

## Gestión de fallos

- Mantener funcionando correctamente la red como un todo y cada uno de sus elementos individualmente.
- Cuando algo falla, es importante, tan rápido como sea posible:
  - Determinar exactamente cual es el fallo
  - Aislar el resto de la red del fallo para que continúe funcionando sin interferencias
  - Reconfigurar o modificar la red de manera que se minimice el impacto del fallo en las operaciones de la organización
  - Reparar o sustituir los componentes que han fallado.
- Hay que distinguir entre fallo y error (los errores son normales y la mayoría de las veces se corrigen solos: CRC, etc.)



## Gestión de fallos (Requisitos de los usuarios)

- Esperan soluciones rápidas y seguras.
- Toleran fallos ocasionales, pero esperan estar informados de manera inmediata del fallo y su rápida solución. Para conseguirlo:
  - necesidad de funciones muy rápidas de detección y diagnóstico.
  - Se puede minimizar el impacto utilizando componentes y rutas redundantes.
  - Despues de corregir un fallo el servicio debe asegurar que no se ha resuelto de verdad u no se han introducido nuevos problemas: se denomina *control y seguimiento de fallos*.
- Esperan estar informados del estado dela red, incluyendo paradas de mantenimiento programadas y no programadas

## Gestión de costes

- En muchas organizaciones, diferentes divisiones, centros de gasto o proyectos, son facturados por el uso de los servicios de red.
- Aun cuando esto no ocurra, es necesario mantener información sobre el uso de los recursos de red por usuarios o tipos de usuarios:
  - Determinados usuarios pueden estar abusando de sus privilegios de acceso y cargar la red afectando a los demás usuarios
  - Los usuarios pueden hacer un uso ineficiente de la red, y el gestor puede ayudar a cambiar procedimientos para aumentar la efectividad.
  - Conocer en detalle las actividades de los usuarios será muy útil para planificar el crecimiento adecuado de la red.

## Gestión de costes (Requisitos de los usuarios)

- Esperan información sobre sus costes y su uso de la red.
- Esperan la existencia de mecanismos que aseguren la autorización de acceso a dicha información (control de confidencialidad)

## Gestión de configuración

- Los modernos elementos físicos y lógicos (p.e., controlador de dispositivos de un S.O.) de una red se pueden configurar para realizar diversas tareas. Un dispositivo se puede configurar por ejemplo para ser un nodo final, un router o ambas cosas.
- Una vez elegido el papel de cada dispositivo el gestor debe elegir el software adecuado y los atributos y valores para ese dispositivo.
- La configuración también se encarga de inicializar y apagar adecuadamente parte de la red.
- También se encarga de mantener, añadir y actualizar la relación entre los componentes y su estado durante la operación de la red.
- Automatización de tareas (encendido/apagado de interfaces, etc.).
- Identificación de elementos (gestión de identificación).



## Gestión de conf. (Requisitos de los usuarios)

- El gestor debe cambiar la conectividad de la red cuando las necesidades de los usuarios cambian
  - La reconfiguración de la red es a menudo necesaria como consecuencia de evaluación de prestaciones o a actualizaciones, recuperación de fallos o problemas de seguridad
- Los usuarios a veces quieren o necesitan estar informados del estado de los componentes y recursos de la red. Por ello cuando se hacen cambios en la red, se les debe informar. Esta reconfiguración puede ser debida a labores de rutina o necesidades del usuario. Antes de reconfigurar, el usuario quiere conocer como van a quedar las cosas (por ejemplo si les cambia el S.O. de su máquina).

## Gestión de prestaciones

- Las redes modernas están compuestas de muchos componentes que se comunican y comparten datos y recursos. La efectividad de una aplicación depende cada vez más de unas prestaciones adecuadas de la red.
- Abarca dos grandes categorías funcionales:
  - **Monitorización:** Seguimiento de la actividad de la red
  - **Control:** Realización de ajustes para mejorar las prestaciones.
- Algunas cosas a plantearse son:
  - ¿Cuál es el nivel de utilización de la capacidad?
  - ¿Hay excesivo tráfico?
  - ¿Las prestaciones se han reducido a niveles inaceptables?
  - ¿Hay cuellos de botella?
  - ¿Está aumentando el tiempo de respuesta?
- Para esta gestión hay que identificar los valores relevantes a monitorizar de la red y definir las métricas adecuadas . Esto permitirá obtener estadísticas que ayuden a mantener las prestaciones adecuadas

(eliminar cuellos de botella reconfigurando rutas o ampliando líneas,  
etc.)



## Gestión de prest. (Requisitos de los usuarios)

- Antes de utilizar una aplicación los usuarios pueden querer saber cosas como el tiempo medio y de caso peor de respuesta y la fiabilidad de los servicios de red. Estos datos deben ser suficientemente conocidos por el gestor para contestar.
- Los usuarios esperarán siempre buenos tiempos de respuesta

## Gestión de seguridad

- Generación, distribución y almacenamiento de claves cifradas.
- Mantenimiento y distribución de *passwords* y otras autorizaciones o controles de acceso a la información.
- Monitorización y control del acceso a los ordenadores de la red y a parte o toda la información de la información de gestión de los nodos de la red.
- Registros (*logs*): recolección, almacenamiento y procesamiento de registros de auditoría y seguridad.
- Gestión (definir, lanzar/parar) las facilidades de registro.

# Gestión de segur. (Requisitos de los usuarios)

- Los usuarios quieren conocer que las políticas de seguridad son las adecuadas y efectivas (otros no pueden acceder a su información, etc.).
- También que las facilidades de seguridad son ellas mismas seguras.

## Planificación de redes

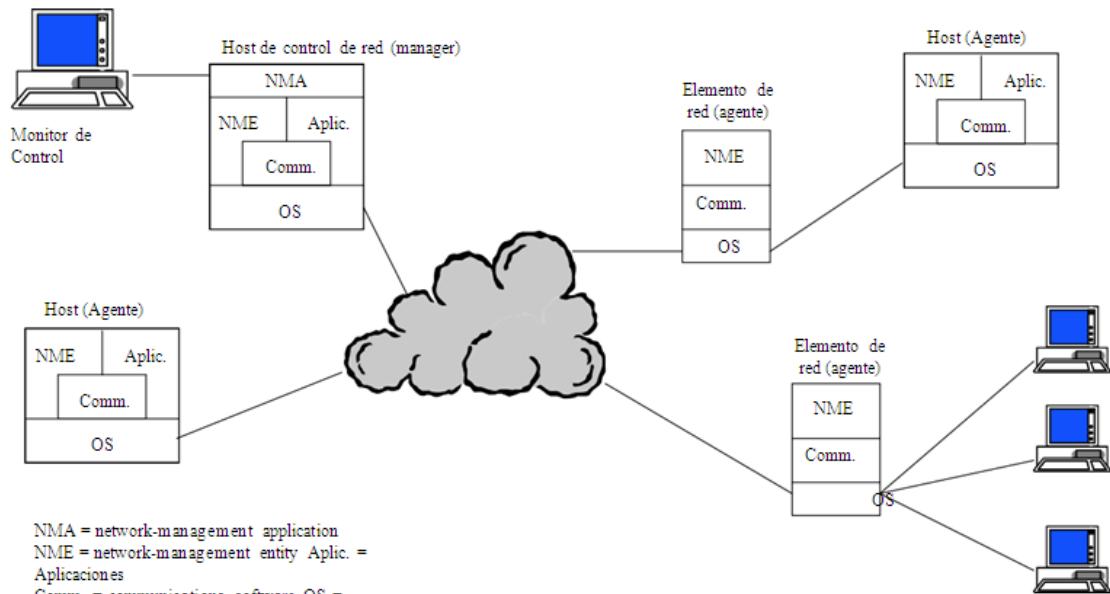
- Mecanismos para tener en cuenta las áreas anteriores a la hora de diseñar o ampliar una red de comunicaciones.
- Por tanto el estudio de las áreas anteriores, junto con algunos conceptos más (simulación, etc.) cubre también la planificación de redes.



# Sistemas de gestión de red

- Colección de herramientas para monitorización y control compuesto por:
  - Una única interfaz de operador con un completo pero *user-friendly* conjunto de comandos para realizar la mayoría de las tareas de gestión. Está diseñado para ver la red completa como una arquitectura unificada, con direcciones y etiquetas asignadas a cada punto y a los atributos específicos de cada elemento y enlace del sistema.
  - Un conjunto mínimo de equipamiento separado. La mayoría del software y hardware de gestión está incorporado en el equipamiento existente. El software utilizado en realizar tareas de gestión reside en los ordenadores y procesadores de comunicaciones (concentradores, bridges, routers, etc.)
- Los elementos activos de la red envían regularmente información de estado al centro de control de red.

## Elementos de un sistema de gestión de red



NMA = network-management application  
NME = network-management entity Aplic. = Aplicaciones  
Comm. = communications software OS = Operating system



**SEP**  
SECRETARÍA DE  
EDUCACIÓN PÚBLICA



CENTRO NACIONAL  
DE EVALUACIÓN PARA



## **EDITORIAL:** **guiaceneval.mx**



## Configuración de un sistema de gestión de red

- Cada nodo contiene un conjunto de software relacionado con la gestión y llamado **entidad de gestión de red (NME)**
- Cada NME realiza las siguientes tareas:
  - Recoge estadísticas de actividades de comunicaciones y actividades de red
  - Almacena estadísticas localmente
  - Responde a comandos del centro de control de red, como:
    - Enviar estadísticas al centro de control de red
    - Cambiar un parámetro (ej. un temporizador usado en un protocolo)
    - Proporcionar información de estado (ej. valores de parámetros, enlaces activos)
    - Generar tráfico artificial para realizar pruebas.
- Cada nodo con un NME se suele denominar **agente**.

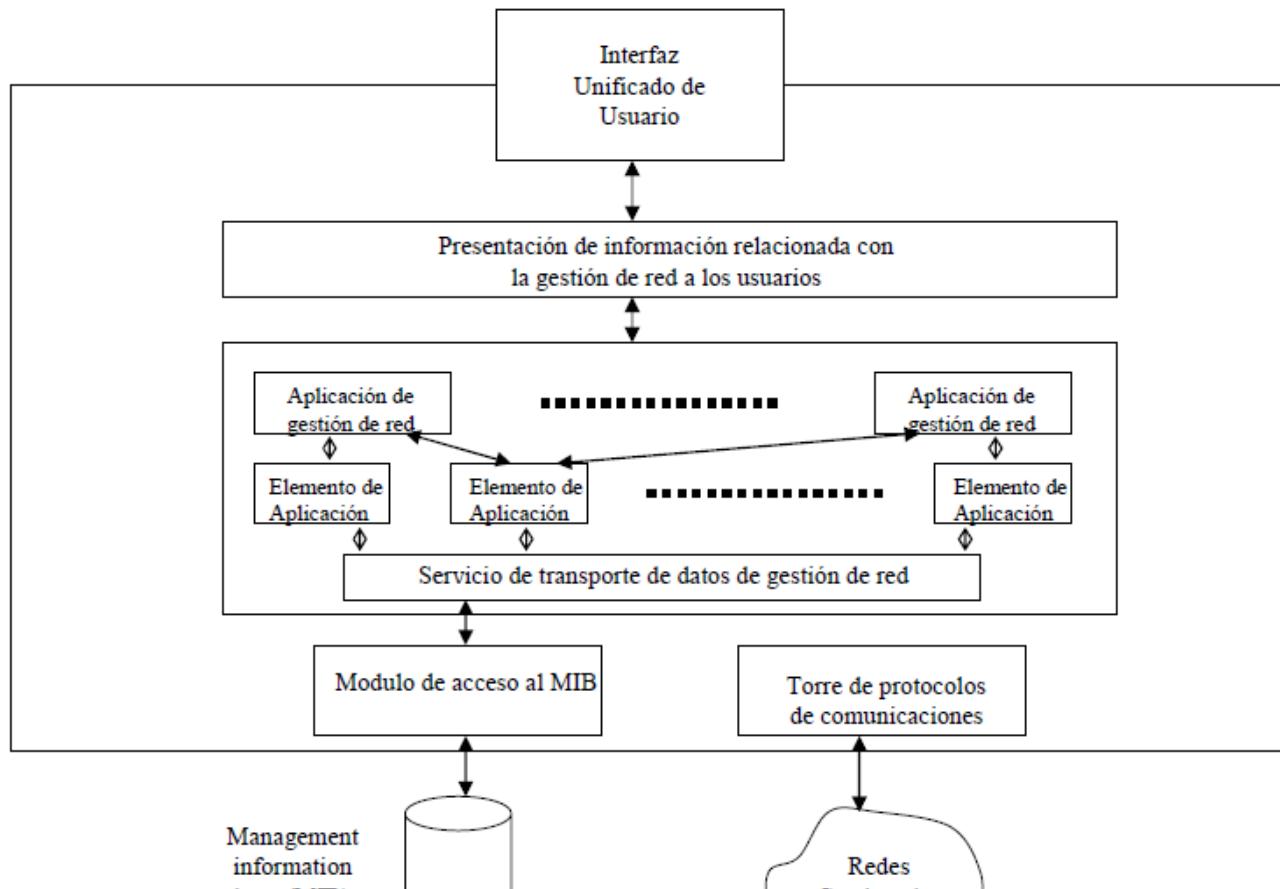
## Configuración de un sistema de gestión de red

- Al menos un nodo de la red es el nodo de gestión de red o **gestor (manager)**. Además del NME, incluye una colección de software denominada la aplicación de gestión de red (NMA).
- El NMA realiza las siguientes tareas:
  - Incluye una interfaz de operador para permitir a usuarios autorizados gestionar la red
  - Responde a comandos del usuario mostrando información y/o enviando comandos a los NME de la red. Esta comunicación se realiza mediante un protocolo de gestión de red a nivel de aplicación de manera similar a cualquier otra aplicación distribuida.
- Normalmente por motivos de seguridad se suelen tener dos o más nodos de gestión de red. Todos menos uno no hacen nada o sólo recogen estadísticas. Si el principal falla, se usa uno de los otros.

# Arquitectura del software de gestión de red

- La funcionalidad del software en un agente o gestor varía mucho dependiendo de la funcionalidad de la plataforma y sus capacidades de gestión.
- En general el software se puede dividir en tres categorías:
  - Software de presentación al usuario
  - Software de gestión de red
  - Software de soporte de gestión de red (gestión de base de datos y comunicaciones)
- La siguiente figura muestra una visión genérica de esta arquitectura software.

## Arquitectura del software de gestión de red



**SEP**

SECRETARÍA DE  
EDUCACIÓN PÚBLICA



CENTRO NACIONAL  
DE EVALUACIÓN PARA  
LA EDUCACIÓN SUPERIOR, A.C.



**EDITORIAL:**  
**guiaceneval.mx**



**Esta guía de estudio, fue vendida por [www.guiaceneval.mx](http://www.guiaceneval.mx) Todos los Derechos Reservados.**

## Software de presentación al usuario

- Proporciona la interacción entre el usuario (gestor) y el software de gestión de red.
- Reside en el gestor de red para monitorizar y controla la red. A veces es útil un software de este tipo en un agente para pruebas, depuración y gestionar algunos parámetros de manera local.
- Una de las claves de este software es proporcionar una interfaz de usuario **unificada**. La interfaz debe ser la misma para cualquier nodo, independientemente del vendedor. Esto permite gestionar una red heterogénea con un mínimo entrenamiento.
- Un peligro es la sobrecarga de información. Una gran cantidad de información está disponible al gestor. Son necesarias herramientas de presentación que organicen, resuman y simplifiquen esta información tanto como sea posible.
- Es preferible información gráfica que textual o tabular.

## Software de gestión de red

- Este software puede ser muy simple, como SNMP o complejo, como OSI. El cuadro central de la figura muestra una arquitectura compleja que refleja la arquitectura de OSI y de sistemas propietarios típicos.
- El software tiene tres niveles:
  - **Aplicaciones:** Proporciona servicios de interés de los usuarios, por ejemplo podrían corresponder con las áreas de OSI: fallos, costes, etc.
  - **Elementos:** Las pocas aplicaciones son soportadas por un número mayor de elementos que implementan funciones más primitivas y más de propósito general, tal como generar alarmas o resumir estadísticas. Son elementos básicos normalmente compartidos por varias aplicaciones. Esta organización sigue los principios tradicionales de diseño modular y reutilización de software.
  - **Servicio de transporte de datos:** Es el protocolo utilizado para intercambiar información de gestión entre gestores y agentes y una interfaz de servicio para los elementos. Esta interfaz proporciona funciones muy primitivas como obtener una información, dar un valor a un parámetro o generar una notificación.



## **Software de soporte de gestión de red**

- Para realizar sus funciones el software de gestión de red necesita acceder a la base de información de gestión local (MIB) y a gestores y agentes remotos.
- La MIB local de un agente contiene información sobre gestión, incluyendo información de la configuración y comportamiento del nodo y parámetros que pueden utilizarse para controlar la operación del nodo.
- La MIB de un gestor contiene la información de su agente y un sumario de la información de los agentes que controla.
- El módulo de acceso ala MIB consiste en un software básico de acceso a ficheros que permite acceder a la MIB. Además puede ser necesario que convierta la información del formato de la MIB local a un formato estandarizado utilizado por el sistema de gestión.
- La comunicación con otros nodos (agentes y gestores) es soportado por la torre de protocolos, como pueden ser TCP/IP u OSI. Es decir, la arquitectura de comunicaciones soporta el protocolo de gestión, que está en el nivel de aplicación.

## **Gestión de red distribuida**

- Al igual que los sistemas centralizados han ido evolucionando hacia sistemas distribuidos, pasando las aplicaciones de los centros de datos a los distintos departamentos con la gestión pasa lo mismo.
- La gestión se distribuye por los mismos motivos que las aplicaciones:
  - Aparición de PC y estaciones de bajo coste y altas prestaciones
  - Proliferación de LANs departamentales
  - Necesidad de control y optimización local de aplicaciones distribuidas
- La gestión distribuida simplemente sustituye el centro de control de red con estaciones de gestión en las LANs de la organización cooperando entre ellas.
- Esto permite a los gestores de los departamentos mantener las redes, sistemas y aplicaciones de sus usuarios locales

## Gestión de red distribuida

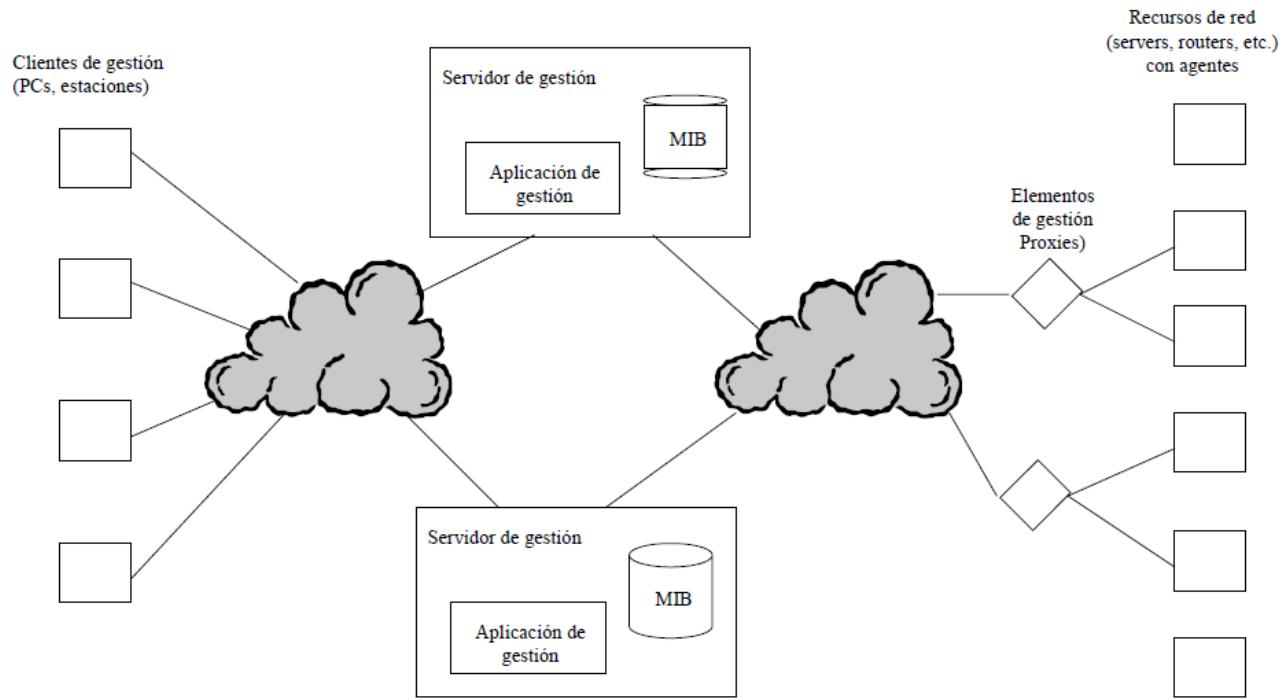
- Para prevenir la anarquía, normalmente se utiliza una arquitectura jerárquica:
  - Las estaciones de gestión distribuidas tienen un acceso limitado a las funciones de monitorización y control, definidas en función de los recursos departamentales que gestiona.
  - Una estación central, con un *backup*, tiene permisos globales para gestionar todos los recursos de la red. También puede interactuar con las estaciones distribuidas y monitorizar y controlar su operación.

## Beneficios de la gestión de red distribuida

- La gestión distribuida mantiene la capacidad de un control centralizado, ofreciendo ventajas adicionales:
  - El tráfico de gestión de red disminuye.
  - Ofrece mayor escalabilidad. Añadir capacidad de gestión consiste simplemente en instalar otra estación de bajo coste en el lugar deseado.
  - El uso de múltiples estaciones de gestión elimina el único punto de fallo que existe en los esquemas centralizados.



# Esquema de un sistema de gestión de red distribuido

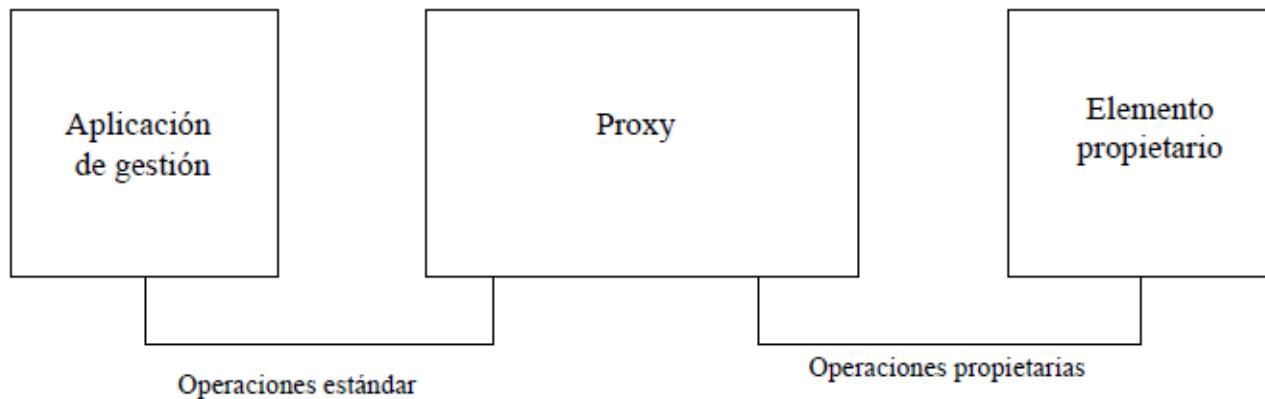


## Proxies

- Hasta ahora suponíamos que todos los elementos de la red tenían un NME con un software de gestión común a toda la red. Actualmente esto no siempre es posible.
- Podemos tener elementos antiguos que no soporten los estándares, pequeños sistemas que no tengan capacidad para soportar el NME completo o elementos como modems, multiplexores, etc. que no soporten software de gestión.
- Para gestionar estos elementos utilizaremos *proxies* que trataremos en profundidad más adelante.

# Proxies

- El *proxy* tiene conectados varios elementos que gestiona.
- Cuando queremos obtener información o controlar uno de estos elementos, el nodo de gestión se comunica con el *proxy* y este traslada la petición al elemento correspondiente y devuelve la información haciendo las conversiones necesarias (por ejemplo para protocolos propietarios, o almacenando la MIB del elemento).



## Lectura 12. Gestión de Redes

# Gestión de Redes

## Sección I: Panorama

### Conceptos claves:

- Qué es la monitorización de redes
- Qué es la gestión de redes
- Lo básico
- Por qué gestión de redes
- Los tres grandes elementos
- Detección de ataques
- Documentación
- Consolidación de la información
- La visión completa

### Detalles de la Gestión de Redes

#### Monitorizamos

- Sistemas y servicios
  - Disponible, alcanzable
- Recursos
  - Planificación de expansión, mantener disponibilidad
- Rendimiento
  - Tiempo de ida y vuelta, tasa máxima de transmisión
- Cambios y configuraciones
  - Documentación, control de versiones, logs



## **Detalles de la Gestión de Redes**

### **Seguimos la pista de**

- **Estadísticas**
  - Para fines de contabilidad
- **Fallos**
  - Detección,
  - Historial de fallos y sus soluciones
- **Los sistemas de gestión de incidencias son buenos para esto**

### **Expectativas**

Una red en operación debe ser monitorizada para:

- Asegurar los SLA proyectados (Acuerdos de Nivel de Servicio)
- Los SLAs dependen de políticas
  - Qué espera la dirección?
  - Qué esperan los usuarios?
  - Qué esperan los clientes?
  - Qué espera el resto de la Internet?
- **Qué se considera bueno? 99.999% de disponibilidad?**  
No hay tal cosa como disponibilidad 100% 

# Expectativas de Disponibilidad

## Qué hace falta para 99.9 %?

30.5 días x 24 horas = 732 horas por mes ( $732 - (732 \times .999)$ ) x 60 = 44 minutos

Sólo 44 minutos de baja por mes!

## Tiene que apagar 1 hora por semana?

$(732 - 4) / 732 \times 100 = 99.4\%$

*Recuerde tomar en cuenta el tiempo de baja planeado, e informe a sus usuarios si está o no incluído en el SLA*

## Cómo se mide la disponibilidad?

En el núcleo (core) ? Extremo a extremo? Desde la Internet?

## Puntos de Referencia

### Qué se considera normal en su red?

Si nunca ha monitorizado su red, tendrá que saber cosas como:

- Carga típica de los enlaces (  Cacti)
- Nivel de variabilidad (jitter) entre dos puntos (  Smokeping)
- Utilización típica de recursos
- Niveles de “ruido” típicos:
  - Escaneos de red
  - Datos descartados
  - Errores reportados y fallos



## **Por qué hacer todo esto?**

### **Saber cuándo se necesita una mejora**

- Su ancho de banda está saturado?
- A dónde va su tráfico?
- Necesita un enlace de más capacidad, u otro proveedor?
- Es demasiado viejo el equipo?

### **Mantener una auditoría de cambios**

- Anotar todos los cambios
- Facilita conocer el origen de los problemas después de cambios y actualizaciones

### **Mantenga un histórico de las operaciones**

- Use un sistema de gestión de incidencias
- Le permite protegerse y saber lo que ha ocurrido

## **Por qué la gestión de redes?**

### **Contabilidad**

- Medir el uso de los recursos
- Cobrar a clientes basado en utilización

### **Saber cuándo hay problemas**

- Entérese antes que los usuarios, sino quedará mal!
- El sistema de gestión puede crear incidencias y notificar al equipo técnico

### **Tendencias**

- Toda esta información sirve para ver las tendencias en la red
- Esto es parte del establecimiento de un punto de referencia, planificación de la capacidad, etc.

## **Los tres “grandes” elementos**

### **Disponibilidad**

- Nagios Servicios, servidores, enrutadores, etc.

### **Fiabilidad**

- Smokeping Retardo, pérdidas, variabilidad

### **Rendimiento**

- Cacti Utilización de enlaces, CPU, memoria, disco, etc.

