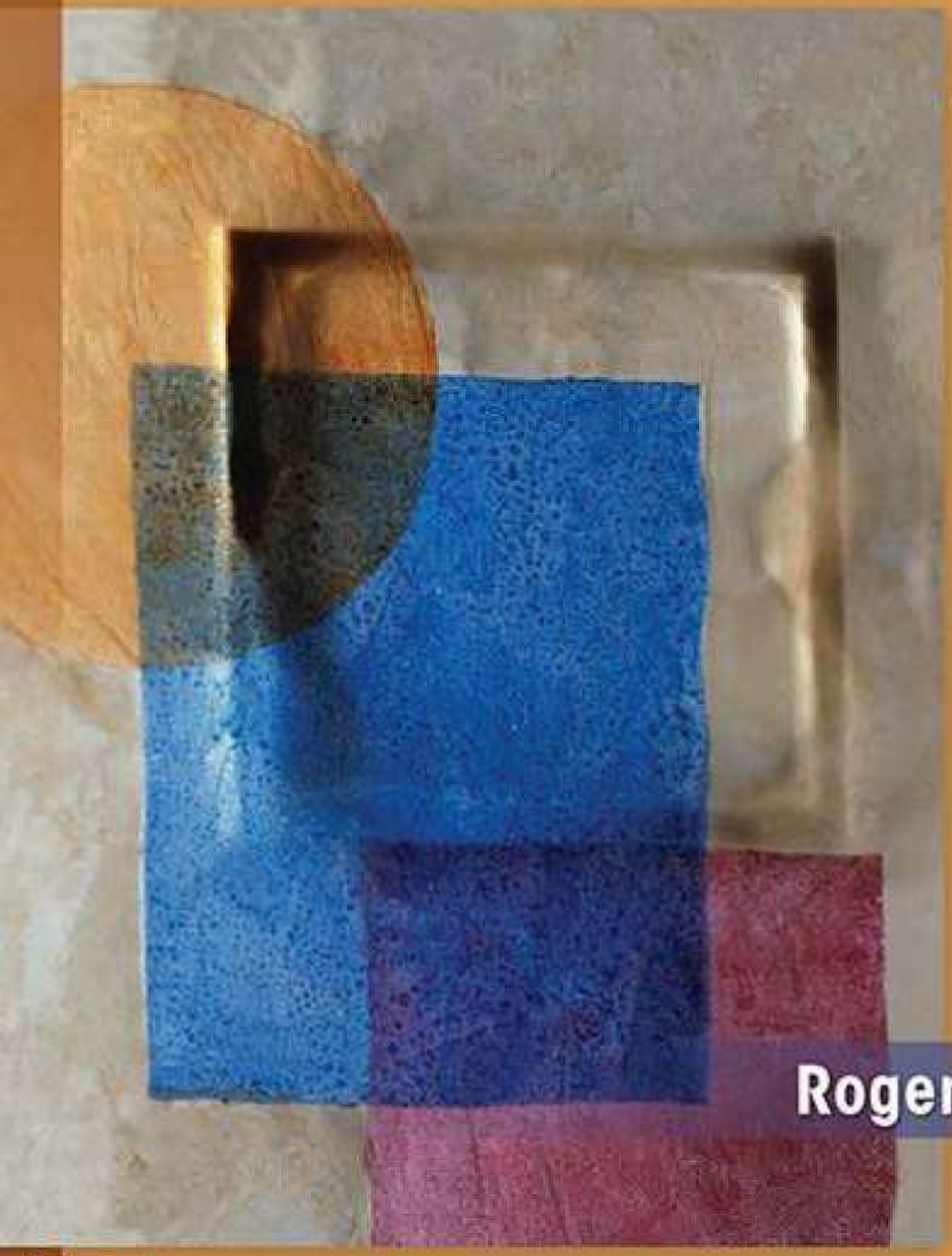
Ingeniería del software

Un enfoque práctico



Séptima edición

Roger S. Pressman

Mc Graw Hill

Ingeniería del software

UN ENFOQUE PRÁCTICO

SÉPTIMA EDICIÓN

Roger S. Pressman, Ph.D.
University of Connecticut

MÉXICO • BOGOTÁ • BUENOS AIRES • CARACAS • GUATEMALA • MADRID
NUEVA YORK • SAN JUAN • SANTIAGO • SÃO PAULO • AUCKLAND • LONDRES • MILÁN
MONTREAL • NUEVA DELHI • SAN FRANCISCO • SINGAPUR • ST. LOUIS • SIDNEY • TORONTO

Director Higher Education: Miguel Ángel Toledo Castellanos
Editor sponsor: Pablo Roig Vázquez
Coordinadora editorial: Marcela I. Rocha Martínez
Editora de desarrollo: María Teresa Zapata Terrazas
Supervisor de producción: Zeferino García García

Traductores: Víctor Campos Olguín
Javier Enríquez Brito
Revisión técnica: Carlos Villegas Quezada
Bárbaro Jorge Ferro Castro

INGENIERÍA DEL SOFTWARE. UN ENFOQUE PRÁCTICO
Séptima edición

Educación

A Subsidiary of ***The McGraw-Hill*** *Companies, Inc.*
Prolongación Paseo de la Reforma 1015, Torre A
Piso 17, Colonia Desarrollo Santa Fe,
Delegación Álvaro Obregón
C.P. 01376, México, D. F.
Miembro de la Cámara Nacional de la Industria Editorial Mexicana, Reg. Núm. 736

ISBN: 978-607-15-0314-5
(ISBN edición anterior: 970-10-5473-3)

Traducido de la séptima edición de SOFTWARE ENGINEERING. A PRACTITIONER'S APPROACH.
Published by McGraw-Hill, a business unit of The McGraw-Hill Companies, Inc., 1221 Avenue of the Americas, New York, NY 10020. Copyright © 2010 by The McGraw-Hill Companies, Inc. All rights reserved.
978-0-07-337597-7

1234567890 109876543210

Impreso en México *Printed in Mexico*

The McGraw-Hill Companies

CAPÍTULO

15 TÉCNICAS DE REVISIÓN

CONCEPTOS CLAVE

Las revisiones son como filtros en el flujo del trabajo del proceso de software. Si son muy pocas, el flujo queda "sucio". Si son demasiadas, se hace lento hasta detenerse. Utilice métricas para determinar cuáles son las revisiones que funcionan y haga énfasis en ellas. Elimine del flujo las revisiones ineficaces, con objeto de acelerar el proceso.

Las revisiones del software son un "filtro" para el proceso del software. Es decir, se aplican en varios puntos durante la ingeniería de software y sirven para descubrir errores y defectos a fin de poder eliminarlos. Las revisiones del software "purifican" los productos del trabajo de la ingeniería de software, incluso los modelos de requerimientos y diseño, código y datos de prueba. Freedman y Weinberg [Fre90] analizan del modo siguiente la necesidad de hacer revisiones:

> El trabajo técnico necesita las revisiones por la misma razón que los lápices necesitan borradores: *errar es humano*. La segunda razón por la que son necesarias las revisiones técnicas es porque, si bien las personas son buenas para detectar algunos de sus propios errores, muchas clases de ellos pasan desapercibidos con más facilidad para quien los comete que para otras personas. Por tanto, este proceso de revisión es la respuesta a la oración de Robert Burns:
>
> *Oh, quiera algún Dios el regalo darnos*
>
> *de vernos a nosotros como los demás nos ven*
>
> Una revisión —cualquiera — es una forma de utilizar la diversidad de un grupo para lo siguiente:
>
> 1. Resaltar las mejoras necesarias en el producto que elaboró una sola persona o equipo;
> 2. Confirme aquellas partes de un producto en las que no se desea o no se necesita hacer una mejora;
> 3. Realice el trabajo técnico de calidad más uniforme, o al menos más predecible, que pueda lograrse sin hacer revisiones, a fin de que el trabajo técnico sea más manejable.

Como parte de la ingeniería de software, pueden realizarse muchos diferentes tipos de revisiones. Cada uno tiene su lugar. Una reunión informal alrededor de la máquina del café es una forma de revisión si se analizan problemas técnicos. La presentación formal de la arquitectura del software a un público de clientes, administradores y técnicos también es una forma de revi-

UNA MIRADA RÁPIDA

¿Qué es? Conforme se desarrollen los productos del trabajo de la ingeniería de software se cometerán errores. No es vergonzoso, mientras se trate de detectarlos y corregirlos con ahínco —con mucho ahínco— antes de que lleguen a los usuarios finales. Las revisiones técnicas son el mecanismo más eficaz para detectar los errores en una etapa temprana del proceso de software.

¿Quién lo hace? Son los ingenieros de software quienes realizan una revisión técnica, también llamada revisión de pares, con sus colegas.

¿Por qué es importante? Si encuentra un error al principio del proceso, es menos caro corregirlo. Además, los errores tienen un modo de amplificarse a medida que avanza el proceso. Por ello, un error relativamente pequeño que se deje sin atender al comenzar el proceso se amplifica en un conjunto más grande de errores en una etapa posterior del proyecto. Finalmente, las revisiones ahorran tiempo, reduciendo la cantidad de repeticiones que se requerirán hacia el final del proyecto.

¿Cuáles son los pasos? El enfoque de las revisiones variará en función del grado de formalidad que se elija. En general, se utilizan seis etapas, aunque no todas se emplean siempre: planeación, preparación, estructurar la reunión, resaltar los errores, hacer las correcciones (fuera de la revisión) y verificar que las correcciones se hayan hecho en forma apropiada.

¿Cuál es el producto final? El resultado de una revisión es una lista de conceptos o errores descubiertos. Además, también se indica el estado técnico del producto final.

¿Cómo me aseguro de que lo hice bien? En primer lugar, seleccione el tipo de revisión que sea apropiada para su cultura de desarrollo. Siga los lineamientos que lleven a ejecutar revisiones exitosas. Si éstas conducen a un software de alta calidad, lo habrá hecho bien.

sión. Sin embargo, en este libro nos centramos en las *revisiones técnicas o por pares*, ejemplificadas por las *revisiones casuales,* walkthroughs e *inspecciones*. Desde el punto de vista del control de calidad, una revisión técnica (RT) es el filtro más eficaz. Realizado por ingenieros de software (y de otro tipo) para ingenieros de software, la RT es un medio eficaz para detectar errores y mejorar la calidad.

15.1 Efecto de los defectos del software en el costo

En el contexto del proceso del software, los términos *defecto* y *falla* son sinónimos. Los dos implican un problema de calidad descubierto *después* de haberse liberado el software a los usuarios finales (o a otra actividad estructural del proceso del software). En capítulos anteriores se empleó el término *error* para denotar un problema de calidad descubierto por ingenieros de software (o de otra clase) *antes* de entregar el software al usuario final (o a alguna actividad estructural del proceso del software).

Información

Equivocaciones, errores y defectos

La meta del control de calidad del software, y en un sentido más amplio de la administración de la calidad en general, es eliminar los problemas de calidad que se encuentren en el software. Se hace referencia a estos problemas con diferentes nombres: *equivocaciones*, *fallas*, *errores* o *defectos*, por mencionar algunos. ¿Son sinónimos estos términos o hay diferencias sutiles entre ellos?

En este libro se hace una distinción clara entre un error (problema de calidad que se detecta antes de que el software se entregue a los usuarios finales) y un defecto (problema de calidad que se encuentra después de haber entregado el software a los usuarios finales[1]). Esta distinción se hace porque los errores y defectos tienen muy distinto efecto económico, empresarial, sicológico y humano. Como ingenieros de software, queremos encontrar y corregir tantos errores como sea posible antes de que el consumidor o el usuario final los encuentren. Queremos evitar los defectos porque hacen (justificadamente) que el personal de software se vea mal.

Sin embargo, es importante observar que la distinción temporal entre errores y defectos que se hace en este libro no constituye la principal forma de pensar. El consenso general de la comunidad de ingeniería de software es que defectos, errores, fallas y equivocaciones son sinónimos. Es decir, el punto en el tiempo en el que se encontró el problema no tiene que ver con el término que se usa para describirlo. Parte de la argumentación a favor de este punto de vista es que en ocasiones es difícil hacer una distinción clara entre el antes y el después de la liberación (por ejemplo, considere un proceso incremental en un desarrollo ágil).

Sin que importe el modo en el que se elija interpretar estos términos, hay que reconocer que el momento en el que se descubre un problema sí importa, y que los ingenieros de software deben tratar de detectar con ahínco —con mucho ahínco— los problemas antes de que sus clientes y usuarios finales los encuentren. Si el lector está más interesado en este tema, puede hallar un análisis razonablemente completo de la terminología acerca de las "equivocaciones" en la dirección **www.softwaredevelopment.ca/bugs.shtml**

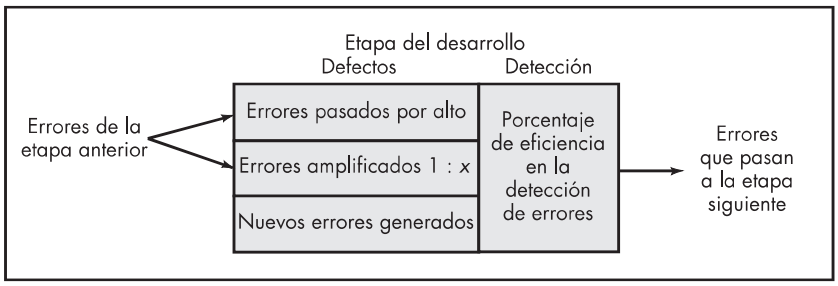
El objetivo principal de una revisión técnica formal es detectar los errores antes de que pasen a otra actividad de la ingeniería de software o de que se entreguen al usuario final.

El objetivo principal de las revisiones técnicas es encontrar errores durante el proceso a fin de que no se conviertan en defectos después de liberar el software. El beneficio obvio de las revisiones técnicas es el descubrimiento temprano de los errores, de modo que no se propaguen a la siguiente etapa del proceso del software.

Varios estudios de la industria indican que las actividades de diseño introducen de 50 a 65 por ciento de todos los errores (y en realidad de todos los defectos) durante el proceso del software. Sin embargo, las técnicas de revisión han demostrado tener una eficacia de hasta 75 por ciento [Jon86] para descubrir fallas del diseño. Al detectar y eliminar un gran porcentaje de estos

1 Si se considera una mejora en el proceso del software, un problema de calidad que se propague de una actividad estructural del proceso (como el **modelado**) a otra (como la **construcción**) también se llama "defecto", porque debe encontrarse el problema antes de que un producto del trabajo (como un modelo del diseño) se "libere" a la siguiente actividad.

FIGURA 15.1
Modelo de amplificación del defecto



errores, el proceso de revisión reduce de manera sustancial el costo de las actividades posteriores en el proceso del software.

15.2 AMPLIFICACIÓN Y ELIMINACIÓN DEL DEFECTO

Cita:

"Dicen los médicos que en sus inicios algunas enfermedades son fáciles de curar pero difíciles de reconocer… mas con el paso del tiempo, si no se detectaron y trataron al principio, se vuelven fáciles de reconocer pero difíciles de curar."

Nicolás Maquiavelo

Para ilustrar la generación y detección de errores durante las acciones de diseño y generación de código de un proceso de software, puede usarse un *modelo de amplificación del defecto* [IBM81]. En la figura 15.1 se ilustra esquemáticamente el modelo. Un cuadro representa una acción de la ingeniería de software. Durante la acción, los errores se generan de manera inadvertida. La revisión puede fracasar en descubrir los errores nuevos que se generan y los cometidos en etapas anteriores, lo que da como resultado cierto número de errores pasados por alto. En ciertos casos, los errores de etapas anteriores ignorados son amplificados (en un factor x de amplificación) por el trabajo en curso. Las subdivisiones de los cuadros representan a cada una de estas características y al porcentaje de eficiencia de la detección de errores, que es una función de la profundidad de la revisión.

La figura 15.2 ilustra un ejemplo hipotético de amplificación del defecto para un proceso de software en el que no se hacen revisiones. En la figura, se supone que en cada etapa de prueba se detecta y corrige 50 por ciento de todos los errores de entrada sin que se introduzcan nuevos errores (suposición optimista). Diez defectos preliminares de diseño se amplifican a 94 errores antes de que comiencen las pruebas. Se liberan al campo 12 errores latentes (defectos). La figura 15.3 considera las mismas condiciones, excepto porque se efectúan revisiones del diseño y código como parte de cada acción de la ingeniería de software. En este caso, son 10 los errores

FIGURA 15.2
Amplificación del defecto. Sin revisiones

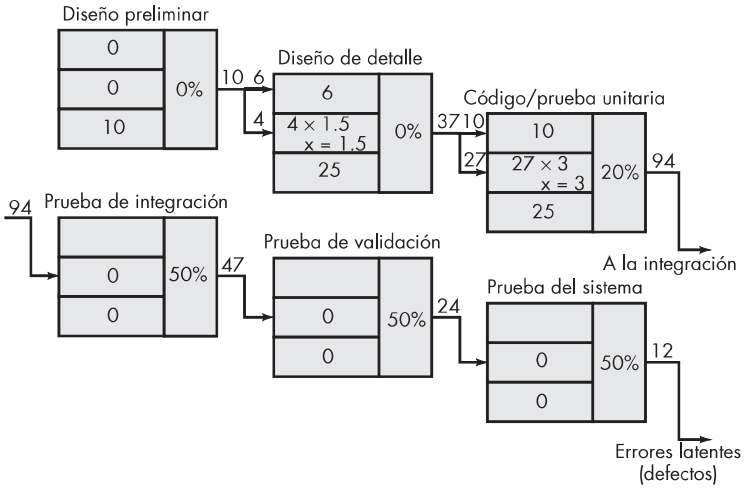
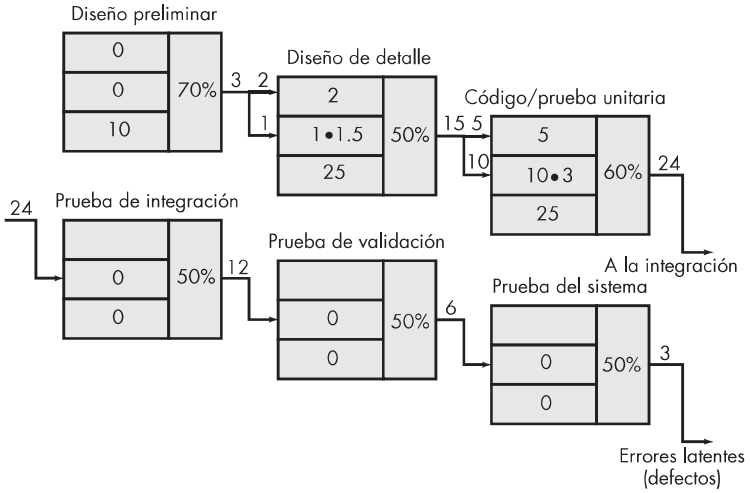


FIGURA 15.3
Amplificación del defecto. Se efectúan revisiones



iniciales de diseño preliminar (arquitectura) que se amplifican a 24 antes de comenzar las pruebas. Sólo existen tres errores latentes. Pueden establecerse los costos relativos asociados con el descubrimiento y corrección de errores, así como el costo general (con y sin revisión para nuestro ejemplo hipotético). El número de errores detectados durante cada una de las etapas citadas en las figuras 15.2 y 15.3 se multiplica por el costo que implica eliminar un error (1.5 unidades de costo para el diseño, 6.5 unidades de costo antes de las pruebas, 15 unidades de costo durante las pruebas y 67 unidades de costo después de la entrega).[2] Con estos datos, el costo total del desarrollo y mantenimiento cuando se efectúan revisiones es de 783 unidades de costo. Cuando no se hacen revisiones, el costo total es de 2 177 unidades, casi tres veces más caro.

Debe dedicarse tiempo y esfuerzo a la realización de revisiones y su organización de desarrollo debe destinar el dinero para ello. Sin embargo, los resultados del ejemplo anterior dejan pocas dudas acerca de lo que puede pagar ahora o de que después deberá pagar mucho más.

15.3 MÉTRICAS DE REVISIÓN Y SU EMPLEO

Las revisiones técnicas son una de las muchas acciones que se requieren como parte de las buenas prácticas de la ingeniería de software. Cada acción requiere un esfuerzo humano dirigido. Como el esfuerzo disponible para el proyecto es finito, es importante que una organización de software comprenda la eficacia de cada acción, definiendo un conjunto de métricas (véase el capítulo 23) que puedan utilizarse para evaluar esa eficacia.

Aunque se han definido muchas métricas para las revisiones técnicas, un conjunto relativamente pequeño da una perspectiva útil. Las siguientes métricas para la revisión pueden obtenerse conforme se efectúe ésta:

- *Esfuerzo de preparación,* E_p: esfuerzo (en horas-hombre) requerido para revisar un producto del trabajo antes de la reunión de revisión real.
- *Esfuerzo de evaluación,* E_a: esfuerzo requerido (en horas-hombre) que se dedica a la revisión real.
- *Esfuerzo de la repetición,* E_r: esfuerzo (en horas-hombre) que se dedica a la corrección de los errores descubiertos durante la revisión.

2 Estos multiplicadores son algo diferentes de los datos presentados en la figura 14.2, que es más actual. Sin embargo, sirven para ilustrar los costos de la amplificación del defecto.

- *Tamaño del producto del trabajo, TPT*: medición del tamaño del producto del trabajo que se ha revisado (por ejemplo, número de modelos UML o número de páginas de documento o de líneas de código).
- *Errores menores detectados, $Err_{menores}$*: número de errores detectados que pueden clasificarse como menores (requieren menos de algún esfuerzo especificado para corregirse).
- *Errores mayores detectados, $Err_{mayores}$*: número de errores encontrados que pueden clasificarse como mayores (requieren más que algún esfuerzo especificado para corregirse).

Estas métricas pueden mejorarse, asociando el tipo de producto del trabajo que se revisó con las métricas obtenidas.

15.3.1 Análisis de las métricas

Antes de comenzar el análisis deben hacerse algunos cálculos sencillos. El esfuerzo total de revisión y el número total de errores descubiertos se definen como sigue:

$$E_{\text{revisión}} = E_p + E_a + E_r$$

$$\text{Err}_{\text{tot}} = \text{Err}_{\text{menores}} + \text{Err}_{\text{mayores}}$$

La *densidad del error* representa los errores encontrados por unidad de producto del trabajo revisada.

$$\text{Densidad del error} = \frac{\text{Err}_{\text{tot}}}{\text{TPT}}$$

Por ejemplo, si se revisa un modelo de requerimientos con objeto de encontrar errores, inconsistencias y omisiones, es posible calcular la densidad del error en varias formas diferentes. El modelo de requerimientos contiene 18 diagramas UML como parte de 32 páginas de materiales descriptivos. La revisión detecta 18 errores menores y 4 mayores. Por tanto, $\text{Err}_{\text{tot}} = 22$. La densidad del error es 1.2 errores por diagrama UML o 0.68 errores por página del modelo de requerimientos.

Si las revisiones se llevan a cabo para varios tipos distintos de productos del trabajo (por ejemplo, modelo de requerimientos, modelo del diseño, código, casos de prueba, etc.), el porcentaje de errores no descubiertos por cada revisión se confronta con el número total de errores detectados en todas las revisiones. Además, puede calcularse la densidad del error para cada producto del trabajo.