*Existe cierta coincidencia de funcionalidades entre los tres*



## Detección de ataques

- La utilización de las tendencias y la automatización, permiten determinar cuándo es víctima de un ataque
- Las herramientas le pueden ayudar a mitigar estos ataques:
  - Flujos (netflow) a través de interfaces
  - Saturación de servicios o servidores específicos
  - Fallos en múltiples servicios

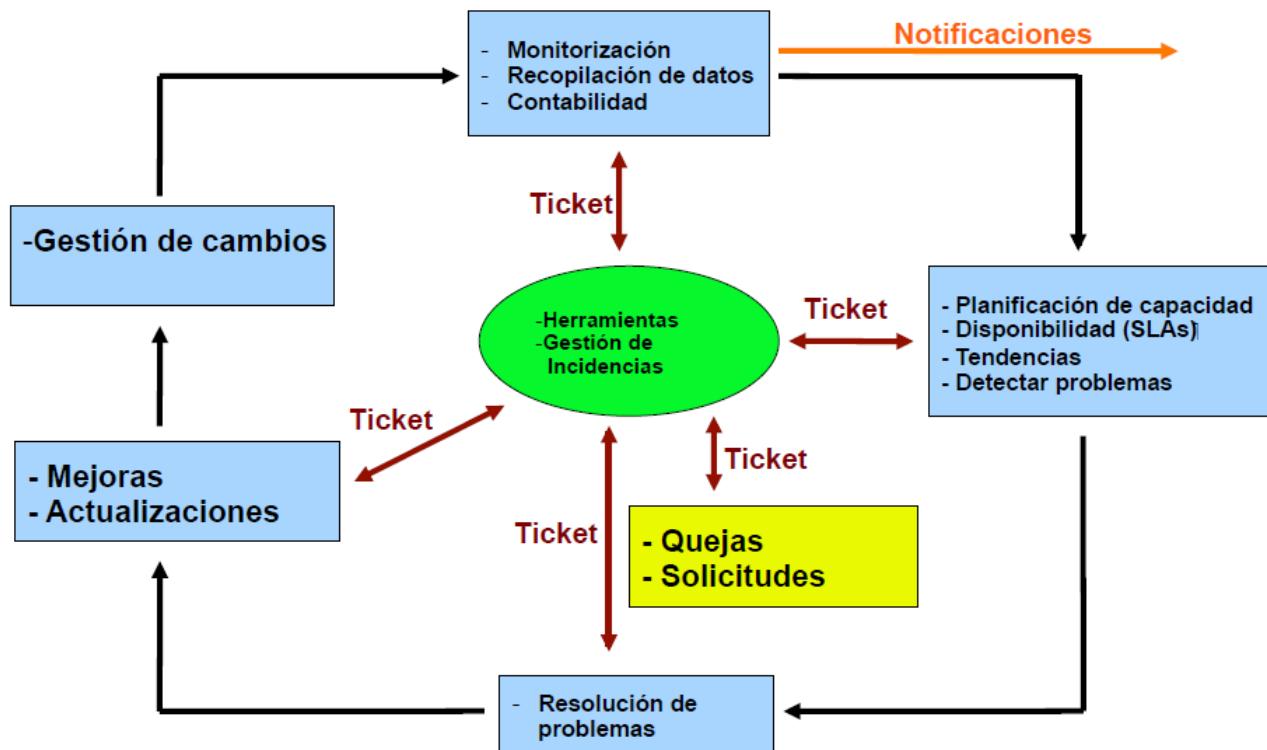
## Consolidación de Datos

**El Centro de Operaciones de la Red (COR, o NOC) es “Donde ocurre todo”**

- Coordinación de tareas
- Estado de la red y los servicios
- Atención de incidencias y quejas
- Donde residen las herramientas (“servidor NOC”)
- Documentación que incluye:
  - Diagramas de red
  - Asignación de puertos en comutadores y enrutadores
  - Descripción de la red
  - Y como veremos mas adelante, mucho más



# Visión General



# Unas pocas soluciones Open Source...

## Rendimiento

- Cricket
- IFPFM
- flowc
- mrtg\*
- NetFlow\*
- NfSen\*
- ntop
- perfSONAR
- pmacct
- RRDtool\*
- SmokePing\*

## Manejo de Incidencias

- RT\*
- Trac\*
- Redmine

## Gestión de Cambios

- Mercurial
- Rancid\* (routers)
- CVS\*
- Subversion\*
- git\*

## Seguridad/(SDI)

- Nessus
- OSSEC
- Prelude
- Samhain
- SNORT
- Untangle

## Registro de Eventos

- swatch\*
- syslog-ng/rsyslog\*
- tenshi\*

## Gestión de Redes

- Big Brother
- Cacti\*
- Hyperic
- Munin
- Nagios\*
- OpenNMS\*
- Observium\*
- Sysmon
- Zabbix

## Documentación

- IPplan
- Netdisco
- Netdot\*
- Rack Table

## Protocolos/Utilidades

- SNMP\*, Perl, ping



## **Sección II: Detalles**

### **Algunos detalles sobre los conceptos :**

- Continuación de la Documentación de Redes
- Herramientas de Diagnóstico
- Herramientas de Monitorizado
- Herramientas de Rendimiento
- Herramientas Activas y Pasivas
- SNMP
- Sistemas de Gestión de Incidentes
- Gestión de configuración y cambios

## **Sección III: Detalles**

### **Algunos detalles acerca de los conceptos claves:**

- Herramientas de Diagnóstico
- Herramientas de Monitorizado
- Herramientas de Rendimiento
- Herramientas Activas y Pasivas
- SNMP
- Sistemas de Gestión de Incidentes
- Gestión de configuración y cambios

## Sistemas de monitorización

### Tres tipos de herramientas

- 1. Diagnóstico** - probar conectividad, comprobar que una ubicación es alcanzable, o que un dispositivo está disponible. Generalmente herramientas activas.
- 2. Monitorización** - ejecución en segundo plano ("demonios" o servicios, que recopilan eventos, pero que también pueden iniciar sus propias verificaciones de estado (usando herramientas de diagnóstico), y anotan el resultado, de manera programada.
- 3. Rendimiento** - Nos dicen cómo la red está manejando los flujos de datos



## **Sistemas y herramientas de monitorización**

### **3. Herramientas de rendimiento**

Mirar cada interfaz del enrutador y conmutadores Dos herramientas populares:

- Netflow/NfSen: <http://nfsen.sourceforge.net/>
- MRTG: <http://oss.oetiker.ch/mrtg/>
- MRTG = “Multi Router Traffic Grapher”

# Sistemas y herramientas de monitorización

## Herramientas activas

- Ping - Probar conectividad hacia un nodo
- Traceroute - Mostrar la ruta de los paquetes
- MTR - Combinación de ping + traceroute
- Colectores de SNMP (polling)

## Herramientas Pasivas

- Monitorización de eventos, SNMP traps, NetFlow

## Herramientas Automáticas

- SmokePing - Recopilar y graficar el retardo en alcanzar nodos y servicios, usando ICMP (Ping) y otros métodos
- MRTG/RRD - Recopilar y graficar la utilización del canal en cada interfaz de un dispositivo

# Sistemas y herramientas de monitorización

## Monitorizado de la red y servicios

- Nagios - Monitor de servidores y servicios
  - Puede monitorizar prácticamente de todo
  - HTTP, SMTP, DNS, Disco, CPU, ...
  - Fácil de escribir nuevas extensiones (plug-ins)
- Solo requiere conocimiento básico de programación para desarrollar nuevas pruebas - Perl, Shell scripts, php, etc...
- Muchas buenas opciones de Fuente Abierta
  - Zabbix, ZenOSS, Hyperic, OpenNMS ...
- Los mecanismos de dependencias son muy útiles



# **Sistemas y herramientas de monitorización**

## **Monitorice sus servicios críticos**

- DNS/Web/Email
  - Radius/LDAP/SQL
  - SSH
- Cómo va a recibir alarmas?**

## **No olvide la gestión de eventos!**

- Cada dispositivo de red (así como servidores Linux y Windows) pueden reportar eventos usando Syslog
- Debe recopilar y monitorizar sus archivos de eventos!
- No hacerlo es uno de los principales errores en la gestión de red

## **Protocolos de Gestión de Red**

### **SNMP – Simple Network Management Protocol**

- Estándar de la industria, cientos de herramientas
- Presente en cualquier elemento de red decente
  - Utilización de canal, errores, CPU, temperatura, ...
- Disco, procesos, ...
- UNIX y Windows también lo implementan

### **SSH y telnet**

- También es posible usar scripts (programas sencillos) para monitorizar remotamente

## Herramientas SNMP

### Conjunto de herramientas Net SNMP

- <http://net-snmp.sourceforge.net/>

## Muy sencillo desarrollar herramientas basadas en estas utilidades

- Recopilar las tablas ARP de enrutadores
- Recopilar las tablas de conmutación de los switches
- Solicitar el estado de un arreglo de discos en RAID.
- Solicitar las temperaturas de servidores, enrutadores, etc.

## Herramientas de estadísticas

### Contabilidad y Análisis de Tráfico

- Cómo está siendo utilizada la red, y qué tanto
- Util para calidad de servicio (QoS), detectar abusos, y facturación
- Protocolo dedicado: NetFlow
- Identificar flujos de tráfico: protocolo, fuente, destino, bytes
- Diferentes herramientas
  - Flowtools, flowc
  - NFSen
  - Muchas más: <http://www.networkuptime.com/tools/netflow/>



## **Gestión de Fallos**

### **Es transitorio el problema?**

- Sobrecarga, falta temporal de recursos

### **Es permanente el problema?**

- Fallo del equipo, línea caída

### **Cómo detectar un problema?**

- Monitorización!
- Quejas

### **Un sistema de incidencias es esencial**

- - Abrir ticket para seguir un evento (ya sea planificado o por fallo)
- - Definir reglas de escalado
  - Quién es responsable de resolver el problema?
  - A quién se le asigna si éste no está disponible?

## **Sistemas de Incidencias**

### **Por qué son importantes?**

- Seguir todos los eventos, fallos y problemas

### **Punto central de comunicación del Help Desk Utilícelo para registrar toda comunicación**

- Tanto interna como externa

### **Eventos originados desde fuera:**

- Quejas de clientes

### **Eventos originados desde dentro:**

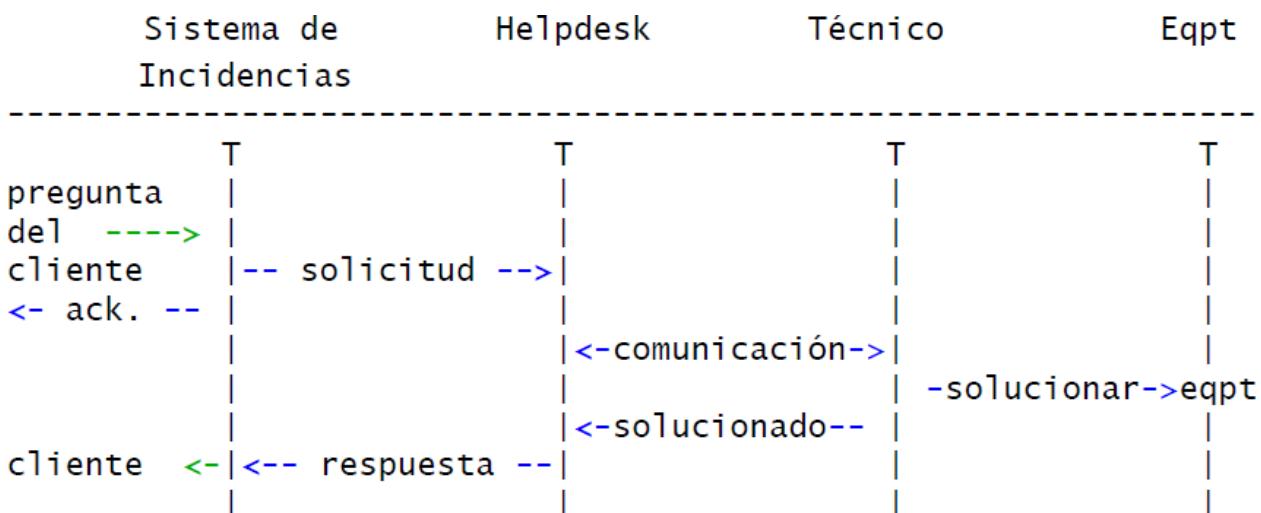
- Salidas de servicio de sistemas
- Actualizaciones o mantenimiento planificados
- Recuerde notificar a los clientes!

## Sistemas de Incidencias

- A cada caso se le asigna un número
  - Cada caso atraviesa un ciclo similar:
    - Nuevo
    - Abierto
    - ...
    - Resuelto
    - Cerrado

# Sistemas de Incidencias

## **Flujo de tareas:**



**SEP**  
SECRETARÍA DE  
EDUCACIÓN PÚBLICA



**EDITORIAL:**  
**guiaceneval.mx**



## Sistemas de incidencias: Ejemplos

### RT (Request Tracker)

- Muy usado mundialmente.
- Un sistema clásico de incidencias que se puede ajustar a cada entidad.
- Un poco difícil de instalar y configurar.
- Puede manejar grandes volúmenes de transacciones

### Trac

- Sistema híbrido que incluye wiki y manejo de proyectos
- Sistema de incidencias inferior a RT, pero funciona bien
- Usado para seguir proyectos de grupo

### Redmine

- Como trac, pero más robusto. Mucho más difícil de instalar

## Sistemas de Detección de Intrusiones de Red (SDI)

Programas que observan los flujos de tráfico y envían alarmas cuando detectan cosas como:

- Nodos infectados o que actúan como fuentes de Spam.

### Algunas herramientas:

- SNORT - Un sistema IDS muy popular: <http://www.snort.org/>
- Prelude - Sistema de Gestión de Información de Seguridad  
<https://dev.prelude-technologies.com/>
- Samhain - SDI Centralizado <http://la-samhna.de/samhain/>
- Nessus - Escáner de vulnerabilidades:  
<http://www.nessus.org/download>  
/

## Gestión y monitorización de configuraciones

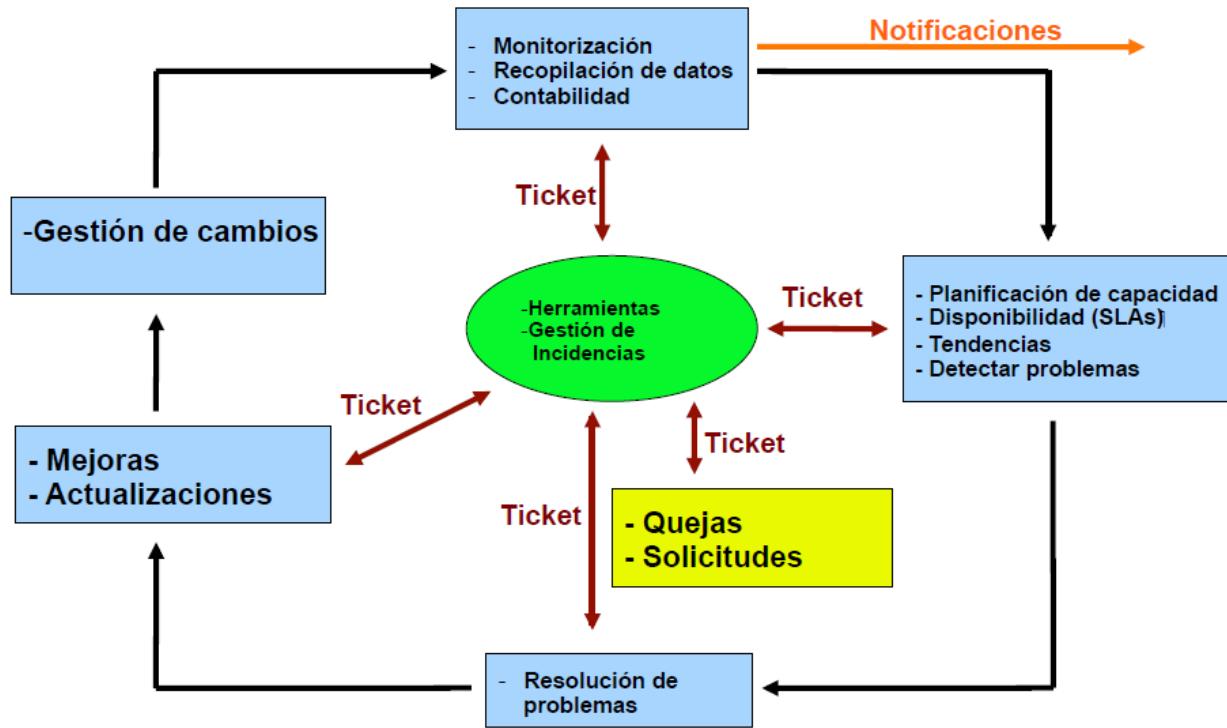
- Registrar cambios en configuraciones de equipos de red usando control de versiones
- Gestión de Inventory (Equipos, IPs, interfaces)
- Usar control de versiones
  - Tan simple como: "cp named.conf named.conf.20070827-01"
- Para archivos de configuración:
  - **CVS, Subversion (SVN)**
  - **Mercurial**
- Para enrutadores:
  - **RANCID**

## Gestión y monitorización de configuraciones

- Se solía usar para código fuente (programas)
- Funciona bien para cualquier configuración en formato texto
  - También para archivos binarios, pero no es posible ver diferencias
- Para equipos de red:
  - **RANCID** (Recopilación y registro automático de configuraciones para Cisco y otros fabricantes)
- Incluido en software de gestión de proyectos como:
  - **Trac**
  - **Redmine**
  - Y muchos otros productos de Wikis. Excelente para documentar la red.



## Revisión de la visión general



## Lectura 13. SISTEMAS OPERATIVOS



# **SISTEMAS OPERATIVOS**

## UNIDAD I:

# COMPONENTES BÁSICOS Y ESTRUCTURA DE LOS SISTEMAS OPERATIVOS

Concepto de un sistema operativo

Es el conjunto de programas que nos permiten utilizar la computadora y que nos sirven para:

- Interfaz con la computadora:
  - Desarrollo de programas
  - Ejecución de programas
  - Acceso a dispositivos de E/S
  - Acceso al sistemas de archivos
  - Protección y seguridad
  - Detección y respuesta de errores
  - Contabilidad
- Gestor de recursos

Componentes de un sistema operativo

1. Núcleo o Kernel: Ejecuta tareas, trabaja con los procesos y las IRQ's.
2. Interprete de comandos: Interpreta instrucciones.
3. Sistema de archivos



## História de los sistemas operativos

- Primera generación (1945-55)
- Segunda generación (1955-65)
- Tercera generación (1965-80)
- Cuarta generación (1980-hoy)

### Primera generación (1945-55)

- Utilidad: máquinas de cálculo.
- Tecnología: Tubos de vacío y paneles.
- Métodos de programación: cables, interruptores y tarjetas perforadas.
- Diseño, construcción, operación, programación y mantenimiento: genios como Aiken, von Newman o Mauchley.

### Segunda generación (1955-65)

- Utilidad: calculo científico e ingeniería.
- Tecnología: transistores y sistemas por lotes, que redujo su tamaño y precio.
- Método de programación: ensamblador y lenguajes de alto nivel (FORTRAN) sobre tarjetas perforadas.
- Fue el paso de procesamiento secuencial a procesamiento por lotes, por ejemplo FMS Y IBSYS.

Tercera generación (1965-80)

- Utilidad: cálculo científico e ingeniería y procesamiento de caracteres.
- Tecnología: circuitos integrados y multiprogramación
- Ejemplos: OS/360, CTSS, MULTICS, UNIX.

Cuarta generación (1980-hoy)

- Tecnología: computadora personal.
- Microprocesadores: 8080, z80, 8086, 286, 386, 486, Pentium, Core 2, Athlon, Alpha, Ultrasparc.
- Logros destacables: GUI, SO de red, SMP, SO distribuidos.

#### Variedad de sistemas operativos

##### Sistemas operativos de mainframe

Son las computadoras del tamaño de un cuarto completo que aún se encuentran en los principales centros de datos corporativos, es decir, las mainframes también están volviendo a figurar en el ámbito computacional como servidores web de alto rendimiento, servidores para sitios de comercio electrónico a gran escala y servidores para transacciones de negocio a negocio.

Los sistemas operativos para las mainframes están profundamente orientados hacia el procesamiento de muchos trabajos a la vez, de los cuales la mayor parte requiere muchas operaciones de E/S. Por lo general ofrecen tres tipos de servicios: procesamiento por lotes, procesamiento de transacciones y tiempo compartido.



Un ejemplo de sistema operativo de mainframe es el OS/390, un descendiente del OS/360. Sin embargo, los sistemas operativos de mainframes están siendo reemplazados gradualmente por variantes de UNIX, como Linux, sistemas operativos para servidores.

Se ejecutan en servidores, que son computadoras personales muy grandes, estaciones de trabajo o incluso mainframes. Dan servicio a varios usuarios a la vez a través de una red y les permiten compartir los recursos de hardware y de software. Los servidores pueden proporcionar servicio de impresión, de archivos o web. Los proveedores de Internet operan muchos equipos servidores para dar soporte a sus clientes y los sitios Web utilizan servidores para almacenar las páginas Web y hacerse cargo de las peticiones entrantes. Algunos sistemas operativos de servidores comunes son Solaris, Free BSD, Linux y Windows Server 200x.

#### Sistemas operativos de multiprocesadores

Una manera cada vez más común de obtener poder de cómputo de las grandes ligas es conectar varias CPU en un solo sistema. Dependiendo de la exactitud con la que se conecten y de lo que se comparta, estos sistemas se conocen como computadoras en paralelo, multicomputadoras o multiprocesadores. Muchos sistemas operativos populares (incluyendo Windows y Linux) se ejecutan en multiprocesadores.

## Sistemas operativos de computadoras personales

Su trabajo es proporcionar buen soporte para un solo usuario. Se utilizan ampliamente para el procesamiento de texto, las hojas de cálculo y el acceso a Internet. Algunos ejemplos comunes son Linux, FreeBSD, Windows Vista y el sistema operativo Macintosh.

## Sistemas operativos de computadoras de bolsillo

Una computadora de bolsillo o PDA (*Personal Digital Assistant*, Asistente personal digital) es una computadora que cabe en los bolsillos y realiza una pequeña variedad de funciones, como libreta de direcciones electrónica y bloc de notas. Además, hay muchos teléfonos celulares muy similares a los PDA's, con la excepción de su teclado y pantalla.

Una de las principales diferencias entre los dispositivos de bolsillo y las PC's es que los primeros no tienen discos duros de varios cientos de gigabytes, lo cual cambia rápidamente. Dos de los sistemas operativos más populares para los dispositivos de bolsillo son Symbian OS y Palm OS.

## Sistemas operativos integrados

Los sistemas integrados (*embedded*), que también se conocen como incrustados o embebidos, operan en las computadoras que controlan dispositivos que no se consideran generalmente como computadoras, ya que no aceptan software instalado por el usuario. Algunos ejemplos comunes son los hornos de microondas, las televisiones, los autos, los grabadores de DVD's, los teléfonos celulares y los reproductores de MP3. Los sistemas como QNX y VxWorks son populares en este dominio.



## Sistemas operativos de nodos sensores

Cada nodo sensor es una verdadera computadora, con una CPU, RAM, ROM y uno o más sensores ambientales. Ejecuta un sistema operativo pequeño pero real, por lo general manejador de eventos, que responde a los eventos externos o realiza mediciones en forma periódica con base en un reloj interno. El sistema operativo tiene que ser pequeño y simple debido a que los nodos tienen poca RAM y el tiempo de vida de las baterías es una cuestión importante.

Estas redes de sensores se utilizan para proteger los perímetros de los edificios, resguardar las fronteras nacionales, detectar incendios en bosques, medir la temperatura y la precipitación para el pronóstico del tiempo, deducir información acerca del movimiento de los enemigos en los campos de batalla y mucho más. TinyOS es un sistema operativo bien conocido para un nodo sensor.

## Sistemas operativos en tiempo real

Estos sistemas se caracterizan por tener el tiempo como un parámetro clave. Muchos de estos sistemas se encuentran en el control de procesos industriales, en aeronáutica, en la milicia y en áreas de aplicación similares. Estos sistemas deben proveer garantías absolutas de que cierta acción ocurrirá en un instante determinado.

Como en los sistemas en tiempo real es crucial cumplir con tiempos predeterminados para realizar una acción, algunas veces el sistema operativo es simplemente una biblioteca enlazada con los programas de aplicación, en donde todo está acoplado en forma estrecha y no hay protección entre cada una de las partes del sistema. Un ejemplo de este tipo de sistema en tiempo real es e-Cos.

## Sistemas operativos de tarjetas inteligentes

Los sistemas operativos más pequeños operan en las tarjetas inteligentes, que son dispositivos del tamaño de una tarjeta de crédito que contienen un chip de CPU. Tienen varias severas restricciones de poder de procesamiento y memoria. Algunos sistemas de este tipo pueden realizar una sola función, como pagos electrónicos; otros pueden llevar a cabo varias funciones en la misma tarjeta inteligente. A menudo éstos son sistemas propietarios.

## Conceptos básicos de sistemas operativos

### Procesos

Un proceso es en esencia un programa en ejecución. Cada proceso tiene asociado un espacio de direcciones, una lista de ubicaciones de memoria que va desde algún mínimo (generalmente 0) hasta cierto valor máximo, donde el proceso puede leer y escribir información. Cuando un proceso se suspende en forma temporal, todos estos apuntadores deben guardarse de manera que una llamada a read que se ejecute después de reiniciar el proceso lea los datos apropiados. En muchos sistemas operativos, toda la información acerca de cada proceso (además del contenido de su propio espacio de direcciones) se almacena en una tabla del sistema operativo, conocida como la tabla de procesos, la cual es un arreglo (o lista enlazada) de estructuras, una para cada proceso que se encuentre actualmente en existencia.



Así, un proceso (suspendido) consiste en su espacio de direcciones, que se conoce comúnmente como imagen de núcleo (en honor de las memorias de núcleo magnético utilizadas antaño) y su entrada en la tabla de procesos, que guarda el contenido de sus registros y muchos otros elementos necesarios para reiniciar el proceso más adelante.

Las llamadas al sistema de administración de procesos clave son las que se encargan de la creación y la terminación de los procesos. Los procesos relacionados que cooperan para realizar un cierto trabajo a menudo necesitan comunicarse entre sí y sincronizar sus actividades. A esta comunicación se le conoce como comunicación entre procesos.

Cada persona autorizada para utilizar un sistema recibe una UID (*User Identification*, Identificación de usuario) que el administrador del sistema le asigna. Cada proceso iniciado tiene el UID de la persona que lo inició. Un proceso hijo tiene el mismo UID que su padre. Los usuarios pueden ser miembros de grupos, cada uno de los cuales tiene una GID (*Group Identification*, Identificación de grupo).

#### Espacios de direcciones

Dado que la administración del espacio de direcciones de los procesos está relacionada con la memoria, es una actividad de igual importancia. Por lo general, cada proceso tiene cierto conjunto de direcciones que puede utilizar, que generalmente van desde 0 hasta cierto valor máximo. En el caso más simple, la máxima cantidad de espacio de direcciones que tiene un proceso es menor que la memoria principal. De esta forma, un proceso puede llenar su espacio de direcciones y aun así habrá suficiente espacio en la memoria principal para contener todo lo necesario.

En muchas computadoras las direcciones son de 32 o 64 bits, con lo cual se obtiene un espacio de direcciones de 232 o 264 bytes, respectivamente. En esencia, el sistema operativo crea la abstracción de un espacio de direcciones como el conjunto de direcciones al que puede hacer referencia un proceso.

El espacio de direcciones se desacopla de la memoria física de la máquina, pudiendo ser mayor o menor que la memoria física. La administración de los espacios de direcciones y la memoria física forman una parte importante de lo que hace un sistema operativo.

#### Archivos

Un archivo es una agrupación de información que se guarda en algún dispositivo no volátil. Desde la perspectiva del usuario, es la unidad mínima de almacenamiento que el sistema le provee.

Se requieren las llamadas al sistema para crear los archivos, eliminarlos, leer y escribir en ellos. Antes de poder leer un archivo, debe localizarse en el disco para abrirse y una vez que se ha leído información del archivo debe cerrarse, por lo que se proporcionan llamadas. Los archivos constan de las siguientes propiedades:

- **Atributos:**

- Nombre: permite identificar el archivo a los usuarios.
- Identificador: el SO le asigna este símbolo que lo identifica (número) que no hace único.
- Tipo: puede ser un programa ejecutable, archivo de datos, etc.
- Ubicación: puntero al dispositivo y lugar donde reside el archivo.
- Tamaño: cantidad de información que contiene.