Una vez recabados los datos para muchas revisiones efectuadas en muchos proyectos, los valores promedio de la densidad del error permiten estimar el número de errores por hallar en un nuevo documento (aún no revisado). Por ejemplo, si la densidad promedio de error para un modelo de requerimientos es de 0.6 errores por página, y un nuevo modelo de requerimientos tiene una longitud de 32 páginas, una estimación gruesa sugiere que el equipo de software encontrará alrededor de 19 o 20 errores durante la revisión del documento. Si sólo encuentra 6 errores, habrá hecho un trabajo extremadamente bueno al desarrollar el modelo de requerimientos o su enfoque de la revisión no fue tan profundo.

Una vez llevada a cabo la prueba (véanse los capítulos 17 a 20), es posible obtener datos adicionales del error, incluso el esfuerzo requerido para detectar y corregir errores no descubiertos durante las pruebas y la densidad del error del software. Los costos asociados con la detección y corrección de un error durante las pruebas pueden compararse con los de las revisiones. Esto se analiza en la sección 15.3.2.

15.3.2 Eficacia del costo de las revisiones

Es difícil medir en tiempo real la eficacia del costo de cualquier revisión técnica. Una organización de ingeniería de software puede evaluar la eficacia de las revisiones y su relación costo-

beneficio sólo después de que éstas han terminado, de que las unidades de medida de la revisión se han recabado, de que los datos promedio han sido calculados y de que la calidad posterior del software ha sido medida (mediante pruebas).

Si regresamos al ejemplo presentado en la sección 15.3.1, se determinó que la densidad promedio del error para los modelos de requerimientos era de 0.6 errores por página. Se reveló que el esfuerzo requerido para corregir un error menor en el modelo era de 4 horas-hombre. Se vio que el esfuerzo necesario para un error mayor en los requerimientos era de 18 horas-hombre. Al estudiar los datos recabados se observa que los errores menores ocurrieron con una frecuencia cercana a 6 veces más que los errores mayores. Por tanto, puede estimarse que el esfuerzo promedio para detectar y corregir un error en los requerimientos durante la revisión es alrededor de 6 horas-hombre.

Los errores relacionados con los requerimientos no detectados durante las pruebas requieren un promedio de 45 horas-hombre para encontrarse y corregirse (no hay datos disponibles acerca de la severidad relativa del error). Con estos promedios se obtiene lo siguiente:

$$\text{Esfuerzo ahorrado por error} = E_{\text{pruebas}} - E_{\text{revisiones}}$$
$$45 - 6 = 30 \text{ horas-hombre/error}$$

Como durante la revisión del modelo de requerimientos se encontraron 22 errores, se tendrá un ahorro cercano a 660 horas-hombre en el esfuerzo dedicado a las pruebas. Y esto se refiere sólo a los errores relacionados con los requerimientos. Al beneficio general se suman aquellos asociados con el diseño y el código. El esfuerzo total conduce a ciclos de entrega más cortos y a un mejor tiempo para llegar al mercado.

En su libro sobre la revisión por pares, Karl Wiegers [Wie02] analiza datos procedentes de anécdotas de compañías grandes que han utilizado *inspecciones* (un tipo relativamente formal de revisión técnica) como parte de sus actividades de control de calidad del software. Hewlett Packard reportó un rendimiento de 10 a 1 sobre la inversión gracias a las inspecciones y afirmó que la entrega real del producto se aceleró en un promedio de 1.8 meses-calendario. AT&T indicaba que las inspecciones habían reducido el costo general de los errores de software en un factor de 10, que la calidad había mejorado en un orden de magnitud y que la productividad se había incrementado 14 por ciento. Otras empresas reportaban beneficios similares. Las revisiones técnicas (en diseño y otras actividades) generan una buena relación costo-beneficio y en verdad ahorran tiempo.

Pero para muchos profesionales del software, esta afirmación va contra la intuición. "Las revisiones toman tiempo", dicen, "y no tenemos tiempo que perder...". Afirman que el tiempo es precioso en cada proyecto de software y que la actividad de revisar "todo producto del trabajo con detalle" absorbe demasiado.

Los ejemplos presentados en esta sección indican otra cosa. Lo más importante es que los datos de la industria sobre revisiones del software se han recabado durante más de dos décadas y se resumen cualitativamente en las gráficas que aparecen en la figura 15.4

En la figura, el trabajo efectuado cuando se utilizan revisiones se refleja pronto en el desarrollo de un incremento de software, pero esta inversión temprana paga dividendos debido a que se reduce el esfuerzo necesario para hacer pruebas y correcciones. De igual importancia es que la fecha de entrega del desarrollo con revisiones ocurre antes que la que se hace sin revisiones. ¡Las revisiones no quitan tiempo, lo ahorran!

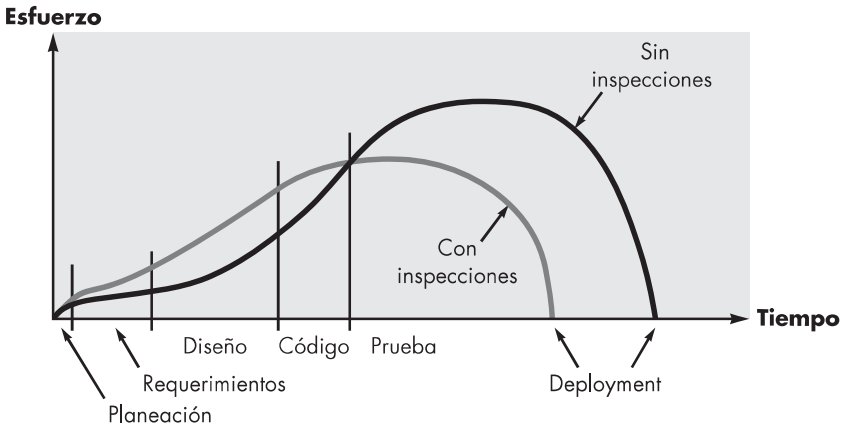
15.4 Revisiones: espectro de formalidad

Las revisiones técnicas deben aplicarse con un nivel de formalidad apropiado para el producto que se va a elaborar, para el plazo que tiene el proyecto y para el personal que realice el trabajo.

FIGURA 15.4

Esfuerzo realizado, con y sin revisiones

Fuente: adaptado de [Fog86].



La figura 15.5 ilustra un modelo de referencia para las revisiones técnicas [Lai02] que identifica cuatro características que contribuyen a la formalidad con la que se efectúa una revisión.

Cada una de las características del modelo de referencia ayuda a definir el nivel de formalidad de la revisión. La formalidad de una revisión se incrementa cuando: 1) se definen explícitamente roles distintos para los revisores, 2) hay suficiente cantidad de planeación y preparación para la revisión, 3) se define una estructura distinta para la revisión (incluso tareas y productos internos del trabajo) y 4) el seguimiento por parte de los revisores tiene lugar para cualesquiera correcciones que se efectúen.

Para entender el modelo de referencia, supongamos que el lector decidió revisar el diseño de la interfaz para **CasaSeguraAsegurada.com**. Esto puede hacerse de varias maneras diferentes, que van de lo relativamente casual a lo riguroso en extremo. Si decide que el enfoque casual es más apropiado, se pide a algunos colegas (pares) que examinen el prototipo de la interfaz en un esfuerzo por descubrir problemas potenciales. Todos deciden que no habrá preparación previa, pero que evaluarán el prototipo en una forma razonablemente estructurada: primero verán la distribución, luego la estética, después las opciones de navegación, etc. Como diseñador que es, el lector decide tomar algunas notas, pero nada formales.

Pero, ¿qué pasa si la interfaz es crucial para el éxito de todo el proyecto? ¿Qué sucede si de lo acertado de su ergonomía dependen vidas humanas? Debió concluirse que era necesario un enfoque más riguroso. Se forma entonces el equipo de revisión. Cada integrante de éste tendrá

FIGURA 15.5

Modelo de referencia para hacer revisiones técnicas

un rol específico: dirigir el equipo, registrar las reuniones, presentar el material, etc. Cada revisor tendrá acceso al producto del trabajo (en este caso, el prototipo de la interfaz) antes de que la revisión tenga lugar y dedicará tiempo a la búsqueda de errores, inconsistencias y omisiones. Se realizará un conjunto de tareas específicas con base en una agenda que se desarrollará antes de que ocurra la revisión. Los resultados de ésta serán registrados de manera formal y el equipo decidirá sobre el estado del producto del trabajo con base en el resultado de la revisión. Los miembros del equipo también verificarán que las correcciones se hagan de manera adecuada.

En este libro se consideran dos grandes categorías de revisiones técnicas: revisiones informales y revisiones técnicas más formales. Dentro de cada una de ellas se escogen varios enfoques diferentes. Éstos se presentan en las secciones que siguen.

15.5 Revisiones informales

Las revisiones informales incluyen una simple verificación de escritorio de un trabajo de ingeniería de software, hecha con algún colega, o una reunión casual (con más de dos personas) con objeto de revisar un producto o aspectos orientados a la revisión de programación por pares (véase el capítulo 3).

Una *verificación de escritorio* simple o una *reunión casual* realizada con un colega constituye una revisión. Sin embargo, como no hay una planeación o preparación por adelantado, ni agenda o estructura de la reunión, y no se da seguimiento a los errores descubiertos, la eficacia de tales revisiones es mucho menor que la de los enfoques más formales. Pero una verificación de escritorio sencilla descubre errores que de otro modo se propagarían en el proceso del software.

Una forma de mejorar la eficacia de una verificación de escritorio es desarrollar un conjunto de listas de revisión para cada producto grande del trabajo generado por el equipo de software. Las preguntas que se plantean en la lista son generales, pero servirán para guiar a los revisores en la verificación del producto. Por ejemplo, veamos una verificación de escritorio del prototipo de la interfaz de **CasaSeguraAsegurada.com**. En vez de sólo jugar con el prototipo en la estación de trabajo del diseñador, éste y un colega lo examinan con el empleo de una lista para interfaces:

- ¿La distribución está diseñada con el empleo de convenciones estándar? ¿De izquierda a derecha? ¿De arriba abajo?
- ¿La presentación necesita ser desplazada verticalmente?
- ¿Se usan con eficacia el color y la ubicación, la tipografía y el tamaño?
- ¿Todas las opciones o funciones de navegación están representadas en el mismo nivel de abstracción?
- ¿Están etiquetadas con claridad todas las elecciones de navegación?

y así sucesivamente. Cualesquiera errores o aspectos señalados por los revisores son registrados por el diseñador para resolverlos tiempo después. Las verificaciones de escritorio se programan en forma *ad hoc* o son obligatorias como parte de las buenas prácticas de la ingeniería de software. En general, la cantidad de material por revisar es relativamente pequeña y el tiempo total dedicado a una revisión de escritorio es de poco más de una hora o dos.

En el capítulo 3 se describió la *programación por pares* en la forma siguiente: "La XP recomienda que dos personas trabajen juntas en una estación de trabajo con objeto de crear el código de una narración. Esto proporciona un mecanismo para resolver problemas y asegurar la calidad en tiempo real (dos cabezas piensan más que una)."

La programación por pares se caracteriza por una verificación de escritorio continua. En vez de programar una revisión en algún momento dado, la programación por pares invita a hacer

una revisión continua a medida que se crea el producto (diseño o código). El beneficio es el inmediato descubrimiento de los errores y, en consecuencia, la mejora de la calidad del producto.

En su estudio sobre la eficacia de la programación por pares, Williams y Kessler [Wil00] afirman lo siguiente:

> Las evidencias anecdóticas e iniciales señalan que la programación por pares es una técnica poderosa para generar productivamente trabajos de software de alta calidad. Los elementos de la pareja laboran y comparten sus ideas para resolver las complejidades del desarrollo del software. Realizan de manera continua inspecciones de lo que hace cada quien, lo que conduce a una forma de eliminación de defectos más rápida y eficiente. Además, se mantienen centrados intensamente en la tarea uno del otro.

Algunos ingenieros de software dicen que la redundancia inherente construida en la programación por parejas es un desperdicio de recursos. Después de todo, ¿por qué asignar dos personas a un trabajo que podría ejecutar sólo una? La respuesta a esta pregunta se encuentra en la sección 15.3.2. Si la calidad del producto del trabajo generado como consecuencia de la programación en parejas es mucho mejor que el trabajo de un individuo, los ahorros relacionados con la calidad justifican de sobra la "redundancia" implícita en la programación por parejas.

INFORMACIÓN

Listas de verificación para revisión

Aun cuando las revisiones estén bien organizadas y se lleven a cabo de manera apropiada, no es mala idea dar a los revisores una "criba". Es decir, es útil tener una lista de verificación que dé a cada revisor las preguntas que debe plantear acerca del producto específico del trabajo que se revisa.

Una de las listas más completas es la desarrollada por la NASA en el Centro Goddard de Vuelos Espaciales, disponible en la dirección **http://sw-assurance.gsfc.nasa.gov/disciplines/quality/index.php**

Hay otras listas útiles de revisión técnica que han sido propuestas por las siguientes entidades:

Process Impact (**www.processimpact.com/pr_goodies.shtml**)

Software Dioxide (**www.softwaredioxide.com/Channels/ConView.asp?id=6309**)

Macadamian (**www.macadamian.com**)

The Open Group Architecture Review Checklist (**www.opengroup.org/architecture/togaf7-doc/arch/p4/comp/clists/syseng.htm**)

DFAS (puede descargarse, **www.dfas.mil/technology/pal/ssps/docstds/spm036.doc**)

15.6 REVISIONES TÉCNICAS FORMALES

Cita:

"No hay nada más urgente para alguien que corregir el trabajo de los demás."

Mark Twain

Una *revisión técnica formal* (RTF) es una actividad del control de calidad del software realizada por ingenieros de software (y otras personas). Los objetivos de una RTF son: 1) descubrir los errores en funcionamiento, lógica o implementación de cualquier representación del software; 2) verificar que el software que se revisa cumple sus requerimientos; 3) garantizar que el software está representado de acuerdo con estándares predefinidos; 4) obtener software desarrollado de manera uniforme y 5) hacer proyectos más manejables. Además, la RTF sirve como método de capacitación, pues permite que los ingenieros principiantes observen distintos enfoques de análisis, diseño e implementación del software. La RTF también funciona para estimular el respaldo y la continuidad debido a que varias personas se familiarizan con software que de otra manera no hubieran visto.

La RTF en realidad es una clase que incluye *walkthroughs* e *inspecciones*. Cada RTF se realiza como una reunión y tendrá éxito sólo si se planea, controla y ejecuta en forma apropiada. En las secciones que siguen se presentan lineamientos similares a aquellos usados para un walkthrough, como representativos de la revisión técnica formal. Si el lector tiene interés en las ins-

pecciones de software y en obtener más información sobre walkthroughs, consulte [Rad02], [Wie02] o [Fre90].

15.6.1 La reunión de revisión

WebRef

El documento *Formal Inspection Guidebook*, de la NASA, puede descargarse del sitio **satc.gsfc.nasa.gov/Documents/fi/gdb/fi.pdf**

Sin importar cuál formato de RTF se elija, cualquiera de ellos debe cumplir las restricciones siguientes:

- En la revisión deben involucrarse de tres a cinco personas (normalmente).
- Debe haber preparación previa, pero no debe exigir más de dos horas de trabajo de cada persona.
- La duración de la reunión de revisión debe ser de al menos dos horas.

Dadas estas restricciones, debe resultar obvio que una RTF se centra en una parte específica (y pequeña) del software general. Por ejemplo, en vez de tratar de revisar todo el diseño, se hacen walkthrougs para cada componente o grupo pequeño de componentes. Al reducir el alcance, la RTF tiene mayor probabilidad de detectar errores.

Una RTF se centra en una parte relativamente pequeña de un producto del trabajo.

La atención de la RTF se dirige a un producto (por ejemplo, una parte del modelo de requerimientos, el diseño detallado de un componente o su código fuente, etc.). El individuo que haya desarrollado el producto —el *productor*— informa al líder del proyecto que ha terminado y que se requiere hacer una revisión. El líder del proyecto contacta al *líder de la revisión*, quien evalúa el producto en cuanto a su conclusión, genera copias de los materiales del producto y las distribuye a dos o tres *revisores* para la preparación previa. Se espera que cada revisor dedique de una a dos horas a la inspección del producto, tome notas y se familiarice con el trabajo. Al mismo tiempo, el líder del proyecto también revisa el producto y establece una agenda para la reunión de revisión, que por lo general se programa para el día siguiente.

En ciertas situaciones, es buena idea que alguien distinto del productor haga el walkthroug del producto que se revisa. Esto lleva a una interpretación literal del producto y a mejorar el reconocimiento de errores.

A la reunión de revisión acuden el líder de ésta, todos los revisores y el productor. Uno de los revisores adopta el rol de *secretario*, es decir, quien registra (por escrito) todos los acontecimientos importantes que surjan durante la revisión. La RTF comienza con el análisis de la agenda y una introducción breve por parte del productor. Después, éste procede a "recorrer" el producto del trabajo, explicando el material, mientras los revisores hacen sus comentarios con base en la preparación que hicieron. Cuando se descubren problemas o errores válidos, el secretario toma nota de ellos.

Al terminar la revisión, todos los asistentes deben decidir si: 1) aceptan el producto sin modificaciones, 2) lo rechazan debido a errores graves (una vez corregidos, se realiza otra revisión) o 3) aceptan el producto de manera provisional (se encontraron errores menores que deben corregirse, pero no se necesita otra revisión). Una vez tomada la decisión, todos los asistentes a la RTF firman el acta que indica su participación y su acuerdo con los descubrimientos del equipo de revisión.

15.6.2 Reporte y registro de la revisión

Durante la RTF, un revisor (el secretario) registra activamente todos los asuntos que se planteen. Éstos se resumen al final de la reunión y se produce la *lista de pendientes de la revisión*. Además se elabora un *reporte técnico formal de la revisión*. Éste responde tres preguntas:

1. ¿Qué fue lo que se revisó?
2. ¿Quién lo revisó?
3. ¿Cuáles fueron los descubrimientos y las conclusiones?

El resumen del reporte de la revisión es una sola página (quizá con anexos) que se vuelve parte del registro histórico del proyecto y se entrega al líder del proyecto y a otras partes interesadas.

La lista de pendientes de la revisión tiene dos propósitos: 1) identificar áreas de problemas en el producto y 2) servir como lista de verificación de acciones que guíe al productor cuando se hagan las correcciones. La lista de pendientes normalmente se anexa al reporte técnico.

Debe establecerse un procedimiento de seguimiento para garantizar que los pendientes de la lista se corrijan de manera apropiada. A menos que esto se haga, es posible que los pendientes anotados "se pierdan en el camino". Un enfoque consiste en asignar la responsabilidad del seguimiento al líder del proyecto.

15.6.3 Lineamientos para la revisión

No señale los errores en forma grosera. Una manera amable de hacerlo es plantear preguntas que lleven al productor a descubrir el error.

Los lineamientos para efectuar revisiones técnicas formales deben establecerse por adelantado, distribuirse a todos los revisores, llegar al consenso y, después, seguirse. Una revisión sin control con frecuencia es peor que si no se hiciera ninguna. Los siguientes representan un conjunto mínimo de lineamientos para hacer revisiones técnicas formales:

1. *Revise el producto, no al productor.* Una RTF involucra personas y sus egos. Si se lleva a cabo en forma adecuada, la RTF debe dejar en todos los participantes una sensación de logro. Si se efectúa de modo inapropiado, adopta un aire inquisitorial. Los errores deben señalarse en forma amable; el tono de la reunión debe ser relajado y constructivo; el trabajo no debe apenar o menospreciar a nadie. El líder de la revisión debe conducir la reunión en tono y actitud apropiados y debe detenerla de inmediato si se sale de control.
2. *Establezca una agenda y sígala*. Una de las fallas clave de las reuniones de todo tipo es la dispersión. Una RTF debe mantenerse encarrilada y dentro del programa. El líder de la revisión tiene la responsabilidad de que así sea y no debe sentir temor de llamar al orden a las personas cuando se dispersen.
3. *Limite el debate y las contestaciones.* Cuando el revisor plantee un asunto, quizá no haya acuerdo universal acerca de su efecto. En vez de perder tiempo en debatir la cuestión, ésta debe registrarse para discutirla después.
4. *Enuncie áreas de problemas, pero no intente resolver cada uno.* Una revisión no es una sesión para resolver problemas. Es frecuente que la solución de un problema la obtenga el productor, solo o con ayuda de otra persona. La solución de los problemas debe posponerse para después de la reunión de revisión.
5. *Tome notas por escrito.* A veces es buena idea que el secretario tome notas en un pizarrón a fin de que la redacción y prioridades sean evaluadas por los demás revisores a medida que la información se registra. De manera alternativa, pueden tomarse notas directamente en una computadora.
6. *Limite el número de participantes e insista en la preparación previa.* Dos cabezas piensan más que una, pero 14 no son necesariamente mejor que 4. Mantenga limitado el número de personas involucradas. Sin embargo, todos los miembros del equipo de revisión deben prepararse. El líder de la revisión tiene que solicitar comentarios por escrito (lo que proporciona un indicador de que el revisor ha inspeccionado el material).
7. *Desarrolle una lista de verificación para cada producto que sea probable que se revise.* Una lista de verificación ayuda al líder del proyecto a estructurar la RTF y a cada revisor a centrarse en los aspectos importantes. Deben desarrollarse listas para los productos del análisis, el diseño, el código e incluso las pruebas.
8. *Asigne recursos y programe tiempo para las RTF.* Para que las revisiones sean eficaces, deben programarse como tareas del proceso de software. Además, debe programarse tiempo para hacer las inevitables modificaciones que ocurrirán como resultado de la RTF.

Cita:

"Una reunión es muy frecuentemente un evento en el cual los minutos son tomados y las horas son gastadas."

Autor desconocido

Cita:

"Es una de las más hermosas compensaciones de la vida, que ningún hombre pueda sinceramente ayudar a otro sin ayudarse a sí mismo."

Ralph Waldo Emerson

9. *Dé una capacitación significativa a todos los revisores.* Para que una revisión sea eficaz, todos los revisores deben recibir cierta capacitación formal. Ésta debe hacer énfasis tanto en aspectos relacionados con el proceso como en el lado de la sicología humana de la revisión. Freedman y Weinberg [Fre90] estiman en un mes la curva de aprendizaje para que 20 personas participen de modo eficaz en una revisión.
10. *Revise las primeras revisiones.* Volver a revisar puede ser benéfico para descubrir problemas con el proceso de revisión en sí mismo. El primer producto por revisar deben ser los lineamientos de la revisión.

Debido a que son muchas las variables (número de participantes, tipo de productos del trabajo, tiempo y duración, enfoque específico de la revisión, etc.) que influyen en que una revisión sea exitosa, la organización de software debe experimentar para determinar el enfoque que mejor funcione en el contexto local.