- Protección: información de control para el acceso al archivo.
- Información de conteo: contiene la fecha de creación, ultimo acceso, etc.
- Operaciones:
  - Crear y abrir: se crea o se abre un archivo.
  - Escribir: permite escribir en el archivo creado.
  - Leer: permite leer el contenido del archivo.
  - Reposicionar dentro de un archivo: logra acceder a cualquier parte del archivo.
  - Eliminar: se destruye el archivo al nivel del sistema de archivos.
  - Truncar: elimina la información que está dentro del archivo, pero sin eliminar el archivo como tal.

Tener un archivo abierto para el sistema implica mantener una estructura que tengan por lo menos:

- Puntero(file pointer)
- Ubicación

En el FCB (File Control Block) se encuentra toda la información de los archivos y los punteros. Así mismo el LOCK es el sistema que provee acceso único a un archivo por parte de los procesos.

#### Entrada/salida

Cada sistema operativo tiene un subsistema de E/S para administrar sus dispositivos de E/S. Parte del software de E/S es independiente de los dispositivos, es decir, se aplica a muchos o a todos los dispositivos de E/S por igual. Otras partes del software, como los drivers de dispositivos, son específicas para ciertos dispositivos de E/S. Existen muchos tipos de dispositivos de entrada y de salida, incluyendo teclados, monitores, impresoras, etcétera. Es la responsabilidad del sistema operativo administrar estos dispositivos.

## Protección

Las computadoras contienen grandes cantidades de información que los usuarios comúnmente desean proteger y mantener de manera confidencial. Esta información puede incluir mensajes de correo electrónico, planes de negocios, declaraciones fiscales y mucho más. Es responsabilidad del sistema operativo administrar la seguridad del sistema de manera que los archivos, por ejemplo, sólo sean accesibles para los usuarios autorizados. Además de la protección de archivos, existen muchas otras cuestiones de seguridad. Una de ellas es proteger el sistema de los intrusos no deseados, tanto humanos como no humanos (por ejemplo, virus).

## El Shell

Sirve como un buen ejemplo de la forma en que se pueden utilizar las llamadas al sistema. También es la interfaz principal entre un usuario sentado en su terminal y el sistema operativo, a menos que el usuario esté usando una interfaz gráfica de usuario. Existen muchos Shell, incluyendo *sh*, *csh*, *ksh* y *bash*. Todos ellos soportan la funcionalidad antes descrita, que se deriva del Shell original (*sh*). Cuando cualquier usuario inicia sesión, se inicia un Shell. El Shell tiene la terminal como entrada estándar y salida estándar. Empieza por escribir el indicador de comandos (prompt), un carácter tal como un signo de dólar, que indica al usuario que el Shell está esperando aceptar un comando.

Actualmente, muchas computadoras personales utilizan una GUI. De hecho, la GUI es sólo un programa que se ejecuta encima del sistema operativo, como un Shell. En los sistemas Linux, este hecho se hace obvio debido a que el usuario tiene una selección de (por lo menos) dos GUI's: Gnome y KDE o ninguna (se utiliza una ventana de terminal en X11).



En Windows también es posible reemplazar el escritorio estándar de la GUI (*Windows Explorer*) con un programa distinto, para lo cual se modifican ciertos valores en el registro, aunque pocas personas hacen esto.

## Estructura del sistema operativo

Un sistema operativo se puede estructurar respecto a:

- Capas
  - Capa 0: proporciona la multiprogramación básica de la CPU, es decir que los procesos cuando ocurren las interrupciones o expiran los cronómetros. Dichos sistemas constan de procesos secuenciales, estos se pueden programar sin importar que varios procesos se estén ejecutando en el mismo procesador.
  - Capa 1: aquí se administra la memoria, al mismo tiempo se asignaba el espacio de memoria principal para los diversos procesos y depósitos de palabras de 512k en el cual se utilizaba para almacenar partes de los procesos, en este caso las páginas.
  - Capa 2: comunicación entre procesos y la consola del usuario.
  - Capa 3: flujo de información a través de hardware, E/S.
  - Capa 4: soporta aplicaciones de los usuarios.
  - Capa 5: proceso operador del sistema.
- Niveles
  - Nivel 1:
    - sincronización entre procesos.
    - comutación de la CPU.
    - gestión de interrupciones.
    - arranque inicial.
  - Nivel 2:
    - Gestión de memoria: asigna la memoria entre procesos.

- Asignación y liberación de memoria.
  - Control, violación de acceso.
  - Nivel 3:
    - Nivel superior de gestión de procesos.
  - Nivel 4:
    - Administra hardware: creación de procesos E/S, asigna y libera, planifica E/S.
  - Nivel 5:
    - Control de archivos.
- 
- Clasificación
    - Por estructura
      - Monolíticos: solo una unidad.
      - Estructurada: agregamos componente adicional.
    - Por usuario
      - Monousuario: solo un usuario.
      - Multiusuario: existen dos accesos por un solo canal.
    - Por tareas
      - Monotareas: solo soporta una acción.
      - Multitareas: realiza muchas acciones al mismo tiempo.
    - Por procesos
      - Uniprocesos: solo se ejecuta un proceso.
      - Multiprocesos:
        - ✓ Simétricos: llegan y asignan a quien esté disponible.
        - ✓ Asimétrico: el SO selecciona a uno de los procesos el cual jugara el papel de procesador maestro.



## UNIDAD II

# MÉTODOS DE COMUNICACIÓN ENTRE PROCESOS, Y ALGORITMOS CLÁSICOS PARA LA CALENDARIZACIÓN

## MÉTODOS DE COMUNICACIÓN ENTRE PROCESOS

Los procesos pueden ser cooperantes o independientes, en el primer caso se entiende que los procesos interactúan entre sí y pertenecen a una misma aplicación. En el caso de procesos independientes en general se debe a que no interactúan y un proceso no requiere información de otros o bien porque son procesos que pertenecen a distintos usuarios.

### Regiones críticas

Para solucionar las condiciones de competencia se implementó un modelo para prohibir que dos procesos accedan al mismo recurso. El modelo en cuestión se denomina exclusión mutua. La parte del programa en la que se tiene acceso a la memoria compartida se denomina región crítica o sección crítica. Necesitamos que se cumplan cuatro condiciones para tener una buena solución:

1. Dos procesos no pueden estar al mismo tiempo dentro de sus regiones críticas.
2. No pueden hacerse suposiciones sobre las velocidades ni el número de las CPU's.
3. Ningún proceso que esté ejecutando afuera de su región crítica puede bloquear a otros procesos.
4. Ningún proceso deberá de tener que esperar de manera indefinida para entrar en su región crítica.

## [Exclusión mutua con espera ocupada](#)

Las soluciones con espera ocupada funcionan de la siguiente manera, cuando un proceso intenta ingresar a su región crítica, verifica si está permitida la entrada. Si no, el proceso se queda esperando hasta obtener el permiso.

## Inhabilitación de interrupciones

El método más simple para evitar las condiciones de competencia es hacer que cada proceso desactive todas sus interrupciones antes de entrar a su sección crítica y las active una vez que salió de la misma. Este modelo como se puede observar, éste modelo tiene una gran problema y es que si se produce una falla mientras que el proceso está en la región crítica no se puede salir de la misma y el sistema operativo no recuperaría el control.

## [Variables de bloqueo](#)

En éste caso se genera una variable la cual puede tener dos valores o bien 0 (no hay ningún proceso en su sección crítica) o bien 1 (indicando que la sección crítica está ocupada) entonces cada proceso antes de ingresar a la sección crítica verifica el estado de la variable de cerradura y en caso de que la misma este en 0, le cambia el valor e ingresa a la misma y en caso de que la misma sea 1 el proceso se queda verificando el estado de la misma hasta que el mismo sea 0. El problema aquí se presenta si dos procesos verifican al mismo tiempo que la variable cerradura está en 0 e ingresan a la región crítica.

## [Alternancia estricta](#)

El algoritmo de alternancia estricta no bloquea el ingreso a la región crítica cuando otro proceso se está ejecutando.



El problema de ésta solución es que cuando un proceso no está en la sección crítica igualmente tiene bloqueado el acceso a la misma y por lo tanto no permite que otro proceso que requiera ingresar a la misma logre hacerlo.

#### Instrucción TSL

Esta solución requiere ayuda del hardware y es debido a que en general las computadoras diseñadas para tener más de un procesador tienen una instrucción TLS RX, bloqueo que funciona: lee el contenido de la palabra de memoria bloqueo, lo coloca en el registro RX y luego guarda un valor distinto de cero en la dirección de memoria bloqueo. La CPU que ejecuta la instrucción TSL cierra el bus de memoria para impedir que otras CPU's tengan acceso a la memoria antes de que termine.

#### Activar y desactivar

El modelo de espera acotada tienen el inconveniente que se desperdicia tiempo de procesador. Este enfoque no sólo desperdicia tiempo de CPU, si no que puede tener efectos inesperados. Consideramos una computadora con dos procesos; A que es prioritario, y B, que no lo es. Las reglas de calendarización son de tal forma que A se ejecuta siempre que está en estado listo. En cierto momento, cuando B está en región crítica, A queda listo para ejecutarse. A inicia una espera activa, pero dado que B nunca se calendariza mientras A se está ejecutando, B nunca tendrá oportunidad de salir de su región crítica, y A seguirá dando vueltas en forma indefinida.

#### El problema del productor y el consumidor

El problema del productor y el consumidor describe el hecho de que cuando hay dos o más procesos interactuando a través de un buffer común habiendo procesos que ponen información o datos y otros que los sacan se pueden llegar a dar condiciones en las cuales los procesos que ingresan los datos no puedan hacerlo

debido a que el buffer ya se encuentra lleno y para el caso de los que sacan los datos del buffer intenten sacar datos cuando ya no hay nada que sacar. Para evitar estas condiciones se desarrollaron métodos de comunicación/sincronización entre procesos en los cuales se impide que esto suceda haciendo que el proceso productor "duerma" si el buffer está lleno y una vez que exista espacio el proceso "consumidor" despierte al productor para que siga generando o viceversa.

### Semáforos

Un semáforo es una variable especial que constituye el método clásico para restringir o permitir el acceso a recursos compartidos en un entorno de multiprocесamiento (en el que se ejecutarán varios procesos concurrentemente). Los semáforos se emplean para permitir el acceso a diferentes partes de programas (llamados secciones críticas) donde se manipulan variables o recursos que deben ser accedidos de forma especial. Según el valor con que son inicializados se permiten a más o menos procesos utilizar el recurso de forma simultánea.

Un tipo simple de semáforo es el binario, que puede tomar solamente los valores 0 y 1. Se inicializan en 1 y son usados cuando sólo un proceso puede acceder a un recurso a la vez. Son esencialmente lo mismo que los mutex. Cuando el recurso está disponible, un proceso accede y decremente el valor del semáforo con la operación P. El valor queda entonces en 0, lo que hace que si otro proceso intenta decrementarlo tenga que esperar. Cuando el proceso que decrementó el semáforo realiza una operación V, algún proceso que estaba esperando comienza a utilizar el recurso. Los semáforos resuelven el problema del despertar perdido. Es indispensable que se implementen de manera indivisible. El procedimiento normal es implementar Down y up como llamadas al sistema y que el sistema operativo inhabilite en forma breve todas las interrupciones, mientras está probando y actualizando el semáforo, así como poniendo el proceso a dormir, si es necesario.



Dado que estas acciones sólo requieren unas cuantas instrucciones, la inhabilitación de las interrupciones no es perjudicial. Si se están utilizando múltiples CPU's, cada semáforo deberá protegerse con una variable de bloqueo, utilizándose la instrucción TSL para garantizar que sólo una CPU a la vez examine el semáforo. Debe entender que el uso de TSL para evitar que varias CPU's tengan acceso simultáneo al semáforo es muy distinto de la espera activa del productor o el consumidor para que el otro vacíe o llene el búfer.

#### Mutexes

Los algoritmos de exclusión mutua (comúnmente abreviad como mutex por mutual exclusión) se usan en programación concurrente para evitar el ingreso a sus secciones críticas por más de un proceso a la vez. La sección crítica es el fragmento de código donde puede modificarse un recurso compartido.

La mayor parte de estos recursos son las señales, contadores, colas y otros datos que se emplean en la comunicación entre el código que se ejecuta cuando se da servicio a una interrupción y el código que se ejecuta el resto del tiempo. Se trata de un problema de vital importancia porque, si no se toman las precauciones debidas, una interrupción puede ocurrir entre dos instrucciones cualesquiera del código normal y esto puede provocar graves fallos.

La técnica que se emplea por lo común para conseguir la exclusión mutua es inhabilitar las interrupciones durante el conjunto de instrucciones más pequeño que impedirá la corrupción de la estructura compartida (la sección crítica). Esto impide que el código de la interrupción se ejecute en mitad de la sección crítica. En un sistema multiprocesador de memoria compartida, se usa la operación indivisible test-and-set sobre una bandera, para esperar hasta que el otro procesador la despeje. La operación test-and-set realiza ambas operaciones sin liberar el bus de memoria a otro procesador. Así, cuando el código deja la sección crítica, se despeja la bandera. Esto se conoce como spin lock o espera activa.

Algunos sistemas tienen instrucciones multioperación indivisibles similares a las anteriormente descritas para manipular las listas enlazadas que se utilizan para las colas de eventos y otras estructuras de datos que los sistemas operativos usan comúnmente.

## Monitores

Un monitor es una colección de procedimientos, variables y estructuras de datos que se agrupan en un tipo especial de módulo o paquete. Los procesos pueden invocar a los procedimientos de un monitor cuando lo deseen, pero no pueden acceder en forma directa a sus estructuras de datos internas desde procedimientos declarados fuera de dicho monitor.

Los monitores tienen una prioridad importante que los hace útiles para lograr exclusión mutua: sólo un proceso puede estar activo en un monitor a la vez. Los monitores son una construcción de lenguaje de programación, así que el compilador sabe que son especiales y puede manejar las llamadas a los procedimientos de monitor, de manera distinta como maneja otras llamadas a procedimientos. Por lo regular, cuando un proceso llama a un proceso de monitor, las primeras instrucciones de éste verifican si algún otro proceso está activo dentro del monitor. Si es así, el proceso invocador se suspenderá hasta que el otro proceso haya salido del monitor. Si ningún otro proceso lo está usando, el proceso invocador puede entrar.

El compilador debe implementar la exclusión mutua en los ingresos a un monitor, pero es posible utilizar un mutex o un semáforo binario. Puesto que el compilador, no el programador, quien “tramita” la exclusión mutua, es mucho menos probable que algo salga mal.



En todo caso, quien escribe el monitor no tiene que saber la manera en que el compilador maneje la exclusión mutua. Basta saber que convirtiendo todas las regiones críticas en procedimiento de monitor, dos procesos nunca podrán ejecutar sus regiones críticas al mismo tiempo.

## Barreras

Algunas aplicaciones se dividen en fases y tienen la regla que ningún proceso puede pasar a la siguiente fase antes de que todos los procesos esté listos para hacerlo. Este comportamiento puede lograrse colocando una barrera al final de cada fase. Cuando un proceso llega a la barrera. Se bloquea hasta que todos los procesos han llegado a ella.

## DIFERENCIA ENTRE PROCESOS Y SUBPROCESOS ASÍ COMO SUS CARACTERÍSTICAS

### Procesos

Un proceso es un programa en ejecución. Un proceso simple tiene un hilo de ejecución, por el momento dejemos esta última definición como un concepto, luego se verá en más detalle el concepto de hilo. Una vez definido qué es un proceso nos podríamos preguntar cuál es la diferencia entre un programa y un proceso, y básicamente la diferencia es que un proceso es una actividad de cierto tipo que contiene un programa, entradas salidas y estados.

### Estados de los procesos

Un proceso puede estar en cualquiera de los siguientes tres estados: listo, en ejecución y bloqueado.

Los procesos en el estado listo son los que pueden pasar a estado de ejecución si el planificador los selecciona. Los procesos en el estado ejecución son los que se están ejecutando en el procesador en ese momento dado. Los procesos que se encuentran en estado bloqueado están esperando la respuesta de algún otro proceso para poder continuar con su ejecución. Por ejemplo operación de E/S.

#### El modelo de procesos

En este modelo. Todo el software ejecutable de una computadora, que a veces incluye al sistema operativo, se organiza en varios procesos secuenciales, o simplemente procesos. Un proceso no es más que un programa en ejecución, e incluye los valores que tienen el contador del programa, los riesgos y las variables. Cada proceso tiene su CPU virtual, claro que en realidad la verdadera CPU cambia en forma continua de un proceso a otro, que trata de comprender en que la CPU cambia de un programa a otro.

Con la CPU comutando entre los procesos, la rapidez con la que un proceso efectúa sus operaciones no será uniforme y es probable que ni siquiera sea reproducible si los mismos procesos se ejecutan a la vez. Los procesos no deben programarse con base en supuestos acerca de los tiempos.

#### Creación de procesos

Los sistemas operativos requieren alguna forma de comprobar que existan todos los procesos necesarios. En los sistemas de propósito general hace falta algún mecanismo para crear y terminar procesos según donde se necesite durante la operación. Hay cuatro sucesos principales que causan la creación de procesos:

1. Inicialización del sistema.
2. Ejecución de una llamada al sistema para crear procesos por parte de un proceso en ejecución.



3. Solicitud de un usuario para crear un proceso.
4. Inicio de un trabajo por lotes.

Cuando se arranca un sistema operativo, por lo regular se crean varios procesos. Algunos son de primer plano, procesos que interactúan con usuarios y que trabajan para ellos. Otros son procesos de segundo plano que no están asociados con un usuario en particular, sino que tiene una función específica. Podría diseñarse un proceso de segundo plano que acepte el correo electrónico entrante; este proceso quedaría inactivo casi todo el día pero entraría en acción repentinamente si llega algún mensaje de correo electrónico. Podría diseñarse otro proceso para aceptar las solicitudes de página web alojadas en esa máquina, y se activaría cuando llegar una solicitud para atenderla. Los procesos que permanecen en segundo plano para encargarse de alguna actividad, como correo electrónico, páginas Web, noticias, impresiones etcétera, se llaman demonios (daemons).

#### Terminación de procesos

Los procesos se ejecutan ya creados y realizan la labor que se les encomendó. Nada es eterno y los procesos no son la excepción. Tarde o temprano el proceso nuevo terminará, por lo regular debido a una de las siguientes condiciones:

1. Terminación normal (voluntaria).
2. Terminación por error (voluntaria).
3. Error fatal (involuntaria).
4. Terminado por otro proceso (involuntaria).

La mayoría de los procesos terminan porque ya realizó su trabajo. Una vez que un compilador ha compilado el programa que se le alimentó, ejecuta una llamada para iniciar al sistema operativo que ya terminó.

## [Jerarquía de procesos](#)

Un proceso termina cuando ejecuta su última instrucción, sin embargo existen circunstancias en que se requiere terminarlo en forma forzada. Un proceso termina a otro mediante una instrucción del tipo kill Id. La operación kill es usualmente invocada solamente por un proceso padre para culminar la ejecución de un hijo. Como la instrucción requiere la identificación del proceso a ser terminado, la instrucción fork da como retorno esta información (Id = fork L). Existen numerosas razones por las cuales un padre puede detener la ejecución de un hijo.

En muchos sistemas operativos se establece la condición de que los procesos hijos no pueden continuar la ejecución si el padre ha sido finalizado. Un proceso que no termina su ejecución durante todo el tiempo en que el sistema operativo está funcionando se dice que es estático. Un proceso que finaliza se dice que es dinámico.

Si un sistema operativo consiste de solo un número limitado de procesos estáticos entonces su correspondiente grafo de procesos será también estático. En caso diferente será dinámico. La estructura estática solo está presente en sistemas operativos muy simples.

## [Estados de procesos](#)

En el modelo de procesos todo el Software ejecutable, a menudo incluyendo el propio sistema de operación, se organiza como procesos secuenciales. Aparentemente cada proceso tiene su propio procesador central, pero en realidad este cambia de uno a otro de acuerdo con el concepto de multiprogramación (seudoparalelismo).



Los procesos son totalmente aleatorios en el tiempo y el comportamiento de un conjunto de ellos dependerá de las condiciones en un instante dado. Esto implica que los programas no pueden ser elaborados asumiendo lo que pasará en el futuro cuando se están procesando. Un proceso puede tener diferentes estados durante su existencia. El número de estados dependerá del diseño del sistema operativo, pero al menos hay tres que siempre estarán presentes:

1. En ejecución: El proceso está en posesión del CPU en ese instante.
2. Listo: El proceso está en condiciones de ejecutar, pero está detenido temporalmente para permitir a otro proceso la ejecución.
3. Bloqueado: El proceso está esperando hasta que ocurra un evento externo (por ejemplo, una E/S).

Desde el punto de vista lógico, los primeros dos estados son similares. En ambos casos, el proceso está dispuesto a ejecutarse, sólo que en el segundo por el momento no hay CPU disponible para él. El tercer estado es diferente, el proceso no puede ejecutarse, aunque la CPU no tenga nada más que hacer.

#### [Implementación de procesos](#)

La implementación del modelo de procesos se logra debido a que el sistema operativo almacena en una tabla denominada tabla de control de procesos información relativa a cada proceso que se está ejecutando en el procesador. Cada línea de esta tabla representa a un proceso. La información que se almacena es la siguiente:

- 1) Identificación del proceso.
- 2) Identificación del proceso padre.
- 3) Información sobre el usuario y grupo.
- 4) Estado del procesador.
- 5) Información de control de proceso

Información del planificador. Segmentos de memoria asignados. 5.3) Recursos asignados.

#### Subprocesos

En los sistemas operativos tradicionales cada proceso tiene un espacio de direcciones y un solo subproceso de control. Abundan las simulaciones en las que es deseable tener varios subprocesos de control en el mismo espacio de direcciones, operando de forma seudoparalela, como si fueran procesos individuales.

#### Modelo de subprocesos

Un proceso tiene un espacio de direcciones que contiene los datos y el texto de programa, así como otros recursos, que podrían incluir archivos abiertos, procesos hijos, alarmas pendientes, manejadores de señales, información contable, etc. Al juntar todas estas cosas en forma de un proceso, se le puede administrar con más facilidad.

El otro concepto que tiene un proceso es un subproceso de ejecución, o simplemente subproceso. Éste tiene un contador de programa que indica cuál instrucción se ejecutará a continuación; tiene riesgos, que contiene sus variables de trabajo actuales, y tiene una pila, que contiene el historial de ejecución, con un marco por cada procedimiento invocado del cual todavía no se haya regresado. Aunque un subproceso debe ejecutarse en algún proceso, el subproceso y su proceso son conceptos distintos que pueden tratarse aparte. Los procesos sirven para agrupar recursos; los subprocesos son las entidades que se calendarizan para ejecutarse en la CPU.



Los subprocessos aportan al modelo de procesos la posibilidad de que haya varias ejecuciones en el mismo entorno de un proceso, en gran medida independientes una de otra. Tener múltiples subprocessos ejecutándose en paralelo en un proceso es análogo a tener múltiples procesos ejecutándose en paralelo en una computadora. Primero, los subprocessos comparten un espacio de direcciones, archivos abiertos y otros recursos. Segundo, los procesos comparten una memoria física, discos, impresoras y otros recursos. Debido a que los subprocessos tienen algunas de las propiedades de los procesos, a veces se les llama procesos ligeros. También se emplea el término múltiples procesos para describir la situación en la que se permiten varios subprocessos en el mismo proceso.

Los distintos subprocessos de un proceso no son tan independientes como son los procesos distintos. Todos los subprocessos tienen exactamente el mismo espacio de direcciones, lo que implica que comparten las mismas variables globales. Puesto que cada subprocesso puede tener acceso a todas las direcciones de memoria del espacio de direcciones del proceso, un subprocesso podría leer, modificar o incluso borrar por completo la pila de otro subprocesso. No existe protección entre los procesos porque es imposible y no debería ser necesaria. A diferencia de procesos distintos, que podrían pertenecer a usuarios distintos y ser hostiles entre sí, un proceso siempre pertenece a un solo usuario, y es de suponer que el usuario creó múltiples subprocessos con el fin de que cooperen, no de que peleen. Además de compartir un espacio de direcciones, todos los subprocessos comparten el mismo conjunto de archivos abiertos, procesos hijos, alarmas, señales, etc.

Si hay múltiples subprocessos, los procesos generalmente empiezan con un solo subprocesso. Éste puede crear subprocessos nuevos invocando a un procedimiento de biblioteca, `thread_create` y cuando termina `thread_exit`. Una vez hecho esto, desaparecerá y ya no podrá calendarizarse.

## Uso de subprocessos

Es precisamente el que se da en favor de tener procesos. Sólo que ahora con los subprocessos añadimos un elemento nuevo: la posibilidad de que las entidades paralelas comparten un espacio de direcciones y todos sus datos. Esta capacidad es indispensable en ciertas aplicaciones, y es por ello que el esquema de múltiples procesos no es la solución.

Un segundo argumento en favor de los subprocessos es que, al no estar enlazados con recursos, son más fáciles de crear y destruir que los procesos. En muchos sistemas, la creación de un subprocesso es 100 veces más rápida que la creación de un proceso. Si el número de subprocessos necesarios cambia en forma dinámica y con rapidez esta propiedad es útil.

Un tercer motivo para tener subprocessos también se relaciona con el desempeño. Los subprocessos no mejoran el desempeño cuando todos usan intensivamente la CPU, pero si se realiza una cantidad considerable tanto de cálculo como de E/S, los subprocessos permiten traslapar estas actividades y así acelerar la aplicación.

Son útiles en sistemas con múltiples CPU's, en los que es posible un verdadero paralelismo.

Si el programa fuera de un solo subprocesso, cada vez que se iniciara un respaldo en disco se ignorarían los comandos provenientes del teclado y el ratón, hasta que dicho respaldo terminara. El usuario percibiría esto como lentitud del sistema. Los sucesos de teclado y ratón interrumpirían el respaldo en disco para que el desempeño mejorara, pero eso daría pie a un modelo de programación complejo, controlado por interrupciones. Con tres subprocessos, el modelo de programación es mucho más sencillo. El primero se limita a interactuar con el usuario. El segundo reformatea el documento cuando se le solicita. El tercero escribe el contenido de la RAM en disco en forma periódica.



## Implementación de subprocessos en espacio de usuario

Hay dos formas principales de implementar subprocessos: en espacio de usuario y en el Kernel. La decisión ha dado pie a cierta controversia, y también existe una implementación híbrida. El primer método consiste en colocar por completo el sistema de subprocessos en espacio de usuario. El Kernel no sabe nada de ellos. En lo que a él respecta, está administrando procesos ordinarios, de un solo subprocesso. La primera ventaja, es que puede implementarse un sistema de subprocessos en el nivel de usuario en un sistema operativo que no maneje subprocessos. Todos los sistemas operativos solían pertenecer a esta categoría, y toda vía subsisten algunos.

Cuando los subprocessos se administran en espacio de usuario, cada proceso necesita su propia tabla de subprocessos privada para dar seguimiento a sus subprocessos. Esta tabla es análoga a la de procesos del Kernel, salvo que sólo guarda las propiedades de subprocessos individuales, como el contador de programa, el programa, apuntador de pila, registros, estado, etcétera, de cada uno. La tabla de subprocessos es administrada por el sistema de tiempo de ejecución. Cuando un proceso pasa al estado listo o al bloqueo, en la tabla de subprocessos se guarda la información necesaria para reiniciarlo, exactamente en la misma forma en la que el Kernel guarda información acerca de los procesos en la tabla de procesos.

Cuando un subprocesso hace algo que podría hacer que se bloquee localmente, como esperar que otro subprocesso del proceso permita cierto trabajo, invoca un procedimiento del sistema de tiempo de ejecución. Este procedimiento verifica si el subprocesso debe colocarse en estado bloqueado. Si es así, en la tabla de subprocessos se guardan los registros del subprocesso, se busca un subprocesso que esté listo para ejecutarse y se cargan en los registros de la máquina los valores guardados de ese nuevo proceso.

Tan pronto como se ha conmutado el apuntador de pila y el contador de programa, el nuevo subprocesso se activa en forma automática. Si la máquina cuenta con una instrucción para almacenar todos los registros y otra para recuperarlos, toda la conmutación de subprocessos puede efectuarse con unas cuantas instrucciones. Este tipo de conmutación de subprocessos es al menos un orden de magnitud más rápido que un salto al Kernel, y es un argumento de peso de manejar los subprocessos en el nivel de usuario.

Tanto el procedimiento que guarda el estado del subprocesso como el calendarizado son procedimientos locales, así que invocarlos es mucho más eficiente que llamar al Kernel. No se necesita una interrupción de sistema, no es preciso conmutar el contexto, no hay que guardar en disco el caché de memoria, etc. Esto agiliza mucho la calendarización de subprocessos.

Los subprocessos en el nivel de usuario tienen otras ventajas, como permitir que cada proceso tenga su propio algoritmo de calendarización personalizado. En algunas aplicaciones, como las que tienen un subprocesso recolector de basura, es una ventaja no tener que preocuparse porque un subprocesso vaya a detenerse en un momento poco conveniente. Es fácil aumentar su escala, pues los subprocessos de Kernel siempre requieren espacio de tabla y de pila en el Kernel, y esto puede ser problemático si hay un gran número de ellos.

Los sistemas de subprocessos en el nivel de usuario tienen problemas importantes. El principal es la forma en la que se implementen las llamadas bloqueadoras al sistema. Otro problema, hasta cierto punto análogo al de las llamadas bloqueadoras al sistema, es el de los fallos de página. Si el programa salta a una instrucción que no está en la memoria, ocurre un fallo de página y el sistema operativo trae la instrucción faltante del disco. El proceso se bloquea mientras se localiza y lee la instrucción necesaria.



Si un subprocesso causa un fallo de página, el Kernel, que ni siquiera sabe de la existencia de los subprocessos, bloqueará todo el proceso hasta que termine la E/S de disco, aunque otros subprocessos puedan seguir ejecutándose. Otro problema de los sistemas de subprocessos en el nivel de usuario es que, si un proceso comienza a ejecutarse, ningún otro subprocesso de ese proceso se ejecutará si el primero no cede de manera voluntaria la CPU. Dentro de un proceso dado no hay interrupciones de reloj, así que es imposible la calendarización de subprocessos por turno circular (round-robin). A menos que un subprocesso ingrese en el sistema de tiempo de ejecución por voluntad propia, el calendarizador no tendrá oportunidad de trabajar.

Una posible solución al problema de la ejecución indefinida de subprocessos es que el sistema de tiempo de ejecución solicite una señal de reloj una vez por segundo para asumir el control, pero esto también es burdo y molesto de programar. No siempre es posible emitir interrupciones de reloj periódicas con mayor frecuencia y, aunque se pudiera, implicaría un procesamiento adicional considerable. Un proceso también podría necesitar una interrupción de reloj, lo cual interferiría con el uso que hace del reloj el sistema de tiempo de ejecución.

#### [Implementación de subprocessos en el Kernel](#)

La tabla de subprocessos del Kernel contiene los riesgos, el estado y demás información de cada subprocessos. Estos datos son los mismos que se usan con subprocessos en el nivel de usuario, pero ahora están en el Kernel, no en el espacio de usuario. Esta información es un subconjunto de la que los kernels tradicionales mantienen acerca de cada uno de los procesos de un solo subprocesso, el estado del proceso. Además, el Kernel también mantiene la tabla de procesos tradicional con la que da seguimiento a los procesos.

Todas las llamadas que podrían bloquear un subprocesso que implementan como llamadas al sistema y tienen un costo mucho mayor que las llamadas a procedimientos de un sistema de tiempo de ejecución. Cuando un subprocesso se bloquea, el Kernel, puede ejecutar otro subprocesso del mismo proceso o alguno de otro proceso. Con subprocessos en el nivel de usuario, el sistema de tiempo de ejecución sigue ejecutando subprocessos de su propio proceso hasta que el Kernel le quita la CPU.

Debido al costo relativamente mayor de crear y destruir subprocessos en el Kernel, algunos sistemas adoptan un enfoque ecológico correcto y reciclan sus procesos. Cuando un subprocesso se destruye, se marca como no ejecutable, pero sus estructuras de datos en el Kernel no sufren alteración. Cuando es necesario crear un subprocesso, se reactiva un antiguo, ahorrando algo de procesamiento extra. Los subprocessos en el nivel de usuario también pueden reciclarse, pero debido a que el procedimiento extra que implica su administración es mucho menor, hay menos incentivo para hacerlo.

Los subprocessos de Kernel no necesitan nuevas llamadas al sistema no bloqueadoras. Si un subprocesso de un proceso causa un fallo de página, el Kernel puede verificar con facilidad si el proceso tiene algún otro subprocesso ejecutable, y en su caso, ejecutar uno de ellos mientras espera que la página necesaria llegue del disco. Su propia desventaja es el costo elevado de una llamada al sistema; si las operaciones de subprocessos son usuales, se requerirá mucho procesamiento extra.

#### Implementaciones híbridas

En este diseño, el Kernel sólo tiene conocimiento de los subprocessos en el nivel de Kernel y únicamente los calendariza a ellos. Algunos de ellos podrían tener multiplexados múltiples subprocessos de nivel de usuario. Estos se crean, destruyen y calendarizan al igual que los de nivel de usuario en un proceso que se



ejecuta en un sistema operativo sin capacidad de múltiples subprocessos, en este modelo, cada subprocesso en el nivel del Kernel tiene algún conjunto de subprocessos en el nivel de usuario que se turnan para usarlo.

#### Activaciones de calendarizador

Lo que se busca en las activaciones del calendarizador es imitar la funcionalidad de los subprocessos del Kernel pero con el desempeño y con la mayor flexibilidad que suelen tener los sistemas de subprocessos implementados en el espacio de usuario. En particular los subprocessos de usuario no deberían tener que emitir llamadas al sistema no bloqueadoras especiales ni verificar con antelación si es posible emitir con seguridad ciertas llamadas al sistema. Cuan un subprocesso sea Bloqueado por una llamada al sistema o un fallo de página, debería ser posible ejecutar otro subprocesso dentro del mismo proceso, si haya alguno listo.

La eficiencia se logra evitando transiciones innecesarias entre el espacio de usuario y el de Kernel. Si un subprocesso se bloquea en espera de que otro haga algo, por otro, el Kernel no tiene por qué intervenir, y es innecesario el procedimiento adicional de la transición Kernel usuario. El sistema de tiempo de ejecución en espacio de usuario puede bloquear el subprocesso sincronizador y calendarizar otro por su cuenta.

Cuando se usan activaciones del calendarizador, el Kernel asigna ciertas numerosas de procesadores virtuales a cada proceso y permite al sistema de tiempo de ejecución asignar subprocessos a procesadores. Este mecanismo también puede usarse en un multiprocesador en el que los procesadores virtuales podrían ser CPU's reales. Al principio el número de procesadores virtuales asignados a un proceso es uno, pero el proceso puede pedir más pero también devolver procesadores que ya necesite. El Kernel también puede recuperar procesadores virtuales ya asignados para asignarlos a otros procesos necesitados.

La idea fundamental que hace este esquema funcione es que cuando el Kernel sabe que un subprocesso sea bloqueado, lo notifica al sistema de tiempo de ejecución del proceso, pasándole como parámetro de la pila el número de subprocessos en cuestión y una descripción del suceso. La notificación consiste en el que el Kernel activa el sistema del tiempo de ejecución en una dirección de inicio conocida, algo parecido al uso de señal. Este mecanismo se denomina llamada directa.

Una vez activado, el sistema de tiempo de ejecución recalendarioz sus subprocessos, casi siempre marcando el actual como bloqueado, tomando otros subprocessos listos preparando sus registros y reiniciándolo. Más adelante cuando el Kernel se entere que el subprocesso original puede continuar su ejecución, en parar otra llamada al sistema de tiempo de ejecución para informarle del suceso. El sistema de tiempo de ejecución, puede reiniciar de inmediato el subprocesso bloqueado o bien marcado como listo para que se ejecute después.

Si se presenta una interrupción de software mientras se está ejecutando un subprocesso de usuario, la CPU interrumpida cambia a modo de Kernel. Si la interrupción se debió a un suceso que no interesa el proceso interrumpido, digamos que terminó la E/S de otro proceso cuando el manejador de interrupciones termine, colocará el subprocesso interrumpido otra vez en el estado que estaba antes de interrupciones termine, colorará el subprocesso interrumpido otra vez en el estado en el que estaba antes de la interrupción. En cambio si la interrupción le interesa al proceso, no se iniciará el subprocesso interrumpido; en vez de ello se suspenderá el sistema de tiempo de ejecución en esa CPU virtual, con el estado del subprocesso interrumpido en la pila. Entonces corresponderá al sistema de tiempo de ejecución decidir cuál subprocesso calendarizará en esa CPU: el interrumpido el que reinicie está listo o algún otro.



## Subprocesos emergentes

Los subprocessos pueden ser útiles en los sistemas distribuidos. El enfoque tradicional consiste en tener un proceso o subproceso que se bloquea después de emitir una llamada, en espera de un mensaje. Cuando llega un mensaje este se acepta y procesa.

Puede adaptarse un enfoque completamente distinto en el que la llegada de un mensaje hace que el sistema crea un subproceso para manejar dicho mensaje. Este tipo de subprocessos se denomina **subproceso emergente**. Una ventaja clave de los subprocessos emergentes es que debió a que son totalmente nuevos no tienen historial-registros, pila, etcétera- que debe restaurarse; cada uno inicia desde cero y todos son idénticos, y esto agiliza su creación. El mensaje que llegó se entrega al subproceso para que lo procese; el resultado de usar subprocessos emergentes es una importante reducción en la tardanza entre la llegada del mensaje y el inicio del subproceso.

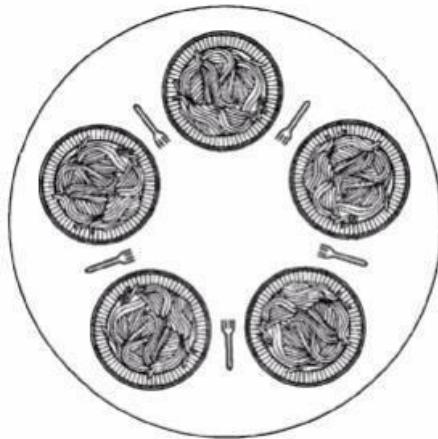
Se necesita planear un poco por adelante cuando se usan subprocessos emergentes. Si el sistema maneja subprocessos que se ejecutan en el contexto del Kernel el subproceso podría ejecutarse allí. Hacer que el subproceso emergente se ejecute en el espacio del Kernel suele ser más fácil y rápido que colocarlo en el espacio de usuario. Además un subproceso emergente en espacio de kernel puede tener acceso con facilidad a todas las tablas del kernel y a los dispositivos E/S lo cual podría ser necesario para procesar interrupciones. Por otra parte un subproceso de kernel con errores podría causar más daño que uno de usuario que también tuviera errores.

## Problemas clásicos de IPC

El problema general de la Calendarización ha sido descrito de diferentes maneras en la literatura. Usualmente es una redefinición en la noción clásica del problema de secuencia, considerando la calendarización como un proceso para la administración de recursos. Esta administración de recursos es básicamente un mecanismo o política usada para el manejo eficiente y efectivo del acceso a los recursos y el uso de los mismos recursos por sus varios consumidores (procesos).

### Filósofos comelones

El problema de la cena de filósofos en 1965, Dijkstra planteó y resolvió un problema de sincronización al que llamó problema de la cena de filósofos. El problema tiene un planteamiento muy sencillo. Cinco filósofos están sentados alrededor de una mesa circular. Cada filósofo tiene ante sí un plato de espagueti. El espagueti es tan resbaloso que un filósofo necesita dos tenedores para comerlo. Entre cada par de platos hay un tenedor.



La vida de un filósofo consiste en periodos alternantes de comer y pensar. Cuando un filósofo siente hambre, trata de adquirir sus tenedores izquierdo y derecho, uno a la vez, en cualquier orden. Si logra adquirir dos tenedores, comerá durante un rato, luego pondrá los tenedores en la mesa y seguirá pensando. La pregunta clave es: ¿podemos escribir un programa para cada filósofo que haga lo que se supone que debe hacer y nunca se trabe?

```
#define N 5 //N = número de comensales, palillos, filósofos, etc...
semáforo palillo[N]={1}; //arreglo de N semáforos que representan los palillos. void filosofo (int i) //proceso de
cada filosofo
{
    while (true) //bucle infinito, se puede colocar condición de parada con variable
    {
        pensar (); //tiempo de espera aleatorio de c/filosofo antes de 'comer' wait (palillo [i], palillo[i
+1] mod 5); //pido los 2 palillos simultáneamente comer () //tiempo de espera mientras come
        (puede ser aleatorio) signal (palillo [i]); //libero palillo izquierdo
        signal (palillo [(i + 1) mod 5]); //libero palillo derecho
        /* El orden en que libero los palillos no es de especial relevancia. */
    }
}
void main()
{
    parbegin (filosofo (0), filosofo (1), filosofo (2), filosofo (3), ..., filosofo (N));
    //inicio de los procesos (filósofos)
}
```

El procedimiento “tomar tenedor” espera hasta que el tenedor especificado está disponible y luego se apodera de él. Desafortunadamente, la solución obvia está equivocada. Supongamos que todos los filósofos toman su tenedor izquierdo simultáneamente. Ninguno podrá tomar su tenedor derecho, y tendremos un

bloqueo mutuo. Podríamos modificar el programa de modo que, después de tomar el tenedor izquierdo, el programa verifique si el tenedor derecho está disponible. Si no es así, el filósofo soltará su tenedor izquierdo, esperará cierto tiempo, y repetirá el proceso. Esta propuesta también fracasa, aunque por una razón distinta. Con un poco de mala suerte, todos los filósofos podrían iniciar el algoritmo simultáneamente, tomar su tenedor izquierdo, ver que su tenedor derecho no está disponible, dejar su tenedor izquierdo, esperar, tomar su tenedor izquierdo otra vez de manera simultánea, y así eternamente. Una situación así, en la que todos los programas continúan ejecutándose de manera indefinida pero no logran avanzar se denomina inanición.

\*Inanición: Ocurre cuando a un proceso o un hilo de ejecución se le deniega siempre el acceso a un recurso compartido. Sin este recurso, la tarea a ejecutar no puede ser nunca finalizada.

Ahora podríamos pensar: “si los filósofos esperan un tiempo aleatorio en lugar del mismo tiempo después de fracasar en su intento por disponer del tenedor derecho, la posibilidad de que sus acciones continuaran coordinadas durante siquiera una hora es excesivamente pequeña”. Esto es cierto, pero en algunas aplicaciones preferiríamos una solución que siempre funcione y que no tenga posibilidad de fallar debido a una serie improbable de números aleatorios. Una mejora que no está sujeta a bloqueo ni inanición consiste en proteger las cinco instrucciones que siguen a la llamada a “pensar” con un semáforo binario. Antes de comenzar a conseguir tenedores, un filósofo ejecutaría DOWN con mutex. Después de dejar los tenedores en la mesa, ejecutaría up con mutex. Desde un punto de vista teórico, esta solución es adecuada.

En la práctica, tiene un problema de rendimiento: sólo un filósofo puede estar comiendo en un instante dado. Si hay cinco tenedores disponibles, deberíamos estar en condiciones de permitir que dos filósofos comieran al mismo tiempo.



La solución correcta admite un paralelismo máximo con un número arbitrario de filósofos. Se utiliza un arreglo “estado” para mantenerse al tanto de si un filósofo está comiendo, pensando o hambriento tratando de disponer de tenedores. Un filósofo sólo puede pasar a la situación de “comiendo” si ninguno de sus vecinos está comiendo. Los vecinos del filósofo *i* están definidos por las macros LEFT y RIGHT.

En otras palabras, si “*i*” es 2, LEFT es 1 y RIGHT es 3.

El programa utiliza un arreglo de semáforos, uno por filósofo, de modo que los filósofos hambrientos pueden bloquearse si los tenedores que necesitan están ocupados. Cada proceso ejecuta el procedimiento “filósofo” como código principal, pero los demás procedimientos, “tomar tenedores”, “poner tenedores” y “probar” son procedimientos ordinarios y no procesos aparte. El problema de la cena de filósofos es útil para modelar procesos que compiten por tener acceso exclusivo a un número limitado de recursos, como dispositivos de E/S.

#### El problema de lectores y escritores

Otro problema famoso es el de los lectores y escritores (Courtois et al., 1971), que modela el acceso a una base de datos. Supóngase una base de datos por ejemplo un sistema de reservaciones de una línea aérea, con muchos procesos que compiten por leer y escribir en ella. Se puede permitir que varios procesos lean de la base de datos al mismo tiempo, pero si uno de los procesos está escribiendo (es decir, modificando) la base de datos, ninguno de los demás debería tener acceso a ésta, ni siquiera los lectores. La pregunta es, ¿cómo programamos a los lectores y escritores?

```

typedef int semaphore;
semaphore mutex = 1;
semaphore db = 1;
int rc = 0;

void reader(void)
{
    while (TRUE) {
        down(&mutex);
        rc = rc + 1;
        if (rc == 1) down(&db);
        up(&mutex);
        read_data_base();
        down(&mutex);
        rc = rc - 1;
        if (rc == 0) up(&db);
        up(&mutex);
        use_data_read();
    }
}

void writer(void)
{
    while (TRUE) {
        think_up_data();
        down(&db);
        write_data_base();
        up(&db);
    }
}

```

/\* use su imaginación \*/  
 /\* controla el acceso a 'rc' \*/  
 /\* controla el acceso a la base de datos \*/  
 /\* núm. de procesos que leen o quieren leer \*/  
  
 /\* repetir indefinidamente \*/  
 /\* obtener acceso exclusivo a 'rc' \*/  
 /\* ahora un lector más \*/  
 /\* si éste es el primer lector ... \*/  
 /\* liberar el acceso exclusivo a 'rc' \*/  
 /\* acceder a los datos \*/  
 /\* obtener acceso exclusivo a 'rc' \*/  
 /\* ahora un lector menos \*/  
 /\* si éste es el último lector ... \*/  
 /\* liberar el acceso exclusivo a 'rc' \*/  
 /\* región no crítica \*/  
  
 /\* repetir indefinidamente \*/  
 /\* región no crítica \*/  
 /\* obtener acceso exclusivo \*/  
 /\* actualizar los datos \*/  
 /\* liberar el acceso exclusivo \*/

En esta solución, el primer lector que obtiene acceso a la base de datos ejecuta DOWN con el semáforo “db”. Los lectores subsecuentes se limitan a incrementar un contador, “rc”. Conforme los lectores salen, se decremente el contador, y el último en salir ejecuta UP con el semáforo para permitir que un escritor bloqueado, si lo existe, entre. Supongamos que mientras un lector está usando la base de datos, llega otro lector. Puesto que tener dos lectores al mismo tiempo no está prohibido, se admite al segundo lector. También pueden admitirse un tercer lector y lectores subsecuentes si llegan.

Supongamos ahora que llega un escritor. El escritor no puede ser admitido en la base de datos, pues requiere acceso exclusivo, de modo que el escritor queda suspendido. Más adelante, llegan lectores adicionales. En tanto haya al menos un lector activo, se admitirán lectores subsecuentes. A consecuencia de esta estrategia, en tanto haya un suministro constante de lectores, entrarán tan pronto como lleguen. El escritor se mantendrá suspendido hasta que no haya ningún lector presente esto implica que el escritos nunca entrara.



Para evitar esta situación, el programa podría incluir una pequeña modificación: cuando llega un lector y un escritor está esperando, el lector queda suspendido detrás del escritor en lugar de ser admitido inmediatamente. Así, un escritor tiene que esperar hasta que terminan los lectores que estaban activos cuando llegó, pero no a que terminen los lectores que llegaron después de él. La desventaja de esta solución es que logra menor concurrencia y por tanto un menor rendimiento.

#### El problema del peluquero dormido

Otro problema de IPC clásico ocurre en una peluquería. Esta peluquería tiene un peluquero, una silla de peluquero y n sillas donde pueden sentarse los clientes que esperan. Si no hay clientes presentes, el peluquero se sienta en la silla de peluquero y se duerme.

Cuando llega un cliente, tiene que despertar al peluquero dormido. Si llegan clientes adicionales mientras el



peluquero está cortándole el pelo a un cliente, se sientan (si hay sillas vacías) o bien salen del establecimiento (si todas las sillas están ocupadas). El problema consiste en programar al peluquero y sus clientes sin entrar en condiciones de competencia. La solución hace uso de semáforos: “customers” que cuenta a los clientes en espera (excluyendo al que está siendo atendido), “barbers” el peluquero que está ocioso, esperando clientes (0 o 1), y mutex (exclusión mutua).

También necesitamos una variable, “esperando” que también cuenta los clientes que están esperando, y que en esencia es una copia de `customers`. Necesitamos esta variable porque no es posible leer el valor actual de un semáforo. En esta solución, un cliente que entra en la peluquería debe contar el número de clientes que esperan. Si este número es menor que el número de sillas, se queda; si no, se va.

Cuando el peluquero llega a trabajar en la mañana, ejecuta el procedimiento `barber` (peluquero) que lo obliga a bloquearse en espera de `customers` (clientes) hasta que llegue alguien. Luego se duerme, cuando un cliente llega, ejecuta `customer` (cliente), cuya primera instrucción es adquirir `mutex` para entrar en una región crítica. Si otro cliente llega poco tiempo después, no podrá hacer nada hasta que el primero haya liberado `mutex`.

A continuación, el cliente verifica si el número de clientes en espera si es menor que el número de sillas. Si no es así, el cliente libera `mutex` y se sale sin su corte de pelo. Si hay una silla disponible, el cliente incrementa la variable “espera” y luego ejecuta `UP` con el semáforo `customers`, lo que despierta al peluquero. En este punto, tanto el peluquero como el cliente están despiertos. Cuando el cliente libera `mutex`, el peluquero lo toma, realiza algo de aseo e inicia el corte de pelo. Una vez terminado el corte de pelo, el cliente sale del procedimiento y de la peluquería. A diferencia de los ejemplos anteriores, no hay un ciclo para el cliente porque cada uno sólo recibe un corte de pelo. El peluquero sí opera en un ciclo, tratando de atender al siguiente cliente. Si hay uno presente, el peluquero realiza otro corte de pelo si no, se duerme.

#### [El problema productor/consumidor](#)

El problema Productor/Consumidor consiste en el acceso concurrente por parte de procesos productores y procesos consumidores sobre un recurso común que resulta ser un buffer de elementos.



Los productores tratan de introducir elementos en el buffer de uno en uno, y los consumidores tratan de extraer elementos de uno en uno. Para asegurar la consistencia de la información almacenada en el buffer, el acceso de los productores y consumidores debe hacerse en exclusión mutua. Adicionalmente, el buffer es de capacidad limitada, de modo que el acceso por parte de un productor para introducir un elemento en el buffer lleno debe provocar la detención del proceso productor. Lo mismo sucede para un consumidor que intente extraer un elemento del buffer vacío.

## ALGORITMOS DE CALENDARIZACIÓN ENTRE PROCESOS

Componente del sistema operativo que decide cuál de los procesos es el que entrara a la CPU. Su decisión es basada según el sistema que este administrando y es resuelta por los Algoritmos de Calendarización. En la época de los sistemas por lote con entradas en forma de imágenes de tarjetas en una cinta magnética, el algoritmo de planificación era sencillo: simplemente se ejecutaba el siguiente trabajo de la cinta: Cuando aparecieron los sistemas de tiempo compartido, el algoritmo de calendarización se volvió más complejo porque casi siempre había varios usuarios en espera de ser atendidos Incluso en las computadoras personales, puede haber varios procesos iniciados por el usuario compitiendo por la CPU, sin mencionar los trabajos de segundo plano, como los demonios de red o de correo electrónico que envían o reciben mensajes.

Antes de examinar algoritmos de planificación específicos, debemos pensar en qué está tratando de lograr el planificador. Después de todo, éste se ocupa de decidir una política, no de proveer un mecanismo.

1. Equitatividad —asegurarse de que cada proceso reciba una parte justa del tiempo de CPU.
2. Eficiencia —mantener la CPU ocupada todo el tiempo.

3. Tiempo de respuesta –minimizar el tiempo de respuesta para usuarios interactivos.
4. Retorno –minimizar el tiempo que los usuarios por lotes tienen que esperar sus salidas.
5. Volumen de producción –maximizar el número de trabajos procesados por hora.

Algunos de estos objetivos son contradictorios. Si queremos minimizar el tiempo de respuesta para los usuarios interactivos, el planificador no deberá ejecutar trabajos por lotes a los usuarios por lotes seguramente no les gustaría este algoritmo, pues viola el criterio 4. Después de todo, la cantidad de tiempo de CPU disponible es finita. Para darle más a un usuario tenemos que darle menos a otro. Una complicación que deben enfrentar los planificadores es que cada proceso es único e impredecible.

Algunos dedican una buena parte del tiempo a esperar E/S de archivos, mientras otros usarán la CPU durante horas si se les permitiera hacerlo. Cuando el planificador comienza a ejecutar un proceso, nunca sabe con certeza cuánto tiempo pasará antes de que dicho proceso se bloquee, sea para E/S, en espera de un semáforo o por alguna otra razón. Para asegurarse de que ningún proceso se ejecute durante demasiado tiempo, casi todas las computadoras tienen incorporado un cronómetro o reloj electrónico que genera interrupciones periódicamente.

Cuando calendarizamos:

- 1.-Al crear un proceso (decidir si entra el padre o el hijo)
- 2.-Al terminar un proceso (Antes de agotar su tiempo se debe elegir a otro listo o inactivo)
- 3.-Al bloquear un proceso



4.-Al recibir una interrupción (debe decidirse cual sacar/meter una vez terminada la interrupción)

La estrategia de permitir que procesos lógicamente ejecutables se suspendan temporalmente se denomina planificación expropiativa. La ejecución hasta terminar se denomina planificación no expropiativa. Un proceso puede ser suspendido en un instante arbitrario, sin advertencia, para que otro proceso pueda ejecutarse. Esto da pie a condiciones de competencia y requiere semáforos, monitores, mensajes o algún otro método avanzado para prevenirlas.

Aunque los algoritmos de planificación no expropiativos son sencillos y fáciles de implementar, por lo regular no son apropiados para sistemas de aplicación general con varios usuarios que compiten entre sí. Por otro lado, en un sistema dedicado como un servidor de base de datos, bien puede ser razonable que el proceso padre inicie un proceso hijo para trabajar con una solicitud y dejarlo que se ejecute hasta terminar o bloquearse.

La diferencia respecto al sistema de aplicación general es que todos los procesos del sistema de bases de datos están bajo el control de uno solo, que sabe lo que cada hijo va a hacer y cuánto va a tardar.

Categorías de algoritmos de Calendarización 1.- POR LOTES

Se recomienda la calendarización no expropiativa. 2.-INTERACTIVOS

Se recomienda la calendarización no expropiativa para que no acaparen recursos 3.- TIEMPO REAL

Casi no requiere la expropiación porque son procesos que actúan sobre una sola aplicación.

## Calendarización en sistemas por lotes

### FIFO (primero en llegar primero en ser atendido)

Es uno de los más sencillos que es no expropiativo, aquí la CPU se asigna a los procesos en el orden en que lo solicitan. Hay una cola de procesos listos. Cuando el primer trabajo entra en el sistema en la mañana, se le inicia de inmediato y se le permite ejecutar todo el tiempo que desee. A medida que llegan otros trabajos se les coloca al final de la cola. No se recomienda en procesos con tiempos muy heterogéneos.

### Trabajo más corto primero

Este supone un conocimiento anticipado de los tiempos de ejecución. Si hay varios trabajos de la misma importancia en la cola de entrada, el calendarizador escoge el trabajo más corto primero.