15.6.4 Revisiones orientadas al muestreo

Idealmente, todo producto del trabajo de la ingeniería de software debe pasar por una revisión técnica. En el mundo real de los proyectos de software, los recursos son limitados y el tiempo, escaso. En consecuencia, es frecuente que las revisiones se omitan, aun cuando se reconozca su valor como un mecanismo de control de calidad.

Thelin *et al.* [The01] sugieren un proceso de revisión orientado al muestreo en el que se toman muestras de todos los productos del trabajo de ingeniería de software a fin de inspeccionarlos para determinar cuáles son más susceptibles de tener errores. Después se enfocan todos los recursos de la RTF sólo en aquellos productos en los que sea muy probable encontrar errores (con base en los datos obtenidos durante el muestreo).

Las revisiones toman tiempo, y éste estará bien invertido. Sin embargo, si hay poco tiempo y no hay otra opción, no omita las revisiones. En vez de ello, aplique la revisión orientadas al muestreo.

Para que sea eficaz, el proceso de revisión orientada al muestreo debe tratar de identificar aquellos productos del trabajo que sean objetivos principales para hacer la RTF. Para lograrlo se sugiere seguir las etapas siguientes [The01]:

1. Inspeccionar una fracción a_i de cada producto del trabajo i. Registrar el número de fallas f_i encontradas dentro de a_i.
2. Desarrollar una estimación gruesa del número de fallas en el producto del trabajo i, con la multiplicación de f_i por $1/a_i$.
3. Ordenar los productos del trabajo en orden descendente de acuerdo con la estimación gruesa del número de fallas que hay en cada uno.
4. Dedicar los recursos disponibles para la revisión a aquellos productos que tengan el número estimado más grande de fallas.

La fracción del producto del trabajo de la que se tomen muestras debe ser representativa del producto del trabajo total y suficientemente grande a fin de que tenga significado para todos los revisores que no hagan muestreo. A medida que aumenta a_i se incrementa la probabilidad de que la muestra sea una representación válida del producto del trabajo. Sin embargo, los recursos requeridos para hacer el muestreo también aumentan. El equipo de ingeniería de software debe establecer el mejor valor de a_i para los tipos de productos generados.[3]

3 Thelin *et al.*, realizaron una simulación detallada que puede ayudar a hacer esto. Consulte [The01] para mayores detalles.

CasaSegura

Aspectos de la calidad

La escena: La oficina de Doug Miller, al comenzar el proyecto de software *CasaSegura*.

Participantes: Doug Miller (gerente del equipo de ingeniería de software de *CasaSegura*) y otros miembros del equipo.

La conversación:

Doug: Sé que no hemos dedicado tiempo a desarrollar un plan de calidad para este proyecto, pero ya estamos en él y tenemos que tomar en cuenta la calidad… ¿Correcto?

Jamie: Seguro. Decidimos que conforme desarrollemos el modelo de requerimientos [capítulos 6 y 7], Ed hará un procedimiento para probar cada uno de ellos.

Doug: Eso es realmente bueno, pero no vamos a esperar a las pruebas para evaluar la calidad, ¿verdad?

Vinod: No, por supuesto que no… Hemos programado revisiones en el plan del proyecto para este incremento de software. Comenzaremos el control de calidad con las revisiones.

Jamie: Me preocupa un poco que no tengamos tiempo suficiente para hacer todas las revisiones. En realidad, sé que no lo tendremos.

Doug: Mmm… ¿Qué proponen?

Jamie: Que seleccionemos aquellos elementos del modelo de requerimientos y diseño que tengan más importancia crítica para *CasaSegura* y que los revisemos.

Vinod: Pero, ¿qué pasa si hay algo mal en una parte que no hayamos revisado?

Shakira: Leí algo sobre una técnica de muestreo [sección 15.6.4] que podría ayudarnos a determinar candidatos a la revisión (Shakira explica este enfoque.)

Jamie: Quizá… pero no estoy seguro de que tengamos tiempo incluso para tomar muestras de cada elemento de los modelos.

Vinod: Doug, ¿qué quieres que hagamos?

Doug: Tomemos algo de la programación extrema [véase el capítulo 3]. Desarrollaremos los elementos de cada modelo en parejas —dos personas— y haremos una revisión informal de cada una conforme avancemos. Entonces nos abocaremos a los elementos "críticos" para hacer una revisión más formal en equipo, pero hay que mantener esas revisiones al mínimo. De ese modo, todo será observado por más de un par de ojos y también cumpliremos nuestras fechas de entrega.

Jamie: Eso significa que vamos a tener que revisar la programación de actividades.

Doug: Así es. La calidad altera la programación de este proyecto.

15.7 Resumen

El objetivo de toda revisión técnica es detectar errores y descubrir aspectos que tendrían un efecto negativo en el software que se va a desarrollar. Entre más pronto se descubra y corrija un error, menos probable es que se propague a otros productos del trabajo de la ingeniería de software y que se amplifique, lo que provocaría un mayor esfuerzo para corregirlo.

A fin de determinar si las actividades de control de calidad funcionan, deben determinarse varias métricas. Éstas se centran en el esfuerzo requerido para realizar la revisión y los tipos y severidad de errores descubiertos durante la revisión. Una vez recabadas las métricas, se usan para evaluar la eficacia de las revisiones que se efectúen. Los datos de la industria indican que las revisiones tienen un rendimiento elevado sobre la inversión.

Un modelo de referencia para la formalidad de la revisión identifica roles de las personas, planeación y preparación, estructura de la reunión, enfoque de corrección y verificación como las características que indican el grado de formalidad con el que se realiza una revisión. Las revisiones informales son de naturaleza casual, pero pueden usarse con eficacia para detectar errores. Las revisiones formales son más estructuradas y tienen una probabilidad mayor de dar como resultado un software de alta calidad.

Las revisiones informales se caracterizan por tener una planeación y preparación mínimas y poco registro de su desarrollo. Las verificaciones de escritorio y la programación por parejas forman parte de esta categoría de revisión.

Una revisión técnica formal es una reunión estilizada que ha demostrado ser extremadamente eficaz para detectar errores. Los walkthrougs y las inspecciones establecen roles definidos para cada revisor, estimulan la planeación y la preparación previa, requieren la aplicación de lineamientos de revisión definidos y ordenan llevar registros y hacer reportes. Las revisiones

por muestreo se utilizan cuando no es posible efectuar revisiones técnicas formales para todos los productos del trabajo.

Problemas y puntos por evaluar

15.1. Explique la diferencia entre un *error* y un *defecto*.

15.2. ¿Por qué no puede esperarse a las pruebas para encontrar y corregir todos los errores del software?

15.3. Suponga que en el modelo de requerimientos se han cometido 10 errores y que cada uno se amplificará en un factor de 2:1 en el diseño, y que se cometerán otros 20 errores de diseño adicionales que luego se amplificarán en un factor de 1.5:1 en el código, donde se cometerán otros 30 errores adicionales. Suponga que todas las pruebas unitarias encontrarán 30 por ciento de todos los errores, que la integración descubrirá 30 por ciento de los restantes y que las pruebas de validación hallarán 50 por ciento de los que queden. No se efectuarán revisiones. ¿Cuántos errores saldrán al público?

15.4. Vuelva a considerar la situación descrita en el problema 15.3, pero ahora suponga que se realizan revisiones en los requerimientos, diseño y código, con 60 por ciento de eficacia en el descubrimiento de todos los errores en esa etapa. ¿Cuántos errores saldrán al público?

15.5. Estudie de nuevo la situación descrita en los problemas 15.3 y 15.4. Si cada uno de los errores que salen al público tiene un costo de $4 800 por ser detectado y corregido, y hacer lo mismo para cada error descubierto en la revisión cuesta $240, ¿cuánto dinero se ahorra por efectuar revisiones?

15.6. En sus propias palabras, describa el significado de la figura 15.4.

15.7. ¿Cuál de las características del modelo de referencia piensa usted que tiene el mayor efecto en la formalidad de la revisión? Explique por qué.

15.8. ¿Se le ocurren algunos casos en los que una verificación de escritorio genere problemas en lugar de beneficios?

15.9. Una revisión técnica formal es eficaz sólo si cada quien se prepara por adelantado. ¿Cómo se reconoce a un participante que no se haya preparado? ¿Qué haría si usted fuera el líder de la revisión?

15.10. Al considerar todos los lineamientos para la revisión presentados en la sección 15.6.3, ¿cuál piensa que sea el más importante y por qué?

Lecturas y fuentes de información adicionales

Se han escrito relativamente pocos libros sobre las revisiones de software. Algunas de las ediciones recientes que dan una guía útil incluyen los textos de Wong (*Modern Software Review*, IRM Press, 2006), Radice (*High Quality, Low Cost Software Inspections*, Paradoxicon Publishers, 2002), Wiegers (*Peer Reviews in Software: A Practical Guide*, Addison-Wesley, 2001) y Gilb y Graham (*Software Inspection*, Addison-Wesley, 1993). El de Freedman y Weinberg (*Handbook of Walkthroughs, Inspections and Technical Reviews*, Dorset House, 1990) sigue siendo un texto clásico y todavía proporciona información útil acerca de este tema tan importante.

En internet existe una amplia variedad de fuentes de información acerca de la calidad del software. En el sitio web del libro, se encuentra una lista actualizada de referencias existentes en la red mundial y que son relevantes para las revisiones de software, en la dirección **www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm**.

CAPÍTULO

16 ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE

CONCEPTOS CLAVE

El enfoque de la ingeniería de software descrito en este libro se dirige a una sola meta: producir software a tiempo y de alta calidad. Pero muchos lectores se preguntarán: "¿Qué es calidad del software?".

Philip Crosby [Cro79], en su libro clásico sobre calidad, da una respuesta irónica a esta pregunta:

> El problema de la administración de la calidad no es lo que la gente ignora de ella. El problema es lo que piensan que saben...
>
> En ese sentido, la calidad tiene mucho en común con el sexo. Todo mundo lo busca (en ciertas condiciones, por supuesto). Todos creen que lo entienden (aunque no querrían explicarlo). Todos piensan que su ejecución sólo consiste en seguir las inclinaciones naturales (después de todo, lo hacemos de algún modo). Y, por supuesto, la mayoría de la gente siente que los problemas en esta área los causan las demás personas (si sólo se dieran el tiempo de hacer las cosas bien).

En realidad, la calidad es un concepto difícil (se abordó con cierto detalle en el capítulo 14).[1]

Algunos desarrolladores de software todavía creen que la calidad del software es algo por lo que hay que empezar a preocuparse una vez generado el código. Nada podría estar más lejos de la verdad... El *aseguramiento de la calidad del software* (con frecuencia llamado *administración*

UNA MIRADA RÁPIDA

¿Qué es? No basta hablar por hablar para decir que la calidad del software es importante. Tiene que 1) definirse explícitamente lo que quiere decir "calidad del software", 2) crearse un conjunto de actividades que ayuden a garantizar que todo producto de la ingeniería de software tenga alta calidad, 3) desarrollarse el control de calidad y las actividades para asegurar ésta en todo proyecto de software, 4) usarse métricas para desarrollar estrategias a fin de mejorar el proceso del software y, en consecuencia, la calidad del producto final.

¿Quién lo hace? Todos los involucrados en el proceso de ingeniería de software son los responsables de la calidad.

¿Por qué es importante? Las cosas pueden hacerse bien o pueden volverse a hacer. Si un equipo de software pone el énfasis en la calidad en todas las actividades de la ingeniería de software, se reduce la cantidad de repeticiones que debe hacer. Eso da como resultado costos más bajos y, lo que es más importante, un mejor tiempo para llegar al mercado.