8	4	4	4
A	B	C	D

(a)

4	4	4	8
B	C	D	A

(b)

Tiempos de retorno:  
 $A=8$ ,  $B=12$ ,  $C=16$  y  $D=20$   
Promedio: 14

Tiempos de retorno:  
 $A=4$ ,  $B=8$ ,  $C=12$  y  $D=20$   
Promedio: 11

a) Ejecución de orden original b) Ejecución trabajo más corto primero



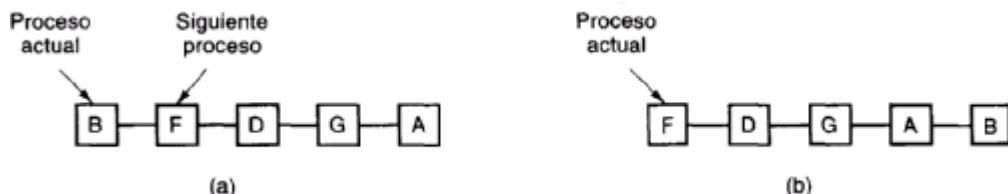
### Tiempo restante más corto primero

Una versión expropiativa de la estrategia anterior es la de tiempo restante más corto a primero. En este algoritmo, el calendarizador siempre escoge el proceso con base en el tiempo que falta para que termine de ejecutarse, en este caso también es preciso conocer los tiempos de ejecución. Este esquema permite que trabajos cortos nuevos obtengan buen servicio.

### Calendarización en sistemas interactivos

#### PLANIFICACIÓN ROUND ROBIN

Uno de los más antiguos, sencillos, equitativos y ampliamente utilizados es el de round robin. A cada proceso se le asigna un intervalo de tiempo, llamado cuanto, durante el cual se le permite ejecutarse. Si el proceso todavía se está ejecutando al expirar su cuanto, el sistema operativo se apropiá de la CPU y se la da a otro proceso. Si el proceso se bloquea o termina antes de expirar el cuanto, la comutación de CPU naturalmente se efectúa cuando el proceso se bloquee. Todo lo que el planificador tiene que hacer es mantener una lista de procesos ejecutables. Cuando un proceso gasta su cuanto, se le coloca al final de la lista.



La única cuestión interesante cuando se usa el round robin es la duración del cuánto.

La conmutación de un proceso a otro requiere cierto tiempo para llevar a cabo las tareas administrativas: guardar y cargar registros y mapas de memoria, actualizar diversas tablas y listas, etc. Supongamos que el primer proceso se inicia de inmediato, el segundo podría no iniciarse hasta cerca de medio segundo después, y así sucesivamente. El pobre proceso que le haya tocado ser último podría tener que esperar 5 segundos antes de tener su oportunidad, suponiendo que los demás procesos utilizan su cuanto completo. Para casi cualquier usuario, un retardo de 5 segundos en la respuesta a un comando corto sería terrible.

El mismo problema puede presentarse en una computadora personal que maneja multiprogramación. Escoger un cuanto demasiado corto causa demasiadas Conmutaciones de procesos y reduce la eficiencia de la CPU, pero escogerlo demasiado largo puede dar pie a una respuesta deficiente a solicitudes interactivas cortas.

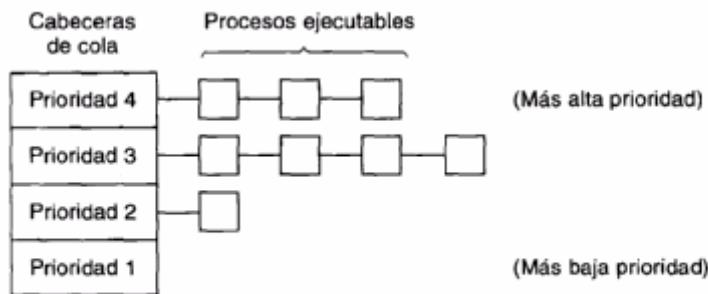
#### Planificación por prioridad

La planificación en round robin supone implícitamente que todos los procesos son igualmente importantes. Con frecuencia, las personas que poseen y operan sistemas de computadora multiusuario tienen ideas diferentes acerca del tema. La necesidad de tener en cuenta factores externos da pie a la planificación por prioridad. La idea básica es sencilla: a cada proceso se le asigna una prioridad, y se permite que se ejecute el proceso ejecutable que tenga la prioridad más alta. Incluso en una PC con un solo dueño, puede haber múltiples procesos, algunos más importantes que otros.

A fin de evitar que los procesos de alta prioridad se ejecuten indefinidamente, el planificador puede reducir la prioridad de los procesos que actualmente se ejecutan en cada tic del reloj (esto es, en cada interrupción de reloj). Si esta acción hace que la prioridad se vuelva menor que la del siguiente proceso con más alta prioridad, ocurrirá una conmutación de procesos. Como alternativa, se podría asignar a cada proceso un cuanto máximo en el que se le permitiera tener la CPU.



continuamente; cuando se agota este cuanto, se da oportunidad al proceso con la siguiente prioridad más alta de ejecutarse. Podemos asignar prioridades a los procesos estática o dinámicamente por hora.



**Figura 2-23.** Algoritmo de planificación con cuatro clases de prioridad.

#### Colas múltiples

Uno de los primeros planificadores por prioridad se incluyó en CTSS (Corbató et al., 1962). CTSS tenía el problema de que la conmutación de procesos era muy lenta porque la 7094 sólo podía contener un proceso en la memoria. Cada conmutación implicaba escribir el proceso actual en disco y leer uno nuevo del disco. Los diseñadores de CTSS pronto se dieron cuenta de que resultaba más eficiente dar a los procesos limitados por CPU un cuanto largo de vez en cuando, en lugar de darles cuantos pequeños muy a menudo (porque se reducía el intercambio). Por otro lado, dar a todos los procesos un cuanto largo implicaría un tiempo de respuesta deficiente, como ya hemos visto. Su solución consistió en establecer clases de prioridad. Los procesos de la clase más alta se ejecutaban durante un cuanto.

Los procesos de la siguiente clase más alta se ejecutaban durante dos cuantos. Los procesos de la siguiente clase se ejecutaban durante cuatro cuantos, y así sucesivamente. Cada vez que un proceso agotaba todos los cuantos que tenía asignados, se le degradaba una clase.

Se adoptó la siguiente política para evitar que un proceso que en el momento de iniciarse necesita ejecutarse durante un tiempo largo pero posteriormente se vuelve interactivo fuera castigado indefinidamente. Se han utilizado muchos otros algoritmos para asignar procesos a clases de prioridad.

Por ejemplo, el influyente sistema XDS 940 (Lampson, 1968), construido en Berkeley, tenía cuatro clases de prioridad, llamadas terminal, E/S, cuanto corto y cuanto largo. Cuando un proceso que estaba esperando entradas de la terminal finalmente se despertaba, pasaba a la clase de prioridad más alta (terminal). Cuando un proceso que estaba esperando un bloque de disco quedaba listo, pasaba a la segunda clase. Si un proceso seguía en ejecución en el momento de expirar su cuanto, se le colocaba inicialmente en la tercera clase, pero si agotaba su cuanto demasiadas veces seguidas sin bloquearse para E/S de terminal o de otro tipo, se le bajaba a la cuarta cola. Muchos otros sistemas usan algo similar para dar preferencia a los usuarios y procesos interactivos por encima de los de segundo plano.

#### [El primer trabajo más corto](#)

La mayor parte de los algoritmos anteriores se diseñaron para sistemas interactivos. Examinemos ahora uno que resulta especialmente apropiado para los trabajos por lotes cuyos tiempos de ejecución se conocen por adelantado. En una compañía de seguros, por ejemplo, es posible predecir con gran exactitud cuánto tiempo tomará ejecutar un lote de 1000 reclamaciones, pues se efectúan trabajos similares todos los días. Si hay varios trabajos de igual importancia esperando en la cola de entrada para ser iniciados, el planificador deberá usar el criterio del primer trabajo más corto. Examinemos la figura. Aquí encontramos cuatro trabajos, A, B, C y D, con tiempos de ejecución de 8, 4, 4 y 4 minutos, respectivamente. Si los ejecutamos en ese orden, el tiempo de retomo para A será de 8 minutos, para B, de 12 minutos, para C, de 16 minutos, y para D, de 20 minutos, siendo el promedio de 14 minutos.



8	4	4	4
A	B	C	D

(a)

4	4	4	8
B	C	D	A

(b)

**Figura 2-24.** Ejemplo de planificación del primer trabajo más corto.

Consideremos ahora la ejecución de estos trabajos usando el primer trabajo más corto. Los tiempos de retomo son ahora de 4, 8, 12 y 20 minutos para un promedio de 11 minutos. Se puede demostrar que la política del primer trabajo más corto es óptima. Consideremos el caso de cuatro trabajos, con tiempos de ejecución de a, b, c y d, respectivamente. El primer trabajo termina en un tiempo a, el segundo, en a + b, etc. El tiempo de retomo medio es  $(4a + 3b + 2c + d)/4$ . Es evidente que a contribuye más al promedio que los demás tiempos, por lo que debe ser el trabajo más corto, siguiendo b, c y por último d, que es el más largo y sólo afecta su propio tiempo de retomo.

El mismo argumento es aplicable a cualquier cantidad de trabajos. Dado que la política del primer trabajo más corto produce el tiempo de respuesta medio mínimo, sería deseable poderlo usar también para procesos interactivos. Esto es posible hasta cierto punto. Los procesos interactivos generalmente siguen el patrón de esperar un comando, ejecutar el comando, esperar un comando, ejecutar el comando, etc. Si consideramos la ejecución de cada comando como un “trabajo” individual, podremos minimizar el tiempo de respuesta global ejecutando primero el trabajo más corto. El único problema es determinar cuál de los procesos ejecutables es el más corto. Una estrategia consiste en hacer estimaciones basadas en el comportamiento histórico y ejecutar el proceso con el tiempo de ejecución estimado más corto.

La técnica de estimar el siguiente valor de una serie calculando la media ponderada del valor medido actual y el estimado previo también se conoce como maduración, y es aplicable a muchas situaciones en las que debe hacerse una predicción basada en valores previos.

## Planificación garantizada

Una estrategia de planificación totalmente distinta consiste en hacer promesas reales al usuario en cuanto al rendimiento y después cumplirlas. Una promesa que es realista y fácil de cumplir es la siguiente: si hay  $n$  usuarios en sesión mientras usted está trabajando, usted recibirá aproximadamente  $1/n$  de la capacidad de la CPU. De forma similar, en un sistema monousuario con  $n$  procesos en ejecución, si todo lo demás es igual, cada uno deberá recibir un de los ciclos de CPU.

Para poder cumplir esta promesa, el sistema debe llevar la cuenta de cuánto tiempo de CPU ha tenido cada proceso desde su creación. A continuación, el sistema calculará el tiempo de CPU al que tenía derecho cada proceso, es decir, el tiempo desde la creación dividido entre  $n$ . Puesto que también se conoce el tiempo de CPU del que cada proceso ha disfrutado realmente, es fácil calcular la relación entre el tiempo de CPU recibido y el tiempo al que se tenía derecho. Una relación de 0.5 implica que el proceso sólo ha disfrutado de la mitad del tiempo al que tenía derecho, y una relación de 2.0 implica que un proceso ha tenido dos veces más tiempo del que debería haber tenido. El algoritmo consiste entonces en ejecutar el trabajo con la relación más baja hasta que su relación haya rebasado la de su competidor más cercano.

## Planificación por lotería

Si bien hacer promesas a los usuarios y después cumplirlas es una idea admirable, es difícil de implementar. Podemos usar otro algoritmo para obtener resultados igualmente predecibles con una implementación mucho más sencilla. El algoritmo se llama planificación por lotería (Waldspurger y Weihl, 1994). La idea básica consiste en dar a los procesos boletos de lotería para los diversos recursos del sistema, como el tiempo de CPU. Cada vez que se hace necesario tomar una decisión de planificación, se escoge al azar un boleto de lotería, y el proceso poseedor de ese boleto obtiene el recurso.



Cuando se aplica a la planificación del tiempo de CPU, el sistema podría realizar una lotería 50 veces por segundo, concediendo a cada ganador 20 ms de tiempo de CPU como premio. Parafraseando a George Orwell, “todos los procesos son iguales, pero algunos son más iguales que otros”.

Podemos dar más boletos a los procesos más importantes, a fin de aumentar sus posibilidades de ganar. Si hay 100 boletos pendientes, y un proceso tiene 20 de ellos, tendrá una probabilidad del 20% de ganar cada lotería. A largo plazo, obtendrá cerca del 20% del tiempo de CPU. En contraste con los planificadores por prioridad, donde es muy difícil establecer qué significa realmente tener una prioridad de 40, aquí la regla es clara: un proceso que posee una fracción de los boletos obtendrá aproximadamente una fracción  $f$  del recurso en cuestión.

La planificación por lotería tiene varias propiedades interesantes. Por ejemplo, si aparece un proceso nuevo y se le conceden algunos boletos, en la siguiente lotería ya tendrá una probabilidad de ganar que será proporcional al número de boletos que recibió. En otras palabras, la planificación por lotería es de respuesta muy rápida

#### [Planificación en tiempo real](#)

Un sistema de tiempo real es uno en el que el tiempo desempeña un papel esencial. Por lo regular, uno o más dispositivos físicos externos a la computadora generan estímulos, y la computadora debe reaccionar a ellos de la forma apropiada dentro de un plazo fijo. Por ejemplo, la computadora de un reproductor de discos compactos recibe los bits conforme salen de la unidad de disco y los debe convertir en música dentro de un intervalo de tiempo muy estricto. Si el cálculo toma demasiado tiempo, la música sonará raro.

Otros sistemas de tiempo real son los de monitoreo de pacientes en las unidades de cuidado intensivo de los hospitales, el piloto automático de un avión y los controles de seguridad de un reactor nuclear. En todos estos casos, obtener la respuesta correcta pero demasiado tarde suele ser tan malo como no obtenerla.

Los sistemas de tiempo real generalmente se clasifican como de tiempo real estricto, lo que implica que hay plazos absolutos que deben cumplirse a como dé lugar, y tiempo real flexible, lo que implica que es tolerable no cumplir ocasionalmente con un plazo. En ambos casos, el comportamiento de tiempo real se logra dividiendo el programa en varios procesos, cada uno de los cuales tiene un comportamiento predecible y conocido por adelantado. Estos procesos generalmente son de corta duración y pueden ejecutarse hasta terminar en menos de un segundo. Cuando se detecta un suceso externo, el planificador debe programar los procesos de modo tal que se cumplan todos los plazos.

Los sucesos a los que puede tener que responder un sistema de tiempo real pueden clasificarse también como periódicos (que ocurren a intervalos regulares) o aperiódicos (que ocurren de forma impredecible). Es posible que un sistema tenga que responder a múltiples corriente de eventos periódicos. Dependiendo de cuánto tiempo requiere cada suceso para ser procesado, tal vez ni siquiera sea posible manejarlos todos.

Por ejemplo, si hay  $m$  eventos periódicos y el evento  $i$  ocurre con el periodo  $P_i$  y requiere  $C_i$  segundos de tiempo de CPU para ser manejado, la carga sólo podrá manejarse si  $\sum C_i \leq P_i$ . Un sistema de tiempo real que satisface este criterio es planificable. Consideraremos brevemente unos cuantos algoritmos de planificación de tiempo real dinámicos. El algoritmo clásico es el algoritmo de tasa monotónica (Liu y Layland, 1973), que asigna por adelantado a cada proceso una prioridad proporcional a la frecuencia de ocurrencia de su evento disparador. Por ejemplo, un proceso que se debe ejecutar cada 20 ms recibe una prioridad de 50, y uno que debe ejecutarse cada 100 ms recibe una prioridad de 10.



En el momento de la ejecución, el planificador siempre ejecuta el proceso listo que tiene la más alta prioridad, desalojando al proceso en ejecución si es necesario. Liu y Layland demostraron que este algoritmo es óptimo. Otro algoritmo de planificación en tiempo real muy utilizado es el del primer plazo más próximo. Cada vez que se detecta un evento, su proceso se agrega a la lista de procesos listos, la cual se mantiene ordenada por plazo, que en el caso de un evento periódico es la siguiente ocurrencia del evento. El algoritmo ejecuta el primer proceso de la lista, que es el que tiene el plazo más próximo.

Un tercer algoritmo calcula primero para cada proceso la cantidad de tiempo que tiene de sobra, es decir, su holgura. Si un proceso requiere 200 ms y debe terminar en un plazo de 250 ms, tiene una holgura de 50 ms. El algoritmo, llamado de menor holgura, escoge el proceso que tiene menos tiempo de sobra. Si bien en teoría es posible convertir un sistema operativo de aplicación general en uno de tiempo real usando uno de estos algoritmos de planificación, en la práctica el gasto extra de la conmutación de contexto de los sistemas de aplicación general es tan grande que el desempeño de tiempo real sólo puede lograrse en aplicaciones con restricciones de tiempo muy holgadas. En consecuencia, en la mayor parte de los trabajos en tiempo real se usan sistemas operativos de tiempo real especiales que tienen ciertas propiedades importantes.

Por lo regular, éstas incluyen un tamaño pequeño, un tiempo de interrupción rápido, una conmutación de contexto rápida, un intervalo corto durante el cual se inhabilitan las interrupciones, y la capacidad para controlar múltiples cronómetros con precisión de milisegundos o microsegundos.

## UNIDAD III:

# ESQUEMAS DE ADMINISTRACIÓN DE MEMORIA: PAGINACIÓN Y SEGMENTACIÓN.

### Memoria

La memoria es un bloque fundamental del computador, cuya misión consiste en almacenar los datos y las instrucciones. La memoria principal, es el órgano que almacena los datos e instrucciones de los programas en ejecución.

La memoria solo puede realizar dos operaciones básicas: lectura y escritura. En la lectura, el dispositivo de memoria debe recibir una dirección de la posición de la que se quiere extraer la información depositada previamente. En la escritura, además de la dirección, se debe suministrar la información que se desea grabar.

Existen muchos tipos de memorias:

- Memoria ROM (Read Only Memory).

Esta memoria es de solo lectura, es decir, no se puede escribir en ella. Su información fue grabada por el fabricante al construir el equipo y no desaparece al apagar el computador. Esta memoria es imprescindible para el funcionamiento de la computadora y contiene instrucciones y datos técnicos de los distintos componentes del mismo.

- Memoria RAM (Random Access Memory).

Esta memoria permite almacenar y leer la información que la CPU necesita mientras está ejecutando un programa. Además, almacena los resultados de las operaciones efectuadas por ella.



Este almacenamiento es temporal, ya que la información se borra al apagar la computadora. La memoria RAM se instala en los zócalos que para ello posee la placa base.

- Memoria cache o RAM cache.

Un caché es un sistema especial de almacenamiento de alta velocidad. Puede ser tanto un área reservada de la memoria principal como un dispositivo de almacenamiento de alta velocidad independiente. Hay dos tipos de caché frecuentemente usados en las computadoras personales: memoria caché y caché de disco. Una memoria caché, llamada también a veces almacenamiento caché o RAM caché, es una parte de memoria RAM estática de alta velocidad (SRAM) más que la lenta y barata RAM dinámica (DRAM) usada como memoria principal. El caché de disco trabaja sobre los mismos principios que la memoria caché, pero en lugar de usar SRAM de alta velocidad, usa la convencional memoria principal. (Serrano, 2012)

## Intercambio

El intercambio consiste en llevar cada proceso completo a memoria, ejecutarlo durante cierto tiempo y después regresarlo al disco. Los procesos inactivos mayormente son almacenados en disco, de tal manera que no se ocupan memoria cuando no se están ejecutando. El intercambio de información es una estrategia y comúnmente más simple para lidiar con la sobrecarga de memoria. (Tanenbaum, 2009)

## Memoria Virtual

La memoria virtual al igual que el intercambio es una estrategia para lidiar con la sobrecarga de memoria esta estrategia permite que cada proceso se comporte como si tuviéramos memoria ilimitada a su disposición. Para lograr esto el sistema operativo crea por cada proceso un espacio de direcciones virtual, o conocida como memoria virtual, en disco. Parte de la memoria virtual se trae a memoria principal real cuando se necesita. De esta forma, muchos procesos pueden compartir una cantidad relativamente pequeña de memoria principal. (Stalling, 2005)

## Administración de memoria

Cuando la memoria se asigna en forma dinámica, el sistema operativo debe administrarla. En términos generales, hay dos formas de llevar el registro del uso de la memoria: mapas de bits y listas libres.

- **Administración de memoria con mapas de bits.**

Con los mapas de bits, la memoria se divide en unidades de asignaciones tan pequeñas como unas cuantas palabras y tan grandes como varios kilobytes. Un mapa de bits proporciona una manera simple de llevar el registro de las palabras de memoria en una cantidad fija de memoria, debido a que el tamaño del mapa de bits solo depende del tamaño de la memoria y el tamaño de la unidad de asignación. El problema principal es, el proceso de buscar en un mapa de bits una serie de cierta longitud es una operación lenta, debido a que el administrador de memoria debe de buscar en el mapa de bits una serie de  $k$  bits consecutivos con el valor 0 en el mapa de bits.



- **Administración de memoria con listas ligadas.**

Cada entrada de la lista especifica un hueco (H) o un proceso (P), la dirección donde comienza, su longitud y un apuntador a la siguiente entrada. La lista de segmentos está ordenada por direcciones. Este orden tiene la ventaja de que al terminar o intercambiar un proceso, la actualización de la lista es directa. Cuando los procesos y los huecos se mantienen en una lista ordenada por direcciones, se pueden utilizar diversos algoritmos para asignar la memoria para un proceso de reciente creación o intercambio. (Anonimo, 2014)

## Gestión de memoria

La gestión de memoria representa un vínculo delicado entre el rendimiento (tiempo de acceso) y la cantidad (espacio disponible). Siempre se busca obtener el mayor espacio disponible en la memoria, pero pocas veces existe la predisposición para comprometer el rendimiento. La gestión de memoria también debe realizar las siguientes funciones:

- Permitir que la memoria se comparta (en sistemas de multiprocesos).
- Asignar bloques de espacio de memoria a distintas tareas.
- Proteger los espacios de memoria utilizados (por ejemplo, evitar que un usuario modifique una tarea realizada por otro usuario).
- Optimizar la cantidad de memoria disponible, específicamente a través de sistemas de expansión de memoria. (Anonimo, Kioskea.net, 2014)

La gestión de memoria debe satisfacer los siguientes requisitos:

## Reubicación

- Los programadores no saben dónde estará el programa en memoria cuando se ejecute.
- Mientras el programa se ejecuta, puede ser movido al disco y devuelto a memoria principal en una posición diferente (reubicado).
- Se deben traducir las referencias a memoria del código a las direcciones físicas reales.

## Protección

Los procesos no deberían ser capaces de referenciar el espacio de memoria de otros procesos sin permiso.

- Es imposible comprobar las direcciones absolutas de los programas puesto que éstos pueden ser reubicados.
- Deben ser traducidas durante la ejecución.
- El sistema operativo no puede anticipar todas las referencias de memoria que un programa puede generar.

## Compartición

- Permitir a varios procesos acceder a la misma zona de memoria.
- Es mejor permitir a cada proceso (persona) acceso a la misma copia del programa que tener cada uno su copia individual.

## Organización Lógica

- Los programas son escritos en módulos.
- Los módulos se pueden escribir y compilar por separado.



- A los módulos se les puede dar diferente grado de protección (sólo lectura, sólo ejecución).
- Módulos compartidos.

#### Organización física

- La memoria disponible para un programa y sus datos puede ser insuficiente.
- El solapamiento permite asignar la misma zona de memoria a diferentes módulos.
- El programador no sabe cuánto espacio habrá disponible. (Stalling, 2005)

#### Partición

Partición es el nombre genérico que recibe cada división presente en una sola unidad física de almacenamiento de datos. Toda partición tiene su propio sistema de archivos (formato); generalmente, casi cualquier sistema operativo interpreta, utiliza y manipula cada partición como un disco físico independiente, a pesar de que dichas particiones estén en un solo discofísico.

#### Particionamiento Fijo

- ❖ El uso de la memoria principal es ineficiente.
- ❖ Un programa, no importa como de pequeño sea, ocupa una partición entera. Esto se conoce como fragmentación interna.

### **Particiones del mismo tamaño**

- ❖ Cualquier proceso de tamaño menor o igual al de una partición puede ser cargado en una partición disponible.
- ❖ Si todas las particiones están ocupadas, el S.O. puede mover a disco un proceso de una partición.
- ❖ Un programa puede no caber en una partición.
- ❖ El programador debe diseñar el programa con overlays.

### [Algoritmo de Ubicación con Particiones](#)

#### Particiones del mismo tamaño

- Como todas las particiones tienen el mismo tamaño, no importa qué partición asignar.

#### Particiones de diferente tamaño

- Se puede asignar a cada proceso la partición más pequeña en la que cabe - cola para cada partición.
- Los procesos se asignan de manera que se minimiza la memoria desperdiciada de una partición. (Stalling, 2005)

### [Particionamiento Dinámico](#)

Las particiones se crean de forma dinámica, de tal forma que cada proceso se cargue en una partición del mismo tamaño que el procesador.

- El tamaño y el número de particiones es variable.



- Al proceso se le asignar exactamente la cantidad de memoria que necesita.
- Aparecen huecos en la memoria. Esto se conoce como fragmentación externa.
- Se debe realizar una compactación para desplazar a los procesos de forma que esté juntos y todo el espacio libre esté en un solo bloque.

#### Particionamiento Dinámico Algoritmo de Ubicación

El sistema operativo debe decidir qué bloque libre asignar a un proceso. Algoritmo del mejor ajuste (best fit).

- Elige el bloque que tiene el tamaño más cercano al solicitado.
- Peor rendimiento de todos.
- Como se busca el bloque más pequeño por proceso, se produce el menor volumen de fragmentación, pero hay que compactar más a menudo.

Algoritmo del primer ajuste (first fit)

- Es el más rápido.
- Puede haber muchos procesos cargados en la zona inicial de la memoria, que debe ser examinada cuando se busca un bloque libre.

Algoritmo del siguiente ajuste (Next-fit)

- A menudo se asigna un bloque de memoria en la última parte de la memoria donde está el mayor bloque.
- El mayor bloque de memoria se parte en pequeños bloques.
- Se necesita compactar para obtener un bloque grande en la última zona final memoria.

## Buddy System

La memoria disponible completa es tratada como un bloque individual de 2 U. Si una petición de tamaño s es tal que  $2 < s \leq 2$ , U-1 U se le asigna el bloque completo

- Si no, el bloque se divide en dos trozos iguales (buddies).
- El proceso continúa hasta generar el bloque más pequeño que es mayor o igual a s.

## Direcciones

### Lógicas

- Referencias a posiciones de memoria independientes de la asignación vigente de datos en memoria.
- La traducción se realiza a dirección física.

### Relativas

- Las direcciones se expresan como posiciones relativas a algún punto conocido.

### Físicas

- Es la dirección absoluta o ubicación real en memoria principal.

## Paginación

- Partición de la memoria en pequeños pedazos del mismo tamaño (chunks) y dividir cada proceso en trozos del mismo tamaño.
- Los trozos (chunks) de un proceso se llaman páginas y los de la memoria se llaman marcos de página (frames).
- El sistema operativo mantiene una tabla de página para cada proceso.



- Contiene la ubicación del marco de página (frame) de cada página del proceso.
- La dirección de memoria consiste en un número de página y un desplazamiento (offset) dentro de la página. (Stalling, 2005)

### Segmentación

- Todos los segmentos de todos los programas no tienen por qué ser del mismo tamaño.
- Hay un tamaño máximo de segmento.
- El direccionamiento consta de dos partes.
- Un número de segmento y un desplazamiento dentro de éste (offset).
- Como los segmentos no son iguales, la segmentación es similar al particionamiento dinámico. (Stalling, 2005)

## UNIDAD IV:

# MECANISMOS DE BLOQUEOS IRREVERSIBLES ENTRE PROCESOS

### Bloqueo irreversible.

Un conjunto de procesos cae en bloqueo irreversibles cada proceso del conjunto está esperando un suceso que otro proceso puede causar, hay bloqueos irreversibles a nivel proceso o a nivel hardware.

Puede ser:

Hardware: un dispositivo de entrada y salida.

Software: información bases de datos archivos registros. Existen recursos de tipo:

- Expropiativo que consta en se le puede quitar el recurso a quien lo tiene sin hacerle daño ya que cuando un proceso este levantado este no podrá terminarlo hasta que el otro termine su ejecución. Ejemplo la memoria se puede expropiar respaldándose en disco a quien la tenía.
- No expropiativo este recurso se le puede quitar el recurso y no falla el proceso que seria que cuando un proceso está levantado y llega otro proceso este automáticamente lo termina aunque el otro no haya concluido con su ejecución. Un ejemplo de esto puede es una impresora o un escáner se puede quitar el recurso y no pasa nada.

Normalmente los bloqueos irreversibles se dan en los recursos no expropiables.



La secuencia de uso de un recurso. Es la siguiente.

- Se solicita el recurso.
- Si está disponible.
- Lo usa.
- Lo libera.

Si no está disponible:

- Devuelve un error.
- Se bloquea.
- Lo intenta más tarde.

Características del bloqueo irreversible.

- para que exista un bloqueo irreversible los procesos solo tienen un sub proceso.
- No hay interrupciones que activen a un bloqueo irreversible.

Condiciones para que exista un bloqueo irreversible.

## Condiciones para manejar el bloqueo irreversible

1.-Exclusión mutua.

Cada recurso está asignado únicamente a un solo proceso o está Disponible.

Un recurso está asignado a un proceso libre.

- Retención y espera.

Si ya tienen procesos pueden pedir más.

- De no expropiación.
- No pueden arrebatarseles, antes deben ser liberados.
- De espera circular.

Cadena circular de dos o más procesos. Exclusión mutua.

No asignar en forma exclusiva todos los recursos, asignar un recurso hasta que sea estrictamente necesario, que la cantidad de procesos que soliciten cierto recurso sea la menor posible.

## 2.-Retencion y espera.

Una segunda técnica es la detección y recuperación. Cuando se usa esta técnica, el sistema no hace otra cosa que no sea vigilar las peticiones y liberaciones de recursos. Cada vez que un recurso se solicita o libera, se actualiza el grafo de recursos, y se determina si contiene algún ciclo. Si se encuentra uno, se termina uno de los procesos del ciclo. Si esto no rompe el bloqueo mutuo, se termina otro proceso, continuando así hasta romper el ciclo. Un método un tanto más burdo consiste en no mantener siquiera el grafo de recursos, y en vez de ello verificar periódicamente si hay procesos que hayan estado bloqueados continuamente durante más de, digamos, una hora. A continuación se terminan esos procesos.

La detección y recuperación es la estrategia que a menudo se usa en las macro computadoras, sobre todo los sistemas por lotes en los que terminar y luego reiniciar un proceso suele ser aceptable. Sin embargo, se debe tener cuidado de restaurar todos los archivos modificados a su estado original, y revertir todos los demás efectos secundarios que pudieran haber ocurrido.



### 3.-De no expropiación.

No es posible quitarle por la fuerza a un proceso los recursos que le fueron otorgados previamente. El proceso que los tiene debe liberarlos explícitamente.

### 4.-Espera circular.

Debe haber una cadena circular de dos o más procesos, cada uno de los cuales están esperando un recurso retenido por el siguiente miembro de la cadena. Deben estar presentes estas cuatro condiciones para que ocurra un bloqueo mutuo. Si una o más de estas Condición de espera circular. Es que un proceso solo pueda tener un recurso a la vez, al solicitar uno, que libere otro (imposible), numerar los recursos, y que los procesos al solicitarlos los vayan tomando en orden ascendente. Otros aspectos a tomar en cuenta:

- Bloqueos de dos fases sobre todo en bases de datos.
- Primera fase bloqueos de registros hasta que pueda bloquearlos todos.
- Segunda fase actualización.
- Bloqueos irreversibles que no son por recursos.
- Puede ser solo entre procesos.
- Que uno espere a que otro lo despierte.
- Caso productor consumidor.
- Inanición.

Puede haber proceso que no estén en un bloqueo irreversible y aun así casi nunca son atendidos por lo que mueren de inanición una solución es usar un algoritmo d calendarización que lo evite.

## Estrategias para el bloqueo irreversibles

El bloqueo mutuo se evitaba no imponiendo reglas arbitrarias a los procesos sino analizando con detenimiento cada petición de recurso para ver si se puede satisfacer sin peligro. Surge la pregunta: ¿hay algún algoritmo que siempre pueda evitar el bloqueo mutuo tomando la decisión correcta en todos los casos? La respuesta es que sí se puede evitar el bloqueo mutuo, pero sólo si se cuenta con cierta información por adelantado. En esta sección examinaremos formas de evitar los bloqueos mutuos mediante una asignación cuidadosa de los recursos y una de las estrategias serían las siguientes:

- 1.- Ignorar el problema.
- 2.- Denar que sucedan, detectarlos y recuperarse de ellos.
- 3.- Evitar que sucedan, con una asignación cuidadosa del recurso.
- 4.- Prevención, anulado una de la cuatro condiciones para evitar un bloqueo irreversible.

### Algoritmo de aveSTRUZ

Es la estrategia más sencilla es el algoritmo del aveSTRUZ: meter la cabeza en la arena y pretender que el problema no existe. La gente reacciona a esta estrategia de diversas maneras. Los matemáticos la encuentran totalmente inaceptable y dicen que los bloqueos mutuos deben prevenirse a toda costa.

Los ingenieros preguntan con qué frecuencia se espera que se presente el problema, qué tan seguido se cae el sistema por otras razones, y qué tan grave es un bloqueo mutuo. Si ocurren bloqueos mutuos una vez cada 50 años en promedio, pero las caídas del sistema debido a fallas de hardware, errores del compilador y defectos del sistema operativo ocurren una vez al mes, la mayoría de



los ingenieros no estarían dispuestos a pagar un precio sustancial en términos de reducción del rendimiento o de la comodidad a fin de evitar los bloqueos mutuos.

En un sistema real los bloqueos irreversibles no son tan comunes, en la mayoría de los sistemas operativos pueden tener bloqueos irreversibles que ni siquiera se detectan. Se tiene límite que pueden originar un bloqueo irreversible. Cantidad de procesos y sub procesos que pueden estar activos al mismo tiempo. Número máximo de archivos abiertos.

#### Algoritmo Excepción Dinámica

Detección de bloqueos irreversibles con un recurso de cada tipo solo hay un recurso de cada tipo, un sistema con estas condiciones podría tener un escáner o un DVD ROM o impresora. Detección de bloqueos irreversibles con múltiples recursos si hay varias copias los recursos se requiere un enfoque para detectar se basa en una comparación de vectores.

#### Prevención

Evitar los bloqueos irreversibles son procesos irreales ya que consisten en evitar que al menos uno de los 4 básicas no se cumpla.

El espacio de intercambio en disco.

Windows y Linux los ignoran, prefieren un bloqueo irreversible de vez en cuando a que haya limitantes con los recursos.

Detección de bloqueos irreversibles y recuperación posterior.

No se intenta prevenir, solo esperar a que suceda, detectarlo y recuperarse del bloqueo.

Detección de bloqueo irreversible con un recurso de cada tipo.

-Solo hay un recurso de cada tipo.

-Si hay uno o más ciclos, existe un bloqueo irreversible si no, no lo hay

Como recuperarse de un bloqueo irreversible.

#### 1.- Mediante la expropiación.

Consiste en quitarle el recurso un proceso y después reanudarlo. Depende del recurso, de preferencia un recurso expropiable.

#### 2.- Por eliminación de procesos.

Elimina un proceso, sino lo rompe, se elimina otro y así sucesivamente.

Podría ser uno que no esté en el ciclo, uno de baja prioridad, uno que pueda reiniciarse.

#### 3.- Mediante reversión.

- Estableciendo puntos de verificación.
- Deben ser archivos nuevos constantemente creados.
- El punto de verificación contiene, la memoria, el estado de los recursos y procesos.

Al detectar un bloqueo irreversible se busca un punto de verificación antes de haber solicitado el recurso para reanudar a partir de ahí.



Como evitar los bloqueos irreversibles.

- Objetivo evitar los bloqueos irreversibles, mediante la asignación cuidadosa de recursos.
- Sin tener que darles todos los recursos que requiere, como el método anterior.

Estados seguros e inseguros.

Estado seguro.

Cuando no ha caído un bloqueo irreversible, existe algún orden de calendarización en el cual todos los procesos pueden ejecutarse hasta terminar. En un estado seguro no se ofrece garantía que los procesos terminen, un estado inseguro necesariamente es un bloqueo irreversible.

Algoritmo del banquero para un solo recurso:

- Se examina cada solicitud al momento que se hace y se analiza si esto llevará a un estado seguro, si es así lo concede si no lo manda a espera.

Para ver si su estado es seguro.

- Checa si hay suficientes recursos para satisfacer el proceso.
- Verifica que proceso está más cercano al límite.

Al igual que el caso de un solo recurso, se requiere conocer las necesidades totales de los procesos antes de ejecutarse, requieren que la cantidad de procesos sea fija, los recursos pueden aparecer y desaparecer en cualquier momento, no se usa en la práctica.

Prevención de bloqueos irreversibles.

Como los mecanismos de bloqueos irreversibles son poco irreales y es difícil caer en un bloqueo irreversible, se opta por la prevención.

Consiste en evitar que al menos una de las cuatro condiciones básicas no se cumpla.



## **UNIDAD V:**

# **ADMINISTRACIÓN DE LOS ARCHIVOS EN LOS DIFERENTES SISTEMAS OPERATIVOS**

Se tienen tres requerimientos esenciales para el almacenamiento de información a largo plazo:

1. Debe ser posible almacenar una cantidad muy grande de información.
2. La información debe sobrevivir a la terminación del proceso que la utilice.
3. Múltiples procesos deben ser capaces de acceder a la información concurrentemente.

Los archivos son unidades lógicas de información creada por los procesos. Los procesos pueden leer los archivos existentes y crear otros si es necesario. La información que se almacena en los archivos debe ser persistente, es decir, no debe ser afectada por la creación y terminación de los procesos.

## **Archivos**

Desde el punto de vista que tiene el usuario acerca del sistema de archivos.

### **Nomenclatura de archivos**

Los archivos son un mecanismo de abstracción. Proporcionan una manera de almacenar información en el disco y leerla después. Pero para ello el sistema debe de poder identificarlos, todos los sistemas operativos actuales permiten cadenas de una a ocho letras, dígitos y caracteres especiales como nombres de archivos legales.

Después del nombre del archivo lleva un punto y algunas letras a las cuales se les conocen como extensión del archivo y por lo general indica algo acerca de su naturaleza.

Algunas de las extensiones de archivos más comunes y sus significados son las siguientes.

Extensión	Significado
archivo.bak	Archivo de respaldo
archivo.c	Programa fuente en C
archivo.gif	Imagen en Formato de Intercambio de Gráficos de CompuServe
archivo.hlp	Archivo de ayuda
archivo.html	Documento en el lenguaje de Marcación de Hipertexto de www
archivo.jpg	Imagen fija codificada con el estándar JPEG
archivo.mp3	Música codificada en formato de audio MPEG capa 3
archivo.mpg	Película codificada con el estándar MPEG
archivo.o	Archivo objeto
archivo.pdf	Archivo en formato de documento portable
archivo.ps	Archivo de PostScript
archivo.tex	Entrada para el programa formateador TEX
archivo.txt	Archivo de texto general
archivo.zip	Archivo comprimido

En algunos sistemas (como UNIX) las extensiones de archivo son solo convenciones y no son impuestas por los sistemas operativos. Por lo contrario Windows está consciente de las extensiones y les asigna significado.

### Estructura de archivos

Tres posibilidades comunes de describir la estructura de los archivos es:



1. El sistema operativo no sabe, ni le importa, qué hay en el archivo. Todo lo que ve son bytes. Tanto UNIX, MS-DOS y Windows utilizan esta metodología. El sistema operativo no ayuda pero no estorba.
- 2 En otro modelo un archivo es una secuencia de registros de longitud fija, cada uno con uno con cierta estructura interna. La idea de esto es que la operación de lectura devuelva un registro y la operación de escritura sobrescriba o agregue un registro.
3. En otra organización, un archivo consiste de un árbol de registros, donde no todos son de la misma longitud; cada uno de ellos contiene un campo llave en una posición fija dentro del registro. El árbol se organiza con base en el campo llave para permitir una búsqueda rápida por una llave específica.

### Tipos de archivos

**Archivos regulares** los que contienen información del usuario. Los directorios son sistemas de archivos para mantener la estructura del sistema de archivos. Por lo general estos archivos son ASCII (consisten en líneas de texto) o binarios.

**Archivos especiales de caracteres** se relacionan con la entrada/salida y se utilizan para modelar dispositivos de entrada/salida en serie, tales como terminales, impresoras y redes.

**Archivos especiales de bloques** se utilizan para modelar discos.

### Acceso a archivos

Existen dos formas de acceso a los archivos.

1. **Archivos de acceso secuencial:** en estos sistemas, un proceso puede leer todos los bytes o registros en un archivo en orden, empezando desde el principio, pero no puede saltar alguno y leerlos fuera de orden.

2. **Archivos de acceso aleatorio:** son los archivos cuyos bytes o registros se pueden leer en cualquier orden. Estos archivos son esenciales para muchas aplicaciones, como los sistemas de base de datos.

### Atributos de archivos

Todo archivo tiene un nombre y sus datos, a estos datos se les llamarán atributos del archivo o metadatos. La lista de atributos varía considerablemente de un sistema a otro. A continuación se muestran algunas posibilidades de atributos.

Atributo	Significado
Protección	Quién tiene acceso al archivo y en qué forma
Contraseña	Contraseña necesaria para acceder al archivo
Creador	ID de la persona que creó el archivo
Propietario	El propietario actual
Bandera de sólo lectura	0 para lectura/escritura; 1 para sólo lectura
Bandera oculto	0 para normal; 1 para que no aparezca en los listados
Bandera del sistema	0 para archivos normales; 1 para archivo del sistema
Bandera de archivo	0 si ha sido respaldado; 1 si necesita respaldarse
Bandera ASCII/binario	0 para archivo ASCII; 1 para archivo binario
Bandera de acceso aleatorio	0 para solo acceso secuencial; 1 para acceso aleatorio
Bandera temporal	0 para normal; 1 para eliminar archivos al salir del proceso
Banderas de bloqueo	0 para desbloqueado; distinto de 0 para bloqueado
Longitud de registro	Número de bytes en un registro



Posición de la llave	Desplazamiento de la llave dentro de cada registro
Hora de creación	Fecha y hora en la que se creó el archivo
Hora de último acceso	Fecha y hora en que se accedió al archivo por última vez
Hora de la última modificación	Fecha y hora en que se modificó por última vez el archivo
Tamaño actual	Número de bytes en el archivo
Tamaño máximo	Número de bytes hasta donde puede crear el archivo
Longitud de llave	Número de bytes en el campo llave

### Operaciones de archivos

Distintos sistemas proveen diferentes operaciones para permitir el almacenamiento y la recuperación. A continuación se muestra las llamadas al sistema más comunes.

1. Crear. El archivo se crea sin datos. El propósito de la llamada es anunciar la llegada del archivo y establecer algunos de sus atributos.
2. Borrar. Cuando el archivo ya no se necesita, se tiene que eliminar para liberar espacio en el disco.
3. Abrir. El propósito de la llamada a open es permitir que el sistema lleve los atributos y la lista de direcciones de disco a memoria principal para tener un acceso rápido a estos datos en llamadas posteriores.
4. Cerrar. Cuando terminan todos los accesos, los atributos y las direcciones de disco ya no son necesarias, por lo que el archivo se debe cerrar para liberar espacio en la tabla interna.
5. Leer. El llamador debe especificar cuántos datos se necesitan y también debe proporcionar un buffer para colocarlos.

6. Escribir. Los datos se escriben en el archivo otra vez por lo general en la posición actual. Si la posición actual es al final del archivo, aumenta su tamaño.
7. Buscar. Reposiciona el apuntador del archivo en una posición específica del archivo. Una vez que se completa esta llamada se pueden leer o escribir datos en esa posición.
8. Renombrar. Con frecuencia ocurre que un usuario necesita cambiar el nombre de un archivo existente.

Cada archivo abierto tiene asociados varios elementos de información.

1. Puntero al archivo: el sistema debe seguir la pista de la última posición de lectura/escritura con un puntero a la posición actual en el archivo. Este puntero es exclusivo para cada proceso que está trabajando con el archivo, por lo que debe mantenerse a parte de los atributos del archivo en disco.
2. Contador de aperturas del archivo: este contador sigue la pista al número de aperturas y cierres, y llega a cero después del último cierre.
3. Ubicación del archivo en disco: la información necesaria para localizar el archivo en disco se mantiene en la memoria para no tener que leerla del disco en cada operación.

## Directarios

Algunos sistemas almacenan miles de archivos de cientos de gigabytes de disco. Para administrar todos estos datos se necesitan organizar. Esta organización por lo regular se efectúa en dos partes:

1. El sistema de archivos se divide en particiones. Típicamente cada disco de un sistema contiene al menos una partición. A veces se usan particiones para contar con varias áreas independientes dentro de un disco.



2. Cada partición contiene información acerca de los archivos que hay en ella. Esta información se mantiene como entradas de un directorio de dispositivo o tabla de contenido del volumen. El directorio del dispositivo registra información como nombre, ubicación, tamaño y tipo de todos los archivos de esa partición.

#### **Operaciones que se realizan con un directorio**

- Buscar un archivo.
- Crear un archivo.
- Eliminar un archivo.
- Cambiar el nombre de un archivo.
- Recorrer el sistema de archivos.
- Listar un directorio.

## **Sistemas de directorios de un solo nivel**

La forma más simple de un sistema de directorios es tener un directorio que contenga todos los archivos, algunas veces se le llama directorio raíz. A menudo se utilizan en dispositivos incrustados simples como teléfonos, cámaras digitales y algunos reproductores de música portátil.

#### **Sistemas de directorios jerárquicos**

Con este esquema, puede haber tantos directorios como se necesite para agrupar los archivos en formas naturales. La capacidad de los usuarios para crear un número arbitrario de subdirectorios provee una poderosa herramienta de estructuración para que los usuarios organicen su trabajo. Por esta razón, casi todos los sistemas de archivos modernos se organizan de esta manera.

## Nombre de rutas

Cuando el sistema de archivos está organizado como un árbol de directorios, se necesita cierta forma de especificar los nombres de los archivos. Por lo general se utilizan dos métodos distintos. En el primer método, cada archivo recibe un nombre de ruta absoluto que consiste en la ruta desde el directorio raíz al archivo. Los nombres de ruta absolutos siempre empiezan en el directorio raíz y son únicos. Sin importar cuál carácter se utilice, si el primer carácter del nombre de la ruta es el separador, entonces la ruta es absoluta.

El otro tipo de nombre es el nombre de ruta relativa. Éste se utiliza en conjunto con el concepto del directorio de trabajo (también llamado directorio actual). Todos los nombres de las rutas que no empiecen en el directorio raíz se toman en forma relativa al directorio de trabajo.

Cada proceso tiene su propio directorio de trabajo, por lo que cuando éste cambia y después termina ningún otro proceso se ve afectado y no quedan rastros del cambio en el sistema de archivos.

1. **Create.** Se crea un directorio. Está vacío, excepto por punto y punto, que el sistema coloca ahí de manera automática.
2. **Delete.** Se elimina un directorio. Se puede eliminar sólo un directorio vacío.
3. **Opendir.** Los directorios se pueden leer. Antes de poder leer un directorio se debe abrir, en forma análoga al proceso de abrir y leer un archivo.
4. **Closedir.** Cuando se ha leído un directorio, se debe cerrar para liberar espacio en la tabla interna.
5. **Readdir.** En contraste, readdir siempre devuelve una entrada en formato estándar, sin importar cuál de las posibles estructuras de directorio se utilice.
6. **Rename.** En muchos aspectos, los directorios son sólo como archivos y se les puede cambiar el nombre de la misma forma que a los archivos.
7. **Link.** La vinculación (ligado) es una técnica que permite a un archivo aparecer en más de un directorio.



8. Unlink. se elimina una entrada de directorio. Si el archivo que se va a desvincular sólo está presente en un directorio, se quita del sistema de archivos. Si está presente en varios directorios, se elimina sólo el nombre de ruta especificado.

## Implementación de sistemas de archivos

Ahora se cambiara del punto de vista que tiene el usuario acerca del sistema de archivos, al punto de vista del que lo implementa.

### Distribución del sistema de archivos

A menudo el sistema de archivos contiene algunos de los siguientes elementos.

El primero es el superbloque. Contiene todos los parámetros clave acerca del sistema de archivos y se lee en la memoria cuando se arranca la computadora o se entra en contacto con el sistema de archivos por primera vez. La información típica en el superbloque incluye un número mágico para identificar el tipo de sistema de archivos, el número de bloque que contiene el sistema de archivos y otra información administrativa clave.

A continuación podría venir información acerca de los bloques libres en el sistema de archivos, por ejemplo en la forma de un mapa de bits o una lista de apuntadores. Éste podría ir seguida de los nodos i, un arreglo de estructuras de datos, uno por archivo, que indica todo acerca del archivo. Despues de eso podría venir el directorio raíz, que contiene la parte superior del árbol del sistema de archivos. Por último, el resto del disco contiene todos los otros directorios y archivos.

## Implementación de archivos

Probablemente la cuestión más importante al implementar el almacenamiento de archivos sea mantener un registro acerca de qué bloques de discos van con cual archivo. Se utilizan varios métodos en distintos sistemas operativos.

### Asignación contigua

El esquema de asignación más simple es almacenar cada archivo como una serie contigua de bloques de disco. La asignación de espacio en disco contiguo tiene dos ventajas significativas. En primer lugar es simple de implementar, ya que llevar un registro de la ubicación de los bloques de un archivo se reduce a recordar dos números: la dirección de disco del primer bloque y el número de bloque en el archivo.

En segundo lugar, el rendimiento de la lectura es excelente debido a que el archivo completo se puede leer del disco en una sola operación. Por desgracia la asignación contigua también tiene una desventaja: con el transcurso del tiempo, los discos se fragmentan. Cuando se quita un archivo los bloques se liberan naturalmente, dejando una serie de bloques libres en el disco. Hay una situación en la que es factible la asignación contigua y de hecho, se utiliza ampliamente: en los CD-ROMs.

### Asignación de lista enlazada (ligada)

El segundo método para almacenar archivos es mantener cada uno como una lista enlazada de bloques de disco. La primera palabra de cada bloque se actualiza como apuntador al siguiente. El resto del bloque es para los datos. A diferencia de la asignación contigua, este método se puede utilizar cada bloque del disco.



No se pierde espacio debido a la fragmentación del disco. Además, para la entrada del directorio sólo le basta con almacenar la dirección de disco del primer bloque. Por otro lado, aunque la lectura secuencial a un archivo es directa, el acceso aleatorio es en extremo lento.

### **Asignación de lista enlazada utilizando una tabla de memoria**

Ambas desventajas de la asignación de lista enlazada se puede eliminar si tomamos la palabra del apuntador de cada bloque de disco y la colocamos en una tabla en memoria. Dicha tabla en memoria principal se conoce como FAT (File Allocation Table, tabla de asignación de archivos).

Utilizando esta organización, el bloque completo está disponible para los datos. Además, el acceso aleatorio es mucho más sencillo. La principal desventaja de este método es que toda la tabla debe estar en memoria todo el tiempo para que funcione.

### **Nodos-i**

Nuestro último método para llevar un registro de qué bloques pertenecen a cuál archivo es asociar con cada archivo una estructura de datos conocida como nodo-i (nodo-índice), la cual lista los atributos y las direcciones de disco de los bloques del archivo. La gran desventaja de este esquema en comparación con los archivos vinculados que utilizan una tabla en memoria, es que el nodo-i necesita estar en memoria sólo cuando está abierto el archivo correspondiente.

En contraste, el esquema del nodo-i requiere un arreglo en memoria cuyo tamaño sea proporcional al número máximo de archivos que pueden estar abiertos a la vez.

## Implementación de directorios

Antes de poder leer un archivo éste debe abrirse. Cuando se abre un archivo, el sistema operativo utiliza el nombre de la ruta suministrado por el usuario para localizar la entrada de directorio. Esta entrada provee la información necesaria para encontrar los bloques de disco.

Cada sistema de archivo mantiene atributos de archivo, como el propietario y la hora de creación de cada archivo, debiendo almacenarse en alguna parte. Una posibilidad obvia es almacenarlos directamente en la entrada de directorio.

Para los sistemas que utilizan nodos-i, existe otra posibilidad para almacenar los atributos en los nodos-i, en vez de hacerlo en la entrada de directorio. En este caso, la entrada de directorio puede ser más corta: sólo un nombre de archivo y un número de nodo-i.

Casi todos los sistemas operativos modernos aceptan nombres de archivos largos, con longitud variable, el esquema más simple para almacenarlos es establecer un límite en la longitud del nombre de archivo que por lo general es de 255 caracteres. Este esquema es simple pero desperdicia mucho espacio de directorio, ya que pocos archivos tienen nombres tan largos.

Una alternativa es renunciar a la idea de que todas las entradas de directorio sean del mismo tamaño. Con este método, cada entrada de directorio contiene una porción fija, que por lo general empieza con la longitud de la entrada y después va seguida de datos con un formato fijo, que comúnmente incluyen el propietario, la hora de creación, información de protección y demás atributos.

Una desventaja de este método es que cuando se elimina un archivo, en su lugar queda un hueco de tamaño variable dentro del directorio, dentro del cual el siguiente archivo a introducir puede que no quepa.



Otra manera de manejar los nombres de longitud variable es hacer que las mismas entradas de directorio sean de longitud fija y mantener los nombres de los archivos juntos en un heap al final del directorio. Este método tiene la ventaja de que cuando se remueva una entrada, el siguiente archivo a introducir siempre cabrá ahí.

En todos los diseños mostrados hasta ahora se realizan búsquedas lineales en los directorios de principio a fin cuando hay que buscar el nombre un archivo. Para los directorios en extremo largos, la búsqueda lineal puede ser lenta. Una manera de acelerar la búsqueda es utilizar una tabla de hash en cada directorio. El uso de una tabla de hash tiene la ventaja de que la búsqueda es mucho más rápida, pero la desventaja de una administración más compleja.

Una manera distinta de acelerar la búsqueda en directorios extensos es colocar en caché los resultados de las búsquedas. Antes de iniciar una búsqueda, primero se realiza una verificación para ver si el nombre del archivo está en la caché. De ser así se puede localizar de inmediato. Desde luego el uso de la caché sólo funciona si un número relativamente pequeño de archivos abarcan la mayoría de las búsquedas.

## Archivos compartidos

Cuando hay varios usuarios trabajando en conjunto en un proyecto, a menudo necesitan compartir archivos. Como resultado, con frecuencia es conveniente que aparezca un archivo compartido en forma simultánea en distintos directorios que pertenezcan a distintos usuarios. La conexión entre el directorio x y el archivo compartido se conoce como un vínculo (liga). El sistema de archivos en sí es ahora un gráfico acíclico dirigido (Directed Acyclic Graph, DAG) en vez de un árbol.

Pero también introduce ciertos problemas. Para empezar, si los directorios en realidad contienen direcciones de disco, entonces habrá que realizar una copia de las direcciones de disco en el directorio x cuando se ligue el archivo.

Este problema se puede resolver de dos formas. En la primera solución, los bloques de disco no se listan en los directorios, sino que en una pequeña estructura de datos asociada con el archivo en sí. Entonces, los directorios apuntarían sólo a la pequeña estructura de datos.

En la segunda solución, x se vincula a uno de los archivos de y haciendo que el sistema cree una archivo, de tipo link introduciendo ese archivo en el directorio de

x. El nuevo archivo contiene sólo el nombre de la ruta del archivo el cual está vinculado. Cuando x lee del archivo vinculado, el sistema operativo ve que el archivo del que se están leyendo datos es de tipo link, busca el nombre del archivo y lee el archivo. A este esquema se le conoce como vínculo simbólico (liga simbólica).

Al crear un vínculo no se cambia la propiedad, sino incrementa la cuenta de vínculos en el nodo-i, por lo que el sistema sabe cuántas entradas de directorio actualmente apuntan al archivo. Al eliminar un vínculo simbólico, el archivo no se ve afectado.