¿Cuáles son las etapas? Antes de iniciar las actividades de aseguramiento de la calidad del software (ACS), es importante definir la *calidad del software* en varios niveles diferentes de abstracción. Una vez que se entiende lo que es la calidad, el equipo de software debe identificar un conjunto de actividades de ACS que filtren los errores de los productos del trabajo antes de que se aprueben.

¿Cuál es el producto final? Se crea un Plan de Aseguramiento de la Calidad del Software para definir una estrategia de ACS del equipo. Durante la modelación y codificación, el producto principal del ACS es la salida de las revisiones técnicas (véase el capítulo 15). Durante las pruebas (capítulos 17 a 20), se generan los planes y procedimientos de prueba, así como otros productos del trabajo asociados con el proceso de mejora.

¿Cómo me aseguro de que lo hice bien? Hay que encontrar los errores antes de que se vuelvan defectos... Es decir, debe trabajarse para mejorar la eficiencia en la eliminación de defectos (capítulo 23), a fin de reducir la cantidad de repeticiones que tenga que hacer el equipo del software.

1 Si no ha leído el capítulo 14, debe leerlo ahora.

de la calidad) es una actividad sombrilla (véase el capítulo 2) que se aplica en todo el proceso del software.

El aseguramiento de la calidad del software (ACS) incluye lo siguiente: 1) un proceso de ACS, 2) tareas específicas de aseguramiento y control de la calidad (incluidas revisiones técnicas y una estrategia de pruebas relacionadas entre sí), 3) prácticas eficaces de ingeniería de software (métodos y herramientas), 4) control de todos los productos del trabajo de software y de los cambios que sufren (véase el capítulo 22), 5) un procedimiento para garantizar el cumplimiento de los estándares del desarrollo de software (cuando sea aplicable) y 6) mecanismos de medición y reporte.

Este capítulo se centra en aspectos de la administración y en las actividades específicas del proceso que permiten a una organización de software garantizar que hace "las cosas correctas en el momento correcto y de la forma correcta".

16.1 Antecedentes

El control y aseguramiento de la calidad son actividades esenciales para cualquier negocio que genere productos que utilicen otras personas. Antes del siglo xx, el control de calidad era responsabilidad única del artesano que elaboraba el producto. Cuando pasó el tiempo y las técnicas de la producción en masa se hicieron comunes, el control de calidad se convirtió en una actividad ejecutada por personas diferentes de aquellas que elaboraban el producto.

Cita:

"Cometes demasiados errores equivocados."

Yogi Berra

La primera función formal de aseguramiento y control de la calidad se introdujo en los laboratorios Bell en 1916 y se difundió con rapidez al resto del mundo de la manufactura. Durante la década de 1940, sugirieron enfoques más formales del control de calidad. Éstos se basaban en la medición y en el proceso de la mejora continua [Dem86] como elementos clave de la administración de la calidad.

Actualmente, toda compañía tiene mecanismos para asegurar la calidad en sus productos. En realidad, en las últimas décadas, las afirmaciones explícitas del compromiso de una compañía con la calidad se han vuelto un mantra de la mercadotecnia.

La historia del aseguramiento de la calidad en el desarrollo del software corre de manera paralela con la historia de la calidad en la manufactura del hardware. En los primeros días de la computación (décadas de 1950 y 1960), la calidad era responsabilidad única del programador. Los estándares para asegurar la calidad del software se introdujeron en los contratos para desarrollar software militar en la década de 1970 y se extendieron con rapidez al desarrollo de software en el mundo comercial [IEE93]. Si se amplía la definición presentada al principio, el aseguramiento de la calidad del software es un "patrón planeado y sistemático de acciones" [Sch98c] que se requieren para garantizar alta calidad en el software. El alcance de la responsabilidad del aseguramiento de la calidad se caracteriza mejor si se parafrasea un comercial de un automóvil popular: "La calidad es el empleo número 1." La implicación para el software es que muchas entidades diferentes tienen responsabilidad en el aseguramiento de la calidad del software: ingenieros de software, gerentes de proyecto, clientes, vendedores y los individuos que trabajan en el grupo de ACS.

El grupo de ACS funciona como representante del cliente en el interior de la empresa. Es decir, la gente que realiza el ACS debe ver al software desde el punto de vista del cliente. ¿El software cumple adecuadamente los factores de calidad mencionados en el capítulo 14? ¿El desarrollo del software se condujo de acuerdo con estándares preestablecidos? ¿Las disciplinas técnicas han cumplido con sus roles como parte de la actividad de ACS? El grupo de ACS trata de responder éstas y otras preguntas para garantizar que se mantenga la calidad del software.

16.2 Elementos de aseguramiento de la calidad del software

WebRef

En la dirección **www.swqual.com/newsletter/vol2/no1/vol2no1.html**, se encuentra un análisis profundo del ACS, que incluye una amplia variedad de definiciones.

El aseguramiento de la calidad del software incluye un rango amplio de preocupaciones y actividades que se centran en la administración de la calidad del software. Éstas se resumen como sigue [Hor03]:

Estándares. El IEEE, ISO y otras organizaciones que establecen estándares han producido una amplia variedad de ellos para ingeniería de software y documentos relacionados. Los estándares los adopta de manera voluntaria una organización de software o los impone el cliente u otros participantes. El trabajo del ACS es asegurar que los estándares que se hayan adoptado se sigan, y que todos los productos del trabajo se apeguen a ellos.

Revisiones y auditorías. Las revisiones técnicas son una actividad del control de calidad que realizan ingenieros de software para otros ingenieros de software (véase el capítulo 15). Su objetivo es detectar errores. Las auditorías son un tipo de revisión efectuada por personal de ACS con objeto de garantizar que se sigan los lineamientos de calidad en el trabajo de la ingeniería de software. Por ejemplo, una auditoría del proceso de revisión se efectúa para asegurar que las revisiones se lleven a cabo de manera que tengan la máxima probabilidad de descubrir errores.

Pruebas. Las pruebas del software (capítulos 17 a 20) son una función del control de calidad que tiene un objetivo principal: detectar errores. El trabajo del ACS es garantizar que las pruebas se planeen en forma apropiada y que se realicen con eficiencia, de modo que la probabilidad de que logren su objetivo principal sea máxima.

Colección y análisis de los errores. La única manera de mejorar es medir cómo se está haciendo algo. El ACS reúne y analiza errores y datos acerca de los defectos para entender mejor cómo se cometen los errores y qué actividades de la ingeniería de software son más apropiadas para eliminarlos.

Administración del cambio. El cambio es uno de los aspectos que más irrumpe en cualquier proyecto de software. Si no se administra en forma adecuada, lleva a la confusión y ésta casi siempre genera mala calidad. El ACS asegura que se hayan instituido prácticas adecuadas de administración del cambio (véase el capítulo 22).

Educación. Toda organización de software quiere mejorar sus prácticas de ingeniería de software. Un contribuyente clave de la mejora es la educación de los ingenieros de software, de sus gerentes y de otros participantes. La organización de ACS lleva el liderazgo en la mejora del proceso de software (capítulo 30) y es clave para proponer y patrocinar programas educativos.

Cita:

"La excelencia es la capacidad ilimitada de mejorar la calidad de lo que se tenga para ofrecer."

Rick Petin

Administración de los proveedores. Son tres las categorías de software que se adquieren a proveedores externos: *paquetes contenidos en una caja* (por ejemplo, *Office*, de Microsoft); *un shell personalizado* [Hor03], que da una estructura básica, tipo esqueleto, que se adapta de manera única a las necesidades del comprador; y *software contratado*, que se diseña y construye especialmente a partir de especificaciones provistas por la organización cliente. El trabajo de la organización de ACS es garantizar que se obtenga software de alta calidad a partir de las sugerencias de prácticas específicas de calidad que el proveedor debe seguir (cuando sea posible) y de la incorporación de cláusulas de calidad como parte de cualquier contrato con un proveedor externo.

Administración de la seguridad. Con el aumento de los delitos cibernéticos y de las nuevas regulaciones gubernamentales respecto de la privacidad, toda organización de software debe instituir políticas para proteger los datos en todos los niveles, establecer cortafuegos de protección para las *webapps* y asegurar que el software no va a ser vulnerado in-

ternamente. El ACS garantiza que para lograr la seguridad del software, se utilicen el proceso y la tecnología apropiados.

Seguridad. Debido a que el software casi siempre es un componente crucial de los sistemas humanos (como aplicaciones automotrices o aeronáuticas), la consecuencia de defectos ocultos puede ser catastrófica. El ACS es responsable de evaluar el efecto de las fallas del software y de dar los pasos que se requieren para disminuir el riesgo.

Administración de riesgos. Aunque el análisis y la mitigación de riesgos (véase el capítulo 28) es asunto de los ingenieros de software, la organización del ACS garantiza que las actividades de administración de riesgos se efectúen en forma apropiada y que se establezcan planes de contingencia relacionados con los riesgos.

Además de cada una de estas preocupaciones y actividades, el ACS tiene como preocupación dominante asegurar que las actividades de apoyo del software (como mantenimiento, líneas de ayuda, documentación y manuales) se lleven a cabo o se produzcan con calidad.

INFORMACIÓN

Recursos para la administración de la calidad

En la red mundial existen decenas de recursos para la administración de la calidad, incluidas sociedades profesionales, organizaciones emisoras de estándares y fuentes de información general. Los sitios siguientes constituyen un buen punto de partida:

American Society for Quality (ASQ), División de Software, **www.asq.org/software**
Association for Computer Machinery, **www.acm.org**, Centro de Datos y Análisis del Software (DACS), **www.dacs.dtic.mil/**
International Organization for Standardization (ISO), **www.iso.ch**
ISO SPIECE, **www.isospiece.com**
Malcolm Baldridge National Quality Award, **www.quality.nist.gov**
Software Engineering Institute, **www.sei.cmu.edu/**
Software Testing and Quality Engineering, **www.stickyminds.com**
Six Sigma Resources, **www.isixsigma.com/** **www.asq.org/sixsigma/**
TickIT International, temas sobre certificación de la calidad, **www.tickit.org/international.htm**
Total Quality Management (TQM)
Información general: **www.gslis.utexas.edu/~rpollock/tqm.html**
Artículos: **www.work911.com/tqmarticles.htm**
Glosario: **www.quality.org/TQM-MSI/TQM-glossary.html**

16.3 TAREAS, METAS Y MÉTRICAS DEL ACS

El aseguramiento de la calidad del software se compone de varias tareas asociadas con dos entidades diferentes: los ingenieros de software que hacen el trabajo técnico y un grupo de ACS que tiene la responsabilidad de planear, supervisar, registrar, analizar y hacer reportes acerca de la calidad.

Los ingenieros de software abordan la calidad (y ejecutan actividades para controlarla), aplicando métodos y medidas técnicas sólidos, realizando revisiones técnicas y haciendo pruebas de software bien planeadas.

16.3.1 Tareas del ACS

El objetivo del grupo de ACS es auxiliar al equipo del software para lograr un producto final de alta calidad. El Instituto de Ingeniería de Software recomienda un conjunto de acciones de ACS que se dirigen a la planeación, supervisión, registro, análisis y elaboración de reportes para el aseguramiento de la calidad. Estas acciones son realizadas (o facilitadas) por un grupo independiente de ACS que hace lo siguiente:

¿Cuál es el rol de un grupo de ACS?

Prepara el plan de ACS para un proyecto. El plan se desarrolla como parte de la preparación del proyecto y es revisado por todos los participantes. Las acciones de aseguramiento de la calidad efectuadas por el equipo de ingeniería de software y por el grupo de ACS son dirigidas por el plan. Éste identifica las evaluaciones que se van a realizar, las auditorías y revisiones por efectuar, los estándares aplicables al proyecto, los procedimientos para reportar y dar seguimiento a los errores, los productos del trabajo que genera el grupo de ACS y la retroalimentación que se dará al equipo del software.

Participa en el desarrollo de la descripción del software del proyecto. El equipo de software selecciona un proceso para el trabajo que se va a realizar. El grupo de ACS revisa la descripción del proceso a fin de cumplir con la política organizacional, los estándares internos para el software, los estándares impuestos desde el exterior (como la norma ISO-9001) y otras partes del plan del proyecto de software.

Revisa las actividades de la ingeniería de software a fin de verificar el cumplimiento mediante el proceso definido para el software. El grupo de ACS identifica, documenta y da seguimiento a las desviaciones del proceso y verifica que se hayan hecho las correcciones pertinentes.

Audita los productos del trabajo de software designados para verificar que se cumpla con aquellos definidos como parte del proceso de software. El grupo de ACS revisa productos del trabajo seleccionados; identifica, documenta y da seguimiento a las desviaciones; verifica que se hayan hecho las correcciones necesarias y reporta periódicamente los resultados de su trabajo al gerente del proyecto.

Asegura que las desviaciones en el trabajo de software y sus productos se documenten y manejen de acuerdo con un procedimiento documentado. Las desviaciones pueden encontrarse en el plan del proyecto, la descripción del proceso, los estándares aplicables o los productos del trabajo de la ingeniería de software.

Registra toda falta de cumplimiento y la reporta a la alta dirección. Se da seguimiento a los incumplimientos hasta que son resueltos.

Además de estas acciones, el grupo de ACS coordina el control y administración del cambio (véase el capítulo 22) y ayuda a recabar y analizar métricas para el software.

16.3.2 Metas, atributos y métricas

Cita:

"La calidad nunca es un accidente; siempre es el resultado de una intención clara, un esfuerzo sincero, una dirección inteligente y una ejecución hábil; representa la elección sabia de muchas alternativas".

William A. Foster

Las acciones de ACS descritas en la sección anterior se realizan con objeto de alcanzar un conjunto de metas pragmáticas:

Calidad de los requerimientos. La corrección, completitud y consistencia del modelo de requerimientos tendrá una gran influencia en la calidad de todos los productos del trabajo que sigan. El ACS debe garantizar que el equipo de software ha revisado en forma apropiada el modelo de requerimientos a fin de alcanzar un alto nivel de calidad.

Calidad del diseño. Todo elemento del modelo del diseño debe ser evaluado por el equipo del software para asegurar que tenga alta calidad y que el diseño en sí se apegue a los requerimientos. El ACS busca atributos del diseño que sean indicadores de la calidad.

Calidad del código. El código fuente y los productos del trabajo relacionados (por ejemplo, otra información descriptiva) deben apegarse a los estándares locales de codificación y tener características que faciliten darle mantenimiento. El ACS debe identificar aquellos atributos que permitan hacer un análisis razonable de la calidad del código.

Eficacia del control de calidad. Un equipo de software debe aplicar recursos limitados, en forma tal que tenga la máxima probabilidad de lograr un resultado de alta calidad. El

FIGURA 16.1 **Metas atributos y métricas de la calidad del software**
Fuente: Adaptado de [Hya96].

Meta	Atributo	Métrica
Calidad de los requerimientos	Ambigüedad	Número de modificadores ambiguos (por ejemplo, muchos, grande, amigable, etc.)
	Completitud	Número de TBA y TBD
	Comprensibilidad	Número de secciones y subsecciones
	Volatilidad	Número de cambios por requerimiento
		Tiempo (por actividad) cuando se solicita un cambio
	Trazabilidad	Número de requerimientos no trazables hasta el diseño o código
	Claridad del modelo	Número de modelos UML
		Número de páginas descriptivas por modelo
		Número de errores de UML
Calidad del diseño	Integridad arquitectónica	Existencia del modelo arquitectónico
	Completitud de componentes	Número de componentes que se siguen hasta el modelo arquitectónico
		Complejidad del diseño del procedimiento
	Complejidad de la interfaz	Número promedio de pasos para llegar a una función o contenido normal
		Distribución apropiada
	Patrones	Número de patrones utilizados
Calidad del código	Complejidad	Complejidad ciclomática
	Facilidad de mantenimiento	Factores de diseño (capítulo 8)
	Comprensibilidad	Porcentaje de comentarios internos
		Convenciones variables de nomenclatura
	Reusabilidad	Porcentaje de componentes reutilizados
	Documentación	Índice de legibilidad
Eficacia del control de calidad	Asignación de recursos	Porcentaje de personal por hora y por actividad
	Tasa de finalización	Tiempo de terminación real *versus* lo planeado
	Eficacia de la revisión	Ver medición de la revisión (capítulo 14)
	Eficacia de las pruebas	Número de errores de importancia crítica encontrados
		Esfuerzo requerido para corregir un error
		Origen del error

ACS analiza la asignación de recursos para las revisiones y pruebas a fin de evaluar si se asignan en la forma más eficaz.

La figura 16.1 (adaptada de [Hya96]) identifica los atributos que son indicadores de la existencia de la calidad para cada una de las metas mencionadas. También se presentan las métricas que se utilizan para indicar la fortaleza relativa de un atributo.

16.4 ENFOQUES FORMALES AL ACS

En las secciones anteriores, se dijo que la calidad del software es el trabajo de cada quien y que puede lograrse por medio de una práctica competente de la ingeniería de software, así como de la aplicación de revisiones técnicas, de una estrategia de pruebas con relaciones múltiples, de un mejor control de los productos del trabajo de software y de los cambios efectuados sobre

WebRef

En la dirección **www.gslis.utexas.edu/~rpollock/tqm.html**, se encuentra información útil acerca del ACS y de los métodos formales de la calidad.

ellos, así como de la aplicación de estándares aceptados de la ingeniería de software. Además, la calidad se define en términos de una amplia variedad de atributos de la calidad y se mide (indirectamente) con el empleo de varios índices y métricas.

En las últimas tres décadas, un segmento pequeño pero sonoro de la comunidad de la ingeniería de software ha afirmado que se requiere un enfoque más formal para el ACS. Puede decirse que un programa de cómputo es un objeto matemático. Para cada lenguaje de programación, es posible definir una sintaxis y semántica rigurosas, y se dispone de un enfoque igualmente riguroso para la especificación de los requerimientos del software (véase el capítulo 21). Si el modelo de los requerimientos (especificación) y el lenguaje de programación se representan en forma rigurosa, debe ser posible usar una demostración matemática para la corrección, de modo que se confirme que un programa se ajusta exactamente a sus especificaciones.

Los intentos de demostrar la corrección de un programa no son nuevos. Dijkstra [Dij76a] y Linger, Mills y Witt [Lin79], entre otros, han invocado pruebas de la corrección de programas y las han relacionado con el uso de conceptos de programación estructurada (véase el capítulo 10).

16.5 Aseguramiento estadístico de la calidad del software

El aseguramiento estadístico de la calidad del software refleja una tendencia creciente en la industria para que se vuelva más cuantitativo respecto de la calidad. Para el software, el aseguramiento estadístico de la calidad implica los pasos siguientes:

? ¿Qué pasos se requieren para efectuar el ACS estadístico?

1. Se recaba y clasifica la información acerca de errores y defectos del software.
2. Se hace un intento por rastrear cada error y defecto hasta sus primeras causas (por ejemplo, no conformidad con las especificaciones, error de diseño, violación de los estándares, mala comunicación con el cliente, etc.).
3. Con el uso del Principio de Pareto (80 por ciento de los defectos se debe a 20 por ciento de todas las causas posibles), se identifica 20 por ciento de las causas de errores y defectos (las *pocas vitales*).
4. Una vez identificadas las pocas causas vitales, se corrigen los problemas que han dado origen a los errores y defectos.

Este concepto relativamente simple representa un paso importante hacia la creación de un proceso adaptativo del software en el que se hacen cambios para mejorar aquellos elementos del proceso que introducen errores.

16.5.1 Ejemplo general

Cita:

"Un análisis estadístico, si se realiza en forma apropiada, es una disección delicada de las incertidumbres, una cirugía de las suposiciones."

M. J. Moroney

A fin de ilustrar el uso de los métodos estadísticos para el trabajo de ingeniería de software, suponga que una organización de ingeniería de software recaba información sobre los errores y defectos cometidos en un periodo de un año. Algunos de dichos errores se descubren a medida que se desarrolla el software. Otros (defectos) se encuentran después de haber liberado el software a sus usuarios finales. Aunque se descubren cientos de problemas diferentes, todos pueden rastrearse hasta una (o más) de las causas siguientes:

- Especificaciones erróneas o incompletas (EEI)
- Mala interpretación de la comunicación con el cliente (MCC)
- Desviación intencional de las especificaciones (DIE)
- Violación de los estándares de programación (VEP)
- Error en la representación de los datos (ERD)

FIGURA 16.2
Colección de datos para hacer ACS estadístico

	Total		Serio		Moderado		Menor	
Error	**No.**	**%**	**No.**	**%**	**No.**	**%**	**No.**	**%**
IES	205	22%	34	27%	68	18%	103	24%
MCC	156	17%	12	9%	68	18%	76	17%
IDS	48	5%	1	1%	24	6%	23	5%
VPS	25	3%	0	0%	15	4%	10	2%
EDR	130	14%	26	20%	68	18%	36	8%
ICI	58	6%	9	7%	18	5%	31	7%
EDL	45	5%	14	11%	12	3%	19	4%
IET	95	10%	12	9%	35	9%	48	11%
IID	36	4%	2	2%	20	5%	14	3%
PIT	60	6%	15	12%	19	5%	26	6%
HCI	28	3%	3	2%	17	4%	8	2%
MIS	56	6%	0	0%	15	4%	41	9%
Totales	942	100%	128	100%	379	100%	435	100%

- Interfaz componente inconsistente (ICI)
- Error en el diseño lógico (EDL)
- Pruebas incompletas o erróneas (PIE)
- Documentación inexacta o incompleta (DII)
- Error en la traducción del lenguaje de programación del diseño (LPD)
- Interfaz humano/computadora ambigua o inconsistente (IHC)
- Varios (V)

Cita:

"20 por ciento del código tiene 80 por cierto de los errores. Encuéntrelos, corríjalos".

Lowell Arthur

Para aplicar el ACS estadístico, se elabora la tabla de la figura 16.2. La tabla indica que EEI, MCC y ERD son las pocas causas vitales que originan 53 por ciento de todos los errores. Sin embargo, debe notarse que EEI, ERD, LPD y EDL se habrían seleccionado como las pocas causas vitales si se consideran sólo errores serios. Una vez que las pocas causas vitales han sido determinadas, la organización de ingeniería de software comienza su acción correctiva. Por ejemplo, a fin de corregir el MCC, deben implementarse técnicas para recabar requerimientos (capítulo 5) que mejoren la calidad de la comunicación y las especificaciones con el cliente. Para mejorar el ERD, deben adquirirse herramientas para desarrollar la modelación de casos y realizar datos y revisiones del diseño más significativos.

Es importante notar que la acción correctiva se centra sobre todo en las pocas causas vitales. En tanto éstas se corrigen, nuevas candidatas se van a la cumbre de la pila.

Las técnicas para el aseguramiento correctivo han sido propuestas para dar una mejora sustancial de la calidad [Art97]. En ciertos casos, las organizaciones de software han tenido una reducción anual de 50 por ciento en defectos después de aplicar esta técnica.