El problema con los vínculos simbólicos es el gasto adicional de procesamiento requerido. Se debe leer el archivo que contiene la ruta, después ésta se debe analizar sintácticamente y seguir, componente por componente, hasta llegar al nodo-i. Toda esta actividad puede requerir una cantidad considerable de acceso adicional al disco.

#### Sistemas de archivos estructurados por registro

LFS (Log-structured File System, Sistema de archivos estructurado por registro). La idea básica es estructurar todo el disco como un registro. De manera periódica, y cuando haya una necesidad especial para ello, todas las escrituras pendientes que se colocaron en el búfer en memoria se recolectan en un solo segmento y se



escriben en el disco como un solo segmento continuo al final del registro. Por lo tanto, un solo segmento puede contener nos-i, bloques de directorio y bloques de datos, todos mezclados entre sí.

Para que sea posible encontrar nodos-i, se mantiene un mapa de nodos-i, indexados por número-i. La entrada i en este mapa apunta al nodo-i i en el disco. El mapa se mantiene en el disco, pero también se coloca en la caché, de manera que las partes más utilizadas estén en memoria la mayor parte del tiempo. (Tanenbaum, 2009)

## Protección

Cuando se guarda información en un sistema de computación una preocupación importante es su protección tanto de daños físicos como de un acceso indebido. La confiabilidad se logra duplicando los archivos.

### Tipos de acceso

Lo que se necesita en un sistema multiusuario es un acceso controlado. Los mecanismos de protección proporcionan un acceso controlado limitando las formas en que se puede acceder a los archivos. Pueden controlarse varios tipos de operaciones distintas como:

- Leer.
- Escribir.
- Ejecutar.
- Anexar.
- Eliminar.
- Listar.

La protección sólo puede proporcionarse en el nivel más bajo.

## Listas y grupos de acceso

La forma más común de abordar el problema de la protección es hacer que el acceso dependa de la identidad del usuario. El esquema más general para implementar el acceso independiente de la identidad es asociar a cada archivo y directorio a una lista de acceso que especifique el nombre del usuario y los tipos de acceso que se permiten a cada usuario.

Para condensar la lista de acceso, muchos sistemas reconocen tres categorías de usuarios en relación con cada archivo:

1. Propietario: el usuario que creó el archivo.
2. Grupo: un conjunto de usuarios que están compartiendo el archivo.
3. Universo: todos los demás usuarios del sistema.

El sistema Unix define tres campos de tres bits cada uno: r w x, donde r controla el acceso de lectura, w el acceso de escritura y x controla la ejecución.

## Implementación del sistema de archivos

El sistema de archivos reside de manera permanente en almacenamiento secundario, cuyo requisito principal es que debe poder contener una gran cantidad de datos permanentemente.

## Estructura del sistema de archivos

Para mejorar la eficiencia de E/S, las transferencias entre la memoria y el disco se efectúan en unidades de bloques. Cada bloque ocupa uno o más sectores. Dependiendo de la unidad de disco, el tamaño de los sectores varía entre 32 bytes y 4096 bytes, aunque por lo regular es de 512 bytes. Los discos tienen dos características importantes que los convierten en un medio cómodo para almacenar muchos archivos.



1. Se puede reescribir en el mismo lugar.
2. Se puede acceder directamente a cualquier bloque de información del disco.

#### Organización del sistema de archivos

Para ofrecer un acceso eficiente y cómodo al disco, el sistema operativo impone en él un sistema de archivos que permita almacenar, encontrar y recuperar con facilidad los datos.

Un sistema de archivos presenta dos problemas de diseño.

1. Definir qué aspecto debe presentar el sistema de archivos a los usuarios.
2. Hay que crear algoritmos y estructuras de datos que establezcan una correspondencia entre el sistema de archivos lógico y los dispositivos de almacenamiento secundario físico.

El sistema de archivos en si generalmente se compone de muchos niveles diferentes, ejemplo:

Programas de aplicación. Sistema de  
archivos lógicos.

Módulo de organización de archivos.

Sistema de archivos básicos.

Control de E/S.

Dispositivos.

El nivel más bajo, el control E/S, consta de drivers o controladores de dispositivos y manejadores de interrupciones para transferir información entre la memoria y el sistema de disco. Se puede considerar un driver de dispositivo como un traductor.

El driver de dispositivo por lo regular escribe patrones de bits específicos en posiciones especiales de la memoria del controlador de E/S para decirle a éste sobre qué posición del dispositivo debe actuar y qué acciones debe emprender. El sistema de archivos básico sólo necesita emitir órdenes genéricas al driver de dispositivo apropiado para leer y escribir bloques físicos en el disco.

El módulo de organización de archivos conoce los archivos y sus bloques lógicos, además de los bloques físicos. Sabiendo el tipo de asignación de archivos empleada y la ubicación del archivo, el módulo de organización de archivos puede traducir las direcciones de bloque lógico en direcciones de bloque físico para que el sistema de archivos básico realice la transferencia. El módulo de organización de archivos también incluye el administrador de espacio libre, que sigue la pista a los bloques no asignados y los proporciona al módulo de organización de archivos cuando se le solicita.

El sistema de archivos lógico emplea la estructura de directorios para proporcionar al módulo de organización de archivos la información que éste necesita, a partir de un nombre de archivo simbólico. El sistema de archivos lógico también se encarga de la protección y la seguridad. (Silberschatz, 1999, págs. 337-372)

## Tipos de sistemas de archivos

### BTRFS

Btrfs (B-tree FS o normalmente pronunciado "Butter FS") es un sistema de archivos copy-on-write anunciado por Oracle Corporation para GNU/Linux. Es un nuevo sistema de archivos con potentes funciones, similares al excelente ZFS de Sun/Oracle. Estas incluyen la creación de instantáneas, striping y mirroring multi-disco (RAID software sin mdadm), sumas de comprobación, copias de seguridad incrementales, y compresión sobre la marcha integrada, que pueden dar un significativo aumento de las prestaciones, así como ahorrar espacio.



## Ext2

Second Extended Filesystem es un consolidado y maduro sistema de archivos para GNU/Linux muy estable. Uno de sus inconvenientes es que no tiene apoyo para el registro (*journaling*). La falta de *registro por diario* («*journaling*») puede traducirse en la pérdida de datos en caso de un corte de corriente o fallo del sistema. También puede no ser conveniente para las particiones root (/) y /home, porque las comprobaciones del sistema de archivos pueden tomar mucho tiempo. Un sistema de archivos ext2 puede ser convertido a ext3.

## Ext3

Ext3 (third extended filesystem o tercer sistema de archivos extendido) se diferencia de ext2 en que trabaja con registro por diario (*journaling*) y porque utiliza un árbol binario balanceado (árbol AVL, creado por los matemáticos rusos Georgii Adelson-Velskii y Yevgeniy Landis) y también por incorporar el método Orlov de asignación para bloques de disco. Además ext3 permite ser montado y utilizado como si fuera ext2 y actualizar desde ext2 hacia ext3 sin necesidad de formatear la partición y sin perder los datos almacenados en ésta.

## Ext4

Ext4 (fourth extended filesystem o cuarto sistema de archivos extendido) es un sistema de archivos con registro por diario, publicado por Andrew Morton como una mejora compatible con el formato Ext3. Las mejoras respecto de Ext3 incluyen, entre otras cosas, el soporte de volúmenes de hasta 1024 PiB, soporte añadido de extends (conjunto de bloques físicos contiguos), menor uso de recursos de sistema, mejoras sustanciales en la velocidad de lectura y escritura y verificación más rápida con fsck. Es el sistema de archivos predeterminado en CentOS 6 y Red Hat™ Enterprise Linux 6.

Acerca del registro por diario (journaling).

El registro por diario (journaling) es un mecanismo por el cual un sistema de archivos implementa transacciones. Consiste en un registro en el que se almacena la información necesaria para restablecer los datos dañados por una transacción en caso de que ésta falle, como puede ocurrir durante una interrupción de energía.

## FAT

FAT es con diferencia el sistema de archivos más simple de aquellos compatibles con Windows NT. El sistema de archivos FAT se caracteriza por la tabla de asignación de archivos (FAT), que es realmente una tabla que reside en la parte más "superior" del volumen. Para proteger el volumen, se guardan dos copias de la FAT por si una resultara dañada. Además, las tablas FAT y el directorio raíz deben almacenarse en una ubicación fija para que los archivos de arranque del sistema se puedan ubicar correctamente.

### Ventajas de FAT

No es posible realizar una recuperación de archivos eliminados en Windows NT en ninguno de los sistemas de archivos compatibles. Las utilidades de recuperación de archivos eliminados intentan tener acceso directamente al hardware, lo que no se puede hacer en Windows NT. Sin embargo, si el archivo estuviera en una partición FAT y se reiniciara el sistema en MS-DOS, se podría recuperar el archivo. El sistema de archivos FAT es el más adecuado para las unidades y/o particiones de menos de 200 MB aproximadamente, ya que FAT se inicia con muy poca sobrecarga.



## Desventajas de FAT

Cuando se utilicen unidades o particiones de más de 200 MB, es preferible no utilizar el sistema de archivos FAT. El motivo es que a medida que aumente el tamaño del volumen, el rendimiento con FAT disminuirá rápidamente. No es posible establecer permisos en archivos que estén en particiones FAT.

## HFS

Sistema de Archivos Jerárquico o Hierarchical File System (HFS), es un sistema de archivos desarrollado por Apple Inc. para su uso en computadores que corren Mac OS. Originalmente diseñado para ser usado en disquetes y discos duros, también es posible encontrarlo en dispositivos de solo-lectura como los CD-ROMs.

## HFS+

HFS Plus o HFS+ es un sistema de archivos desarrollado por Apple Inc. para reemplazar al HFS (Sistema jerárquico de archivos). También es el formato usado por el iPod al ser formateado desde un Mac. HFS Plus también es conocido como HFS Extended y Mac OS Extended. Durante el desarrollo, Apple se refirió a él con el nombre clave Sequoia.

HFS Plus es una versión mejorada de HFS, soportando archivos mucho más grandes (Bloques direccionables de 32 bits en vez de 16) y usando Unicode (En vez de Mac OS Roman) para el nombre de los archivos, lo que además permitió nombres de archivo de hasta 255 letras.

## NTFS

Desde el punto de vista de un usuario, NTFS sigue organizando los archivos en directorios que, al igual que ocurre en HPFS, se ordenan.

Sin embargo, a diferencia de FAT o de HPFS, no hay ningún objeto "especial" en el disco y no hay ninguna dependencia del hardware subyacente, como los sectores de 512 bytes. Además, no hay ninguna ubicación especial en el disco, como las tablas de FAT o los superbloques de HPFS.

Los objetivos de NTFS son proporcionar lo siguiente:

- Confiabilidad, que es especialmente deseable para los sistemas avanzados y los servidores de archivos
- Una plataforma para tener mayor funcionalidad
- Compatibilidad con los requisitos de POSIX
- Eliminación de las limitaciones de los sistemas de archivos FAT y HPFS

#### Ventajas de NTFS

NTFS es la mejor opción para volúmenes de unos 400 MB o más. El motivo es que el rendimiento no se degrada en NTFS, como ocurre en FAT, con tamaños de volumen mayores.

La posibilidad de recuperación está diseñada en NTFS de manera que un usuario nunca tenga que ejecutar ningún tipo de utilidad de reparación de disco en una partición NTFS.

#### Desventajas de NTFS

No se recomienda utilizar NTFS en un volumen de menos de unos 400 MB, debido a la sobrecarga de espacio que implica. Esta sobrecarga de espacio se refiere a los archivos de sistema de NTFS que normalmente utilizan al menos 4 MB de espacio de unidad en una partición de 100 MB.



NTFS no integra actualmente ningún cifrado de archivos. Por tanto, alguien puede arrancar en MS-DOS u otro sistema operativo y emplear una utilidad de edición de disco de bajo nivel para ver los datos almacenados en un volumen NTFS.

No es posible formatear un disco con el sistema de archivos NTFS; Windows NT formatea todos los discos con el sistema de archivos FAT porque la sobrecarga de espacio que implica NTFS no cabe en un disco. (MICROSOFT, s.f.)

## **UNIDAD VI:**

### **ENTRADA/SALIDA EN LOS DIFERENTES SISTEMAS OPERATIVOS**

El papel del sistema operativo en el sistema E/S de una computadora es administrar y controlar las operaciones y dispositivos de E/S.

#### [Vista general](#)

Los elementos básicos del hardware E/S - puertos, buses y controladores de dispositivos - acomodan una amplia variedad de dispositivos de E/S. Para encapsular los detalles y peculiaridades de los diferentes dispositivos, se estructura el kernel del sistema operativo para usar módulos de manejadores de dispositivo. Los manejadores de dispositivos presentan una interfaz uniforme de acceso a dispositivos con el subsistema de E/S, de manera muy similar a como las llamadas al sistema proporcionan una interfaz estándar entre la aplicación y el sistema operativo.

#### [Hardware de E/S](#)

Las computadoras operan muchas clases de dispositivos. En los tipos generales incluyen los dispositivos de almacenamiento (discos, cintas), dispositivos de transmisión (tarjetas de red, módem) y dispositivos para la interfaz con el ser humano (pantalla, teclado, ratón). Un dispositivo se comunica con un sistema de cómputo enviando señales a través de un cable o incluso a través del aire. El dispositivo se comunica con la máquina mediante un punto de conexión denominado puerto (por ejemplo, un puerto serial).



Si uno o más dispositivos utilizan un conjunto común de cables, la conexión se denomina bus. Un bus es un conjunto de cables y un protocolo definido rígidamente que especifica un conjunto de mensajes que pueden enviarse por los cables. En términos de la electrónica, los mensajes se transmiten mediante patrones de voltajes eléctricos que se aplican a los cables con tiempos (timings) definidos. Cuando el dispositivo A tiene un cable que se conecta al dispositivo B, y el dispositivo B tiene un cable que se conecta al dispositivo C y el dispositivo C tiene un cable que se conecta a un puerto en la computadora, este arreglo se denomina cadena de margarita. Generalmente opera como unbus.

Un controlador es un conjunto de componentes electrónicos que pueden operar un puerto, un bus o un dispositivo. Un controlador de puerto serial es un ejemplo de un controlador de dispositivos sencillo. Es una sola pastilla (chip) en la computadora que controla las señales en los cables de un puerto serial.

¿Cómo puede el procesador entregar comandos y datos a un controlador para realizar una transferencia de E/S? La respuesta sencilla es que el controlador tiene uno o más registros para datos y señales de control. El procesador se comunica con el controlador leyendo y escribiendo patrones de bits en estos registros. Una forma en que puede darse esta comunicación es mediante el uso de instrucciones especiales de E/S que especifican la transferencia de un byte o palabra a la dirección de un puerto de E/S. La instrucción de E/S activa líneas del bus para seleccionar el dispositivo apropiado y mover bits dentro o fuera de un registro del dispositivo. De manera alterna, el controlador de dispositivo puede soportar E/S con mapeo en memoria. En este caso, los registros de control del dispositivo se mapean en el espacio de direcciones del procesador. La CPU ejecuta solicitudes de E/S utilizando las instrucciones estándar de transferencia de datos para leer y escribir los registros de control del dispositivo.

El controlador de graficación tiene puertos de E/S para las operaciones de control básicas, pero el controlador tiene una gran región con mapeo en memoria para alojar los contenidos de la pantalla. El controlador envía salida a la pantalla escribiendo datos en dicha región.

El controlador genera la imagen en la pantalla con base en el contenido de esta memoria. La facilidad de escritura en un controlador de E/S con mapeo en memoria tiene como contrapeso una desventaja: debido a que un tipo común de fallo de software en la operación de escritura a través de un apuntador incorrecto a una región no deseada de la memoria, un registro de dispositivo con mapeo en memoria es vulnerable a una modificación accidental. Por supuesto, la memoria protegida reduce este riesgo.

Un puerto de E/S típicamente consta de cuatro registros, denominados registros de status, control, data-in y data-out. El registro status contiene bits que el anfitrión puede leer. Estos bits indican diversos estados, tales como: si el comando actual ya se completó, si está disponible un byte para leer del registro data-in, o si ha habido un error de dispositivo. El anfitrión puede escribir el registro control para iniciar un comando o cambiar el modo de un dispositivo. El anfitrión lee el registro data-in para obtener entradas y escribe el registro data-out para enviar salidas. Los registros de datos tienen comúnmente un tamaño de uno a cuatro bytes. Algunos controladores cuentan con pastillas (chips) FIFO que pueden contener varios bytes de datos de E/S para ampliar la capacidad del controlador más allá del tamaño del registro de datos. Una pastilla FIFO puede contener una pequeña ráfaga de datos hasta que el dispositivo o anfitrión sea capaz de recibir dichos datos.

#### Interrupciones

El mecanismo básico de interrupción funciona de la siguiente manera. El hardware de la CPU tiene un cable llamado línea de solicitud de interrupciones que la CPU revisa luego de ejecutar cada instrucción. Cuando la CPU detecta que un controlador ha colocado una señal en la línea de solicitud de interrupciones, la CPU guarda una pequeña cantidad del estado (como el valor actual del apuntador de instrucciones) y salta a la rutina del manejador de interrupciones en una dirección fija en memoria.



El manejador de interrupciones determina la causa de la interrupción, realiza el procesamiento necesario y ejecuta la instrucción return from interrupt para regresar la CPU al estado de ejecución antes de la interrupción. El controlador de dispositivo genera una interrupción y despacha al manejador de interrupciones, y éste apaga (clear) la interrupción dando servicio al dispositivo. El mecanismo básico de interrupción habilita a la CPU para responder a un evento asíncrono, el cual puede ser que el controlador de dispositivo quede listo para dar servicio.

La mayoría de la CPU tiene dos líneas de solicitud de interrupción. Una es la interrupción no mascarable, que se reserva para eventos como errores de memoria no recuperables. La segunda línea de interrupción es mascarable: puede ser apagada por la CPU antes de la ejecución de secuencias de instrucciones críticas que no deben ser interrumpidas. La interrupción mascarable es utilizada por los controladores de dispositivos para solicitar servicio.

El mecanismo de interrupción acepta una dirección - un número que selecciona una rutina específica para manejo de interrupciones de entre un pequeño conjunto. En la mayoría de las arquitecturas, esta dirección es un desplazamiento en una tabla denominada vector de interrupciones. Este vector contiene las direcciones de memoria de los manejadores de interrupción especializados. El propósito de un mecanismo de interrupción con base en el vector es reducir la necesidad de que un manejador único busque todas las posibles fuentes de interrupción para determinar cuál necesita servicio. En la práctica, sin embargo, las computadoras tienen más dispositivos que elementos de direcciones en el vector de interrupciones.

El mecanismo de interrupción también implementa un sistema de niveles de prioridad de interrupción. Este mecanismo habilita a la CPU para diferir el manejo de interrupciones de baja prioridad sin enmascarar todas las interrupciones y hace posible que una interrupción de alta prioridad tenga precedencia sobre la ejecución de una interrupción de baja prioridad.

Un sistema operativo tiene otros buenos usos para un mecanismo eficiente de hardware que guarda una pequeña cantidad del estado del procesador, y luego llama una rutina privilegiada en el kernel. Otro uso se encuentra en la implementación de llamadas al sistema. También se pueden emplear interrupciones para administrar el flujo de control dentro del kernel.

Las interrupciones se emplean en todos los sistemas operativos modernos para manejar los eventos asíncronos y para generar trampas que se comunican a rutinas en modo supervisor en el kernel. Para lograr que primero haga el trabajo más urgente, las computadoras modernas utilizan un sistema de prioridades de interrupción. Los controladores de dispositivos, los fallos de hardware y las llamadas al sistema, todos ellos generan interrupciones para activar rutinas del kernel. Debido a que las interrupciones se emplean intensamente para el procesamiento sensible al tiempo, se requiere un eficiente manejo de interrupciones para un buen desempeño del sistema.

#### Acceso Directo a Memoria

Para iniciar una transferencia del controlador de acceso directo a memoria (Direct Memory Access, DMA), el anfitrión escribe un bloque de comandos DMA en la memoria. Este bloque contiene un apuntador a la fuente de una transferencia, un apuntador al destino de la transferencia y una cuenta del número de bytes a transferir. El CPU escribe la dirección de este bloque de comandos en el controlador de DMA, luego sigue con otro trabajo. El controlador de DMA procede entonces a operar directamente el bus de memoria, colocando direcciones en dicho bus para realizar transferencias sin la ayuda de la CPU principal. Un controlador de DMA sencillo es un componente estándar en las PC, y unas tarjetas de E/S de control de bus para la PC generalmente contienen su propio hardware DMA de alta velocidad.



La secuencia de reconocimiento entre el controlador de DMA y el controlador de dispositivo se efectúa mediante un par de cables denominados DMA-request (petición) y DMA-acknowledge (reconocimiento). El controlador de dispositivo coloca una señal en el cable DMA-request cuando está disponible para su transferencia una palabra de datos. Esta señal hace que el controlador de DMA se apropie del bus de memoria, que coloque la dirección deseada en los cables de dirección de memoria, y que coloque una señal en el cable DMA-acknowledge.

Cuando el controlador de dispositivo recibe esta señal de reconocimiento, transfiere a memoria la palabra de datos y remueve la señal DMA-request. Cuando termina toda la transferencia, el controlador de DMA interrumpe a la CPU. Algunas arquitecturas de computadoras utilizan direcciones de memoria física para DMA, pero otras efectúan un acceso directo a memoria virtual (direct virtual memory access, DVMA), utilizando direcciones virtuales que pasan por un proceso de traducción de dirección de memoria virtual a dirección de memoria física. El DVMA puede realizar una transferencia entre dos dispositivos con mapeo en memoria sin la intervención de la CPU o el uso de la memoria principal.

En los kernels de modo protegido, el sistema operativo generalmente impide que los procesos emitan directamente comandos a dispositivos. Esta disciplina protege a los datos de violaciones al control de acceso y también protege al sistema de un empleo erróneo de controladores de dispositivo que podrían provocar una caída del sistema. En los kernels sin protección de memoria, los procesos pueden acceder directamente a los controladores de dispositivos. Se puede emplear este acceso directo para obtener un alto desempeño, ya que con él se puede evitar comunicación con el kernel, conmutación de contexto y capas de software de kernel. Desafortunadamente, el acceso directo interfiere con la seguridad y estabilidad del sistema.

Principales conceptos de los aspectos de hardware del sistema de E/S son:

- Un bus.
- Un controlador.

- Un puerto de E/S y sus registros.
- La relación de secuencia de reconocimiento entre el anfitrión y un controlador de dispositivo.
- La ejecución de esta secuencia de reconocimiento en un ciclo de escrutinio o mediante interrupciones.
- La descarga de este trabajo a un controlador de DMA para transferencias grandes. (Silberchatz, 2002, págs. 401-412)

## Fundamentos del software de E/S

### Objetivos del software de E/S

Un concepto clave en el diseño del software de E/S se conoce como independencia de dispositivo. Lo que significa es que debe ser posible escribir programas que puedan acceder a cualquier dispositivo de E/S sin tener que especificar el dispositivo por adelantado.

Un objetivo muy relacionado con la independencia de los dispositivos es la denominación uniforme. El nombre de un archivo o dispositivo simplemente debe ser una cadena o un entero sin depender del dispositivo de ninguna forma.

Otra cuestión importante relacionada con el software de E/S es el manejo de errores. En general los errores se deben manejar lo más cerca del hardware que sea posible. Si el controlador descubre un error de lectura, debe tratar de corregir el error por sí mismo. Si no puede entonces el software controlador de dispositivo debe manejarlo, tal vez con sólo tratar de leer el bloque de nuevo.

Otra cuestión clave es la de las transferencias síncronas (de bloqueo) contra las asíncronas (controladas por interrupciones). La mayoría de las operaciones de E/S son asíncronas: la CPU inicia la transferencia y se va a hacer algo más hasta que llega a la interrupción.



Otra cuestión relacionada con el software de E/S es el uso de búfer. A menudo los datos que provienen de un dispositivo no se pueden almacenar directamente en su destino final.

#### E/S programada

Hay tres maneras fundamentales distintas en que se puede llevar a cabo la E/S.

1. E/S programada.
2. E/S controlada por interrupciones.
3. E/S mediante el uso de DMA.

La forma más simple de E/S es cuando la CPU hace todo el trabajo. A este método se le conoce como E/S programada.

La E/S programada es simple, pero tiene la desventaja de ocupar la CPU tiempo completo hasta que se completen todas las operaciones de E/S. Si el tiempo para “imprimir” un carácter es muy corto (debido a que todo lo que hace la impresora es copiar el nuevo carácter en un búfer interno), entonces está bien usar ocupado en espera. Además, en un sistema incrustado o embebido, donde la CPU no tiene más que hacer, ocupado en espera es razonable. Sin embargo, en sistemas más complejos en donde la CPU tiene otros trabajos que realizar, ocupado en espera es ineficiente.

#### E/S controlada por interrupciones

La forma de permitir que la CPU haga algo más mientras espera a que la impresora esté lista es utilizar interrupciones.

## E/S mediante el uso de DMA

Una obvia desventaja de la E/S controlada por interrupciones es que ocurre una interrupción en cada carácter. Las interrupciones requieren tiempo, por lo que este esquema desperdicia cierta cantidad de tiempo de la CPU. Una solución es utilizar DMA. Aquí la idea es permitir que el controlador de DMA alimente los caracteres a la impresora uno a la vez, sin que la CPU se moleste. Es esencial, el DMA es E/S programada, sólo que el controlador de DMA realiza todo el trabajo en vez de la CPU principal. Esta estrategia requiere hardware especial (el controlador de DMA) pero libera la CPU durante la E/S para realizar otro trabajo.

La gran ganancia con DMA es reducir el número de interrupciones de una por cada carácter a una por cada búfer impreso. Si hay muchos caracteres y las interrupciones son lentas, esto puede ser una gran mejora. Por otra parte, el controlador de DMA es comúnmente más lento que la CPU principal.

## Capas del software de E/S

Por lo general el software de E/S se organiza en cuatro capas.

NIVEL	FUNCIONES DE E/S
Proceso de usuario.	Hacer la llamada de E/S; aplicar formato a la E/S; poner en cola.
Software independiente del dispositivo.	Nombramiento, protección, bloqueo, uso de búfer, asignación.
Controladores de dispositivo.	Establecer los registros de dispositivo; verificar el estado.
Manejadores de interrupciones.	Despertar el controlador cuando se completa la E/S.
Hardware.	Realizar operaciones de E/S.



## Manejadores de interrupciones

Una vez se haya completado la interrupción de hardware, se realizarán una serie de pasos que se deben llevar a cabo en el software.

1. Guardar los registros que no han sido guardados por el hardware de la interrupción.
2. Establecer un contexto para el procedimiento de servicio de interrupción.
3. Establecer una pila para el procedimiento de servicio de interrupciones.
4. Reconocer el controlador de interrupciones.
5. Copiar los registros desde donde se guardaron a la tabla de procesos.
6. Ejecutar el procedimiento de servicio de interrupciones.
7. Elegir cuál proceso ejecutar a continuación.
8. Establecer el contexto de la MMU para el proceso que se va a ejecutar a continuación.
9. Cargar los registros del nuevo proceso.
10. Empezar a ejecutar el nuevo proceso.

## Drivers de dispositivos

Cada dispositivo de E/S conectado a una computadora necesita cierto código específico para controlarlo. Este código, conocido como driver, es escrito por el fabricante del dispositivo y se incluye junto con el mismo.

Cada driver maneja un tipo de dispositivo o, a lo más, una clase de dispositivos estrechamente relacionados.

Generalmente los sistemas operativos clasifican los controladores en una de un pequeño número de categorías. Las categorías más comunes son los dispositivos de bloque como los discos, que contienen varios bloques de datos que se pueden direccionar de manera independiente, y los dispositivos de carácter como los teclados y las impresoras, que generan o aceptan un flujo de caracteres.

Un controlador de dispositivo tiene varias funciones. La más obvia es aceptar peticiones abstractas de lectura y escritura del software independiente del dispositivo que está por encima de él, y ver que se lleven a cabo. Pero también hay otras tantas funciones que deben realizar. Por ejemplo, el controlador debe inicializar el dispositivo, si es necesario. También puede tener que administrar sus propios requerimientos y eventos del registro.

#### Software de E/S independiente del dispositivo

Las funciones que se realizan comúnmente en el software independiente del dispositivo son:

- Interfaz uniforme para controladores de dispositivos.
- Uso de búfer.
- Reporte de errores.
- Asignar y liberar dispositivos dedicados.
- Proporcionar un tamaño de bloque independiente del dispositivo.

La función básica del software independiente del dispositivo es realizar las funciones de E/S que son comunes para todos los dispositivos y proveer una interfaz uniforme para el software a nivel de usuario.

#### Software de E/S en espacio de usuario

Aunque la mayor parte del software de E/S está dentro del sistema operativo, una pequeña porción de éste consiste en bibliotecas vinculadas entre sí con programas de usuario, e incluso programas enteros que se ejecutan desde el exterior del kernel. Las llamadas al sistema, incluyendo las llamadas al sistema de E/S, se realizan comúnmente mediante procedimientos de biblioteca. El software de E/S de bajo nivel consiste en procedimientos de biblioteca.



Otra categoría importante es el sistema de colas. El uso de colas (spooling) es una manera de lidiar con los dispositivos de E/S dedicados a un sistema de multiprogramación

## Discos

### Hardware de disco

Los discos son de varios tipos. Los más comunes son los discos magnéticos (discos duros y discos flexibles). Se caracterizan por el hecho de que las operaciones de lectura y escritura son igual de rápidas, lo que los hace ideales como memoria secundaria. Algunas veces se utilizan arreglos de estos discos para ofrecer un almacenamiento altamente confiable. Para la distribución de programas, datos y películas son también importantes varios tipos de discos ópticos (CD-ROM's, CD-grabable y DVD).

#### Discos magnéticos

Los discos magnéticos se organizan en cilindros, cada uno de los cuales contiene tantas pistas como cabezas apiladas en forma vertical. Las pistas se dividen en sectores. El número de sectores alrededor de la circunferencia es por lo general de 8 a 32 en los discos flexibles, y hasta vario cientos en los discos duros. El número de cabezas varía entre 1 y 16.

En otros discos, en especial los discos IDE (electrónica de unidad integrada) y SATA (ATA serial), la unidad de disco contiene un microcontrolador que realiza un trabajo considerable y permite al controlador real emitir un conjunto de comandos de nivel superior. A menudo el controlador coloca las pistas en caché, reasigna los bloques defectuosos y mucho más.

Los discos modernos se dividen en zonas, con más sectores en las zonas exteriores que en las interiores.

## RAID

La idea básica de un RAID es instalar una caja llena de discos a un lado de la computadora (que por lo general es un servidor grande), reemplazar la tarjeta controladora de discos con un controlador RAID, copiar los datos al RAID y después continuar la operación normal.

Además de aparecer como un solo disco para el software, todos los RAID's tiene la propiedad de que los datos se distribuyen entre las unidades, para permitir la operación en paralelo. Patterson y sus colaboradores definieron varios esquemas distintos para hacer esto, y ahora se conocen como RAID nivel 0 hasta RAIDnivel

## CD-ROM's

En años recientes se han empezado a utilizar los discos ópticos (en contraste a los magnéticos). Estos discos tienen densidades de grabación mucho más altas que los discos magnéticos convencionales. Los discos ópticos se desarrollaron en un principio para grabar programas de televisión, pero se les puede dar un uso más estético como dispositivos de almacenamiento de computadora.

Un CD se prepara en varios pasos. El primero consiste en utilizar un láser infrarrojo de alto poder para quemar hoyos de 0.8 micrones de diámetro en un disco maestro con cubierta de vidrio. A partir de este disco maestro se fabrica un molde, con protuberancias en lugar de los hoyos del láser. En este molde se inyecta resina de policarbonato fundido para formar un CD con el mismo patrón de hoyos que el disco maestro de vidrio. Después se deposita una capa muy delgada de aluminio reflectivo en el policarbonato, cubierta por una laca protectora y



finalmente una etiqueta. Las depresiones en el sustrato de policarbonato se llaman hoyos (pits); las áreas no quemadas entre los hoyos se llaman áreas lisas (lands).

#### CD-regrabables

Las unidades de CD-RW utilizan láseres con tres potencias: en la posición de alta energía el láser funde la aleación y la convierte del estado cristalino de alta reflectividad al estado amorfo de baja reflectividad para representar un hoyo; en la posición de energía media la aleación se funde y se vuelve a formar en su estado cristalino natural para convertirse en un área lisa nuevamente; en baja energía se detecta el estado del material (para lectura), pero no ocurre una transición de estado.

#### DVD

Ahora se conoce oficial como disco versátil digital (digital versatile disk). Los DVD's utilizan el mismo diseño general que los CD's, con discos de policarbonato moldeado por inyección de 120 mm que contienen hoyos y áreas lisas, que se iluminan mediante un diodo láser y se leen mediante un fotodetector. Lo nuevo es el uso de:

1. Hoyos más pequeños (0.4 micrones).
2. Una espiral más estrecha.
3. Un láser rojo.

En conjunto, estas mejoras elevan la capacidad siete veces, hasta 4.7 GB. Una unidad de DVD 1X opera a 1.4 MB/seg (en comparación con los de 150 KB/seg de los CDs).

## Formato de disco

Un disco duro consiste en una pila de platos de aluminio, aleación de acero o vidrio, de 5.25 o 3.5 pulgadas de diámetro. En cada plato se deposita un óxido de metal delgado magnetizable.

Antes de poder utilizar el disco, cada plato debe recibir un formato de bajo nivel mediante software. El formato consiste en una serie de pistas concéntricas, cada una de las cuales contiene cierto número de sectores con huecos cortos entre los sectores.

El preámbulo empieza con cierto patrón de bits que permite al hardware reconocer el inicio del sector. También contiene los números de cilindro y sector, junto con cierta información adicional. El tamaño de la porción de datos se determina con base en el programa de formato de bajo nivel. La mayoría de los discos utilizan sectores de 512 bytes.

Una vez que se completa el formato de bajo nivel, el disco se particiona. En sentido lógico, cada partición es como un disco separado. Las particiones son necesarias para permitir que coexistan varios sistemas operativos. En la mayoría de las computadoras, el sector 0 contiene el registro de inicio maestro (MBR), el cual contiene cierto código de inicio además de la tabla de particiones al final. La tabla de particiones proporciona el sector inicial y el tamaño de cada partición.

## Relojes

Los relojes (también conocidos como temporizadores) son esenciales para la operación de cualquier sistema de multiprogramación, por una variedad de razones. Mantienen la hora del día y evitan que un proceso monopolice la CPU. El software de reloj puede tomar la forma de un software controlado de dispositivo, aun y cuando un reloj no es un dispositivo de bloque (como un disco) ni un dispositivo de carácter (como un ratón).



## Hardware de reloj

Hay dos tipos de relojes de uso común en las computadoras, y ambos son bastante distintos de los relojes que utilizan las personas. Los relojes más simples están enlazados a la línea de energía de 110 o 220 voltios y producen una interrupción en cada ciclo de voltaje, a 50 o a 60 Hz. Estos relojes solían dominar el mercado, pero ahora son raros.

El otro tipo de reloj se construye a partir de 3 componentes: un oscilador de cristal, un contador y un registro contenedor. Cuando una pieza de cristal de cuarzo se corta en forma apropiada y se monta bajo tensión, puede generar una señal periódica con una precisión muy grande, por lo general en el rango de varios cientos de megahertz, dependiendo del cristal elegido. Mediante el uso de componentes electrónicos, esta señal base puede multiplicarse por un pequeño entero para obtener frecuencias de hasta mil MHz o incluso más. Por lo menos uno de esos circuitos se encuentra comúnmente en cualquier computadora, el cual proporciona una señal de sincronización para los diversos circuitos de la misma. Esta señal se alimenta al contador para hacer que cuente en forma descendente hasta cero. Cuando el contador llega a cero, produce una interrupción de la CPU.

Si se utiliza un cristal de 500 MHz, entonces se aplica un pulso al contador cada dos nseg. Con registros de 32 bits (sin signo), se pueden programar interrupciones para que ocurran a intervalos de 2 nseg hasta 8.6 seg. Los chips de reloj programables por lo general contienen dos o tres relojes que pueden programarse de manera independiente.

Para evitar que se pierda la hora actual cuando se apaga la computadora, la mayoría cuentan con un reloj de respaldo energizado por batería, implementando con el tipo de circuitos de baja energía que se utilizan en los relojes digitales. El reloj de batería puede leerse al iniciar el sistema. Si no está presente, el software puede pedir al usuario la fecha y hora actuales.

## Software de reloj

Todo lo que hace el hardware de reloj es generar interrupciones a intervalos conocidos. Todo lo demás que se relacione con el tiempo debe ser realizado por el software controlador del reloj. Las tareas exactas del controlador de reloj varían de un sistema operativo a otro, pero por lo general incluyen la mayoría de las siguientes tareas:

1. Mantener la hora del día.
2. Evitar que los procesos se ejecuten por más tiempo del que tienen permitido.
3. Contabilizar el uso de la CPU.
4. Manejar la llamada al sistema alarm que realizan los procesos de usuario.
5. Proveer temporizadores guardianes (watchdogs) para ciertas partes del mismo sistema.
6. Realizar perfilamiento, supervisión y recopilación de estadísticas.

## Temporizadores de software

La mayoría de las computadoras tienen un segundo reloj programable que se puede establecer para producir interrupciones del temporizador, a cualquier velocidad que quiera un programa. Mientras que la frecuencia de interrupción sea baja, no habrá problema al usar este segundo temporizador para fines específicos de la aplicación. El problema surge cuando la frecuencia del temporizador específico de la aplicación es muy alta.

Los temporizadores de software evitan las interrupciones. En vez de ello, cada vez que el kernel se ejecuta por alguna otra razón, justo antes de regresar al modo de usuario comprueba el reloj de tiempo real para ver si ha expirado un temporizador de software.



Si el temporizador ha expirado, se realiza el evento programado (por ejemplo, transmitir paquetes o comprobar si llegó un paquete), sin necesidad de cambiar al modo kernel debido a que el sistema ya se encuentra ahí. Una vez realizado el trabajo, el temporizador de software se restablece para empezar de nuevo. Todo lo que hay que hacer es copiar el valor actual del reloj en el temporizador y sumarle el intervalo de tiempo de inactividad.

## Interfaces de usuario: teclado, ratón, monitor

### Software de entrada

La entrada de usuario proviene principalmente del teclado y del ratón. En una computadora personal, el teclado contiene un microprocesador integrado que por lo general se comunica, a través de un puerto serial especializado, con un chip controlador en la tarjeta principal (aunque cada vez con más frecuencia, los teclados se conectan a un puerto USB). Se genera una interrupción cada vez que se oprime una tecla, y se genera una segunda interrupción cada vez que se suelta. En cada una de estas interrupciones de teclado, el software controlador del mismo extrae la información acerca de lo que ocurre desde el puerto de E/S asociado con el teclado. Todo lo demás ocurre en el software y es muy independiente del hardware.

### Software de teclado

El número en el puerto de E/S es el número de tecla, conocido como código de exploración, no el código ASCII. Los teclados tienen menos de 128 teclas, por lo que solo se necesitan 7 bits para representar el número de tecla. El octavo bit se establece en 0 cuando se oprime una tecla, y el 1 cuando se suelta.

Se pueden adoptar dos filosofías posibles para el controlador. En la primera, el trabajo del controlador es sólo aceptar la entrada y pasarl hacia arriba sin modificarla. Un programa que lee del teclado obtiene una secuencia pura de códigos ASCII.

La segunda filosofía: el controlador maneja toda la edición entre líneas, y envía sólo las líneas corregidas a los programas de usuario. La primera filosofía está orientada a caracteres; la segunda está orientada a líneas. En un principio se conocieron como modo crudo y modo cocido, respectivamente.

#### Software de ratón

La mayoría de las PC's tienen un ratón, o algunas veces un trackball, que sencillamente es un ratón boca arriba. Un tipo común de ratón tiene una bola de goma en su interior que se asoma por un hoyo en la parte inferior y gira, a medida que el ratón se desplaza por una superficie dura, frotándose contra unos rodillos posicionados en ejes ortogonales. Otro tipo popular de ratón es el óptico, que está equipado con uno o más diodos emisores de luz y fotodetectores en su parte inferior. Los ratones ópticos modernos tienen un chip de procesamiento de imágenes en ellos y sacan fotos continuas de baja resolución de la superficie debajo de ellos, buscando cambios de imagen en imagen.

Los ratones inalámbricos son iguales a los alámbricos, excepto que en vez de devolver sus datos a la computadora a través de un cable, utilizan radios de baja energía, por ejemplo mediante el uso del estándar bluetooth.

#### Software de salida

Los editores de pantalla y muchos otros programas sofisticados necesitan la capacidad de actualizar la pantalla en formas complejas, como sustituir una línea a mitad de la pantalla.



Para satisfacer esta necesidad la mayoría de los controladores de software de salida proporcionan una serie de comandos para desplazar el cursor, insertar y eliminar caracteres o líneas en el cursor, entre otras tareas. A menudo estos comandos se conocen como secuencias de escape. En cierto momento, la industria vio la necesidad de estandarizar la secuencia de escape por lo que se desarrolló un estándar ANSI.

## El sistema X Window

El sistema X window es muy portátil y se ejecuta por completo en espacio de usuario. En un principio tenía como propósito principal conectar un gran número de terminales de usuario remotas con un servidor de cómputo central, por lo que está dividido lógicamente en software cliente y software servidor, que puede ejecutarse potencialmente en distintas computadoras.

Cuando se inicia un programa de X, abre una conexión a uno o más servidores X, que se van a llamar estaciones de trabajo, aun cuando se podrían colocar en el mismo equipo que el programa X en sí. X considera que esta conexión es confiable en cuanto a que los mensajes perdidos y duplicados se manejan mediante el software de red y no tienen que preocuparse por errores de comunicación. Por lo general se utiliza TCP/IP entre el cliente y el servidor. Pasan cuatro tipos de mensajes a través de la conexión:

1. Comandos de dibujo del programa a la estación de trabajo.
2. Respuestas de la estación de trabajo a las solicitudes del programa.
3. Mensajes del teclado, del ratón y de otros eventos.
4. Mensajes de error.

Un concepto clave en X es el recurso. Un recurso es una estructura de datos que contiene cierta información. Los programas de aplicación crean recursos en las estaciones de trabajo.

Los recursos se pueden compartir entre varios procesos en la estación de trabajo. Los recursos tienen un tiempo de vida corto y no sobreviven a los reinicios de la estación de trabajo. Algunos recursos típicos son las ventanas, los tipos de letra, los mapas de colores (paletas de colores), mapas de píxeles (mapas de bits), los cursores y los contextos gráficos. Estos últimos se utilizan para asociar las propiedades con las ventanas y son similares en concepto a los contextos de dispositivos en Windows.

### Interfaces gráficas de usuario

Una GUI tiene cuatro elementos esenciales, denotados por los caracteres WIMP. Las letras representan ventanas (windows), iconos (Icons), menús (Menus) y dispositivo señalador (pointing device), respectivamente. Las ventanas son áreas rectangulares en la pantalla que se utilizan para ejecutar programas. Los iconos son pequeños símbolos en los que se puede hacer clic para que ocurra una acción. Los menús son listas de acciones, de las que se puede elegir una. Por último, un dispositivo señalador es un ratón, trackball u otro dispositivo de hardware utilizado para desplazar un cursor alrededor de la pantalla para seleccionar elementos.

El elemento básico de la pantalla es un área rectangular llamada ventana. La posición y el tamaño de una ventana se determinan en forma única al proporcionar las coordenadas (en píxeles) de dos esquinas diagonalmente opuestas. Una ventana puede contener una barra de título, una barra de menús, una barra de desplazamiento vertical y una barra de desplazamiento horizontal.

### Mapas de bits

Los procedimientos de la GDI son ejemplos de gráficos vectoriales. Se utilizan para colocar figuras geométricas y texto en la pantalla.



Se pueden escalar con facilidad a pantallas más grandes o pequeñas (siempre y cuando el número de píxeles en la pantalla sea el mismo). También son relativamente independientes del dispositivo. Una colección de llamadas a procedimientos de la GDI se puede ensamblar en un archivo que describa un dibujo completo. A dicho archivo se le conoce como metarchivo de Windows y es ampliamente utilizado para transmitir dibujos de un programa de Windows a otro.

No todas las imágenes se pueden manipular las computadoras se pueden generar mediante gráficos vectoriales. Por ejemplo, las fotografías y los videos no utilizan gráficos vectoriales en vez de ello, estos elementos se exploran al sobreponer una rejilla en la imagen. Los valores rojo, verde y azul promedio de cada cuadro de la rejilla se muestran y se guardan como el valor de un píxel. A dicho archivo se le conoce como mapa de bits. Hay muchas herramientas en Windows para manipular mapas de bits.

Otro uso para los mapas de bits es el texto. Una forma de representar un carácter específico en cierto tipo de letra es mediante un pequeño mapa de bits. Al agregar texto a la pantalla se convierte entonces en cuestión de mover mapas de bits. Un problema con los mapas de bits es que no se escalan. (Tanenbaum A. , 2009, págs. 343-412)

## UNIDAD VII:

### CONCEPTOS DE SEGURIDAD Y PROTECCIÓN EN LOS SISTEMAS OPERATIVOS

Un sistema operativo puede dar soporte de ejecución a múltiples procesos de múltiples usuarios, que ejecutan de manera concurrente. Por ello, una de las funciones principales del sistema operativo es proteger los recursos de cada usuario para que pueda ejecutar en un entorno seguro. Donde los mecanismos permiten controlar el acceso a los objetos del sistema permitiéndolo o denegándolo sobre la base de información tal como la identificación del usuario, el tipo de recurso, la pertenencia del usuario a cierto grupo de personas, las operaciones que puede hacer el usuario o el grupo con cada recurso.

La protección consiste en controlar y, en su caso, impedir el acceso de los programas, procesos o usuarios a los recursos del sistema (archivos, memoria, CPU), las cuales son la perdida de datos y los intrusos.

#### Causas de pérdida de datos:

- Actos divinos: Incendios, inundaciones, terremotos, guerras, revoluciones o ratas que roen las cintas o discos flexibles.
- Errores de Hardware o Software: Mal funcionamiento de la CPU, discos o cintas ilegibles, errores de telecomunicación o errores en el programa.
- Errores Humanos: Entrada incorrecta de datos, mal montaje de las cintas o el disco, ejecución incorrecta del programa, perdida de cintas o discos.

La seguridad estudia cómo proteger la información almacenada en el sistema (datos o código) contra accesos indebidos o no autorizados (intrusos, fallos de la privacidad, etc).



Las razones para proveer protección a un sistema operativo son:

La necesidad de prevenirse de violaciones intencionales de acceso por un usuario.

La necesidad de asegurar que cada componente de un programa, use solo los recursos del sistema de acuerdo con las políticas fijadas para el uso de esos recursos.

Para proteger un sistema, debemos optar las necesarias medidas de seguridad en cuatro niveles distintos:

Físico: El nodo o nodos que contengan los sistemas informáticos deben dotarse de medidas de seguridad físicas frente a posibles intrusiones armadas o subrepticias por parte de potenciales intrusos. Hay que dotar de seguridad tanto a las habitaciones donde las maquinas residan como a los terminales o estaciones de trabajo que tengan acceso a dichas maquinas.

Humano: La autorización de los usuarios debe llevarse a cabo con cuidado, para garantizar que solo los usuarios apropiados tengan acceso al sistema. Sin embargo, incluso los usuarios autorizados pueden verse “motivados” para permitir que otros usen su acceso (por ejemplo, a cambio de un soborno). También pueden ser engañados para permitir el acceso de otros, mediante técnicas de ingeniería social.

Uno de los tipos de ataque basado en las técnicas de ingeniería social es el denominado phishing; con este tipo de ataque, un correo electrónico o página web de aspecto auténtico llevan a engaño a un usuario para que introduzca información confidencial. Otra técnica comúnmente utilizada es el análisis de desperdicios, un término autorizado a la computadora (por ejemplo, examinando el contenido de las papeleras, localizando listines de teléfonos encontrando notas con contraseñas). Estos problemas de seguridad son cuestiones relacionadas con la gestión y con el personal, más que problemas relativos a los sistemas operativos.

Sistema operativo: El sistema debe auto protegerse frente a los diversos fallos de seguridad accidentales o premeditados. Un problema que este fuera de control puede llegar a constituir un ataque accidental de denegación de servicio. Asimismo, una cierta consulta a un servicio podría conducir a la revelación de contraseñas o un desbordamiento de la pila podría permitir que se iniciara un proceso no autorizado. La lista de posibles fallos es casi infinita.

Red: Son muchos los datos en los modernos sistemas informáticos que viajen a través de líneas arrendadas privadas, de líneas compartidas como Internet, de conexiones inalámbricas o de líneas de acceso telefónico. La interceptación de estos datos podría ser tan dañina como el acceso a un computador, y la interrupción en la comunicación podría constituir un ataque remoto de denegación de servicio, disminuyendo la capacidad de uso del sistema y la confianza en el mismo por parte de los usuarios.

### Problemas de seguridad

- Usuarios inexpertos o descuidados.
- Usuarios no autorizados.
- Ataques por programa.
- Caballo de Troya.
- Puerta secreta.
- Amenazas al sistema.
- Gusanos.
- Virus.

Posibles efectos de las amenazas

- ❖ Revelación de información no autorizada.
- ❖ Destrucción de información.
- ❖ Utilización indebida de servicios del sistema.



- ❖ Daños físicos al sistema.
- ❖ Degradación en el funcionamiento del sistema.
- ❖ Denegación de acceso a usuarios autorizados.

#### Vigilancia de amenazas

- Buscar patrones de actividad sospechosos.
- Contar las veces que se proporcionan contraseñas incorrectas.
- Explorar el sistema en busca de agujeros.
- Contraseñas cortas o fáciles de adivinar.
- Programas no autorizados en directorios del sistema.
- Procesos con duración inusitada.
- Protecciones inapropiadas en directorios y archivos del sistema.
- Cambios en los programas del sistema.
- Cortafuegos (firewall) en computadores conectados en red.
- Separa los sistemas confiables de los no confiables.
- Limita el acceso por red a determinados dominios.

#### Diferencias entre riesgo y seguridad

La seguridad: es la ausencia de un riesgo. Aplicando esta definición a al tema correspondiente, se hace referencia al riesgo de accesos no autorizados, de manipulación de información, manipulación de las configuraciones, entre otros.

La protección: son los diferentes mecanismos utilizados por el SO para cuidar la información, los procesos, los usuarios, etc.

Un sistema de seguridad debe cumplir con unos requisitos:

- Confidencialidad: Acceso solo a usuarios autorizados.
- Integridad: Modificación solo por usuarios autorizados.
- Disponibilidad: Recursos solamente disponibles para usuario autorizado.

La seguridad se clasifica en:

- Externa: protección contra desastres y contra intrusos.
- Operacional: básicamente nos determina que acceso se permite a quien.

Una de las obligaciones de un sistema seguro es permanecer en constante vigilancia, verificando y validando las posibles amenazas, esto lo hacen con uso de contraseñas, controles de acceso

Se plantea que es más fácil hacer un sistema seguro si esto se ha incorporado desde los inicios del diseño, porque no se puede hablar de un SO seguro si su núcleo no lo es; de igual manera es posible hacer seguridad por hardware donde se obtiene como ventaja la velocidad de operación permitiendo controles más frecuentes y mejora el performance.

Con respecto a los SO más seguros es difícil listarlos ya que todos tienen sus seguidores y contractares los cuales por instinto suelen defender lo que usan, pero es sin duda alguna lo que responden las encuestas hay una de las distribuciones de Linux denominada open BSD que es conocido como el SO más seguro aparte de que no deja de ser software libre, de igual manera es situado a los SO de Windows encima del Mac OSX donde apenas la última versión empieza a aplicar completamente algoritmos de seguridad que desde antes eran utilizados por la competencia pero sin duda alguna los sistemas libres ganan la batalla con respecto a la seguridad

Para poder garantizar la seguridad es fundamental proteger nuestro sistema, por eso básicamente los mecanismos articulados para la protección son los que nos llevan a un sistema seguro; existen diferentes formas de realizar la protección tal vez la más común y más básica sea definir cuáles son los archivos u objetos a proteger para que posteriormente se delimiten que usuarios pueden acceder a que información



Como objetivos de la protección esta:

- Controlar el acceso a los recursos
- Utilización por diferentes usuarios

Generalmente surgen dudas sobre qué es lo que debemos proteger o que debo cuidar más y la respuesta es siempre variable según el tipo de necesidades de cada usuario, pero generalmente los más afectados son la CPU, la memoria, terminales, procesos, archivos y las bases de datos

Un sistema de protección deberá tener la flexibilidad suficiente para poder imponer una diversidad de políticas y mecanismos. La protección se refiere a los mecanismos para controlar el acceso de programas, procesos, o usuarios a los recursos definidos por un sistema de computación. Seguridad es la serie de problemas relativos a asegurar la integridad del sistema y sus datos.

### **Principales elementos de seguridad en los sistemas operativos**

**Contraseñas:** Idealmente, no quieres que tu sistema operativo (OS) vaya directamente al escritorio cuando se inicia el equipo. Es mejor ir a una pantalla donde el usuario tiene que introducir una contraseña.

**Fuerza de la contraseña:** No todas las contraseñas son iguales. No deseas utilizar contraseñas relativamente fáciles de adivinar: cosas como segundos nombres, direcciones, cumpleaños, números de teléfono, códigos postales, o cualquier otra información pública vinculada a ti.

**Cifrado:** El cifrado codifica tus datos para que sólo se puedan leer cuando se proporcione una contraseña. Puedes encriptar archivos individuales y carpetas enteras o "volúmenes" de dispositivos de almacenamiento con software como PGP o TrueCrypt.

**Protección contra malware:** Los virus, troyanos, gusanos y keyloggers son colectivamente conocidos como "malware". Un escáner de antivirus de Symantec, McAfee o Kaspersky puede ayudarte a proteger tu equipo frente a estas amenazas.

**Comportamiento riesgoso:** Estos escáneres no pueden detectar el 100 por ciento de todas las amenazas. Ten mucho cuidado cuando navegas por un sitio web dudoso o abres un archivo adjunto de correo electrónico. Ten cuidado con los correos electrónicos fraudulentos que intentan divulgar tu información personal confidencial.



## REFERENCIAS

*Angelfire.* (6 de Noviembre de 2014). Obtenido de <http://sistemasoperativos.angelfire.com/html/6.1.html>

Anonimo. (5 de Noviembre de 2014). Obtenido de Kioskea: <http://es.kioskea.net/contents/648-gestion-de-memoria>

Anonimo. (Jueves de Noviembre de 2014). *sistemas operativos.* Obtenido de  
<http://mixteco.utm.mx/~resdi/historial/materias/capitulo5.pdf>

Media docencia. (24 de Septiembre de 2014). *Laurel.* Obtenido de  
[http://laurel.datsi.fi.upm.es/\\_media/docencia/asignaturas/dso/seg\\_y\\_prot\\_074pp.pdf](http://laurel.datsi.fi.upm.es/_media/docencia/asignaturas/dso/seg_y_prot_074pp.pdf)

MICROSOFT. (s.f.). *FILE SYSTEM.* Recuperado el 11 de Octubre de 2014, de SOPORTE TECNICO:  
<http://support2.microsoft.com/kb/100108/es>

*Nebrija.* (19 de Septiembre de 2014). Obtenido de <http://www.nebrija.es/~jmaestro/AT3148/Seguridad.pdf>

Serrano, F. (13 de Septiembre de 2014). Obtenido de Blogspot: <http://serranop4030.blogspot.mx/2012/09/tipos-de-memoria-exposicion.html>

*Silberchatz, G. (2002). Sistemas operativos. Mexico: Limusa Wiley. Silberschatz, G. (1999). Sistemas operativos. México.: Adison Wesley.*

*Stalling, W. (2005). Sistemas operativos Aspectos internos y principios de diseño. Madrid España.: Pearson.*

*Tanenbaum, A. (2009). Sistema Operativos Modernos. México: Pearson, Prentice Hall.*