La aplicación del ACS estadístico y el Principio de Pareto se resumen en una sola oración: *Pasa tu tiempo viendo las cosas que realmente importan, pero primero asegúrate de que entiendes lo que realmente importa...*

16.5.2 Seis Sigma para la ingeniería de software

Seis Sigma es la estrategia más ampliamente usada hoy para el aseguramiento estadístico de la calidad en la industria. La estrategia Seis Sigma fue popularizada originalmente por Motorola en la década de 1980 y "es una metodología rigurosa y disciplinada que usa datos y análisis estadísticos para medir y mejorar el desempeño operativo de una compañía, identificando y eliminando defectos en procesos de manufactura y servicios" [ISI08]. El término Seis Sigma se deriva

de seis desviaciones estándar —3.4 casos (defectos) por millón de ocurrencias—, lo que implica un estándar de calidad extremadamente alto. La metodología Seis Sigma define tres etapas fundamentales:

? ¿Cuáles son las etapas fundamentales de la metodología Seis Sigma?

- *Definir* los requerimientos del cliente y los que se le entregan, así como las metas del proyecto a través de métodos bien definidos de comunicación con el cliente.
- *Medir* el proceso existente y su resultado para determinar el desempeño actual de la calidad (recabar métricas para los defectos).
- *Analizar* las métricas de los defectos y determinar las pocas causas vitales.

Si se trata de un proceso de software existente que se requiere mejorar, Seis Sigma sugiere dos etapas adicionales:

- *Mejorar* el proceso, eliminando las causas originales de los defectos.
- *Controlar* el proceso para asegurar que el trabajo futuro no vuelva a introducir las causas de los defectos.

Estas etapas fundamentales y adicionales en ocasiones son conocidas como método DMAMC (definir, medir, analizar, mejorar y controlar).

Si una organización va a desarrollar un proceso de software (en vez de mejorar uno existente), a las etapas fundamentales se agregan las siguientes:

- *Diseñar* el proceso para 1) evitar las causas originales de los defectos y 2) cumplir los requerimientos del cliente.
- *Verificar* que el modelo del proceso en realidad evite los defectos y cumpla los requerimientos del cliente.

Esta variación en ocasiones es denominada método DMADV (definir, medir, analizar, diseñar y verificar).

El estudio detallado de Seis Sigma se deja a fuentes dedicadas a ese tema. Si el lector tiene interés al respecto, consulte [ISI08], [Pyz303] y [Sne03].

16.6 Confiabilidad del software

Cita:

"El precio inevitable de la confiabilidad es la simplicidad."

C. A. R. Hoare

No hay duda de que la confiabilidad de un programa de cómputo es un elemento importante de su calidad general. Si un programa falla repetida y frecuentemente en su desempeño, importa poco si otros factores de la calidad del software son aceptables.

La confiabilidad del software, a diferencia de muchos otros factores de la calidad, se mide y estima directamente mediante el uso de datos históricos del desarrollo. La *confiabilidad del software* se define en términos estadísticos como "la probabilidad que tiene un programa de cómputo de operar sin fallas en un ambiente específico por un tiempo específico" [Mus87]. Para ilustrar lo anterior, digamos que se estima que el programa *X* tiene una confiabilidad de 0.999 durante ocho horas de procesamiento continuo. En otras palabras, si el programa *X* fuera a ejecutarse 1 000 veces y requiriera un total de ocho horas de tiempo de procesamiento continuo (tiempo de procesamiento), es probable que operara correctamente (sin fallas) 999 veces.

Siempre que se trate de la confiabilidad del software, surge una pregunta crucial: ¿qué significa el término *falla*? En el contexto de cualquier análisis de la calidad y confiabilidad del software, la falla significa la falta de conformidad con los requerimientos del software. Pero, incluso con esta definición, hay gradaciones. Las fallas pueden ser leves o catastróficas. Una falla podría corregirse en segundos, mientras que otra tal vez requiera de varias semanas o meses de trabajo para ser corregida. Para complicar más el asunto, la corrección de una falla quizá dé como resultado la introducción de otros errores que a su vez originen otras fallas.

16.6.1 Mediciones de la confiabilidad y disponibilidad

PUNTO CLAVE

Los problemas de confiabilidad del software casi siempre pueden seguirse hasta encontrar defectos en el diseño o en la implementación.

Los primeros trabajos sobre confiabilidad del software trataban de extrapolar la teoría matemática de la confiabilidad del hardware a la predicción de la confiabilidad del software. La mayor parte de modelos relacionados con el hardware se abocan a la falla debida al uso, en lugar de a la que tiene su origen en los defectos de diseño. En el hardware, las fallas debidas al uso físico (por ejemplo, los efectos de temperatura, corrosión y golpes) son más probables que las debidas al diseño. Desafortunadamente, con el software ocurre lo contrario. En realidad, todas las fallas del software pueden rastrearse en problemas de diseño o de implementación; el uso (véase el capítulo 1) no entra en el escenario.

Ha habido un debate permanente acerca de la relación que existe entre los conceptos clave en la confiabilidad del hardware y su aplicabilidad al software. Aunque es posible establecer un vínculo irrefutable, es útil considerar algunos conceptos sencillos que se aplican a ambos elementos del sistema.

PUNTO CLAVE

Es importante observar que el tiempo medio entre fallas y otras medidas relacionadas se basa en tiempo del CPU, no en tiempo de reloj.

Si se considera un sistema basado en computadora, una medida sencilla de su confiabilidad es el *tiempo medio entre fallas* (TMEF):

$$\text{TMEF} = \text{TMPF} + \text{TMPR}$$

donde las siglas TMPF y TMPR significan *tiempo medio para la falla* y *tiempo medio para la reparación*,[2] respectivamente.

Muchos investigadores afirman que el TMEF es una medición más útil que otras relacionadas con la calidad del software que se estudian en el capítulo 23. En pocas palabras, a un usuario final le preocupan las fallas, no la cuenta total de defectos. Como cada defecto contenido en un programa no tiene la misma tasa de fallas, la cuenta total de defectos indica muy poco acerca de la confiabilidad del sistema. Por ejemplo, considere un programa que haya estado en operación durante 3 000 horas de procesador sin falla. Muchos defectos de este programa estarían sin detectar durante decenas de miles de horas antes de ser descubiertos. El TMEF de tales errores oscuros podría ser de 30 000 o hasta 60 000 horas de procesador. Otros defectos, no descubiertos, podrían tener una tasa de fallas de 4 000 a 5 000 horas. Aun si cada uno de los errores en esta categoría (los que tienen un TMEF largo) se eliminara, el efecto que tendrían sobre el software sería despreciable.

CONSEJO

Algunos aspectos de la disponibilidad (que no se estudian aquí) no tienen que ver con las fallas. Por ejemplo, la programación del tiempo fuera de operación (para funciones de apoyo) hace que el software no esté disponible.

Sin embargo, el TMEF puede ser problemático por dos razones: 1) proyecta un tiempo entre fallas, pero no da una tasa de fallas proyectada y 2) puede interpretarse mal, como la vida promedio, cuando *no* es esto lo que implica.

Una medición alternativa de confiabilidad es la de las *fallas en el tiempo* (FET): medición estadística de cuántas fallas tendrá un componente en mil millones de horas de operación. Por tanto, 1 FET es equivalente a una falla en cada mil millones de horas de operación.

Además de una medida de la confiabilidad, también debe desarrollarse otra para la disponibilidad. La *disponibilidad del software* es la probabilidad de que un programa opere de acuerdo con los requerimientos en un momento determinado de tiempo, y se define así:

$$\text{Disponibilidad} = \frac{\text{TMPF}}{\text{TMPF} + \text{TMPR}} \times 100\%$$

La medición del TMEF para la confiabilidad es igualmente sensible al TMPF y al TMPR. La medición de la disponibilidad es un poco más sensible al TMPR, que es una medición indirecta de la facilidad que tiene el software para recibir mantenimiento.

2 Aunque tal vez se requiera depurar (y hacer otras correcciones relacionadas) como consecuencia de la falla, en muchos casos el software funcionará de manera apropiada después de reiniciar, sin ningún otro cambio.

16.6.2 Seguridad del software

Cita:

"La seguridad de las personas debe ser la ley máxima."

Cicerón

La *seguridad del software* es una actividad del aseguramiento del software que se centra en la identificación y evaluación de los peligros potenciales que podrían afectarlo negativamente y que podrían ocasionar que falle todo el sistema. Si los peligros se identifican al principio del proceso del software, las características de su diseño se especifican de modo que los eliminen o controlen.

Como parte de la seguridad del software, se lleva a cabo un proceso de modelado y análisis. Inicialmente se identifican los peligros y se clasifican según su riesgo. Por ejemplo, algunos de los peligros asociados con un control de crucero basado en computadora para un automóvil podrían ser los siguientes: 1) ocasionar una aceleración incontrolada que no pudiera detenerse, 2) no responder a la presión en el pedal de frenado (porque se apague), 3) no encender cuando se active el interruptor y 4) perder o ganar velocidad poco a poco. Una vez identificados estos peligros en el nivel del sistema, se utilizan técnicas de análisis para asignar severidad y probabilidad de ocurrencia a cada uno.[3] Para ser eficaz, el software debe analizarse en el contexto de todo el sistema. Por ejemplo, un error sutil en la entrada de un usuario (las personas son componentes del sistema) podría ampliarse por una falla del software y producir datos de control que situaran equivocadamente un dispositivo mecánico. Si y sólo si se encontrara un único conjunto de condiciones ambientales externas, la posición falsa del dispositivo mecánico ocasionaría una falla desastrosa. Podrían usarse técnicas de análisis [Eri05], tales como árbol de fallas, lógica en tiempo real y modelos de red de Petri, para predecir la cadena de eventos que ocasionarían los peligros, así como la probabilidad de ocurrir que tendría cada uno de los eventos para generar la cadena.

Cita:

"No puedo imaginar ninguna condición que hiciera que esta nave se hundiera. La construcción naval moderna ha llegado más allá de eso".

E. I. Smith, capitán del *Titanic*

Una vez identificados y analizados los peligros, pueden especificarse requerimientos relacionados con la seguridad para el software. Es decir, la especificación contendría una lista de eventos indeseables y las respuestas deseadas del sistema ante ellos. Después se indicaría el papel del software en la administración indeseable de los mismos.

Aunque la confiabilidad y la seguridad del software están muy relacionadas, es importante entender la sutil diferencia entre ellas. La primera utiliza técnicas de análisis estadístico para determinar la probabilidad de que ocurra una falla del software. Sin embargo, la ocurrencia de una falla no necesariamente da como resultado un peligro o riesgo. La seguridad del software examina las formas en las que las fallas generan condiciones que llevan a un peligro. Es decir, las fallas no se consideran en el vacío, sino que se evalúan en el contexto de la totalidad del sistema basado en computadora y de su ambiente.

WebRef

En la dirección **www.safeware-eng.com/**, se encuentran varios artículos sobre seguridad del software.

El estudio exhaustivo de la seguridad del software está más allá del alcance de este libro. Si el lector está interesado en la seguridad del software y en otros aspectos relacionados, consulte [Smi05], [Dun02] y [Lev95].

16.7 Las normas de calidad ISO 9000[4]

Un *sistema de aseguramiento de la calidad* se define como la estructura organizacional, responsabilidades, procedimientos, procesos y recursos necesarios para implementar la administración de la calidad [ANS87]. Los sistemas de aseguramiento de la calidad se crean para ayudar a las organizaciones a asegurar que sus productos y servicios satisfagan las expectativas del con-

3 Este enfoque es similar a los métodos de análisis del riesgo descritos en el capítulo 28. La diferencia principal es el énfasis que se pone en aspectos de la tecnología en lugar de en los relacionados con el proyecto.

4 Esta sección, escrita por Michael Stovski, ha sido adaptada a partir de "Fundamentos de ISO 9000", libro de trabajo desarrollado para *Essential Software Engineering*, video desarrollado por R. S. Pressman & Associates, Inc. Se reimprime con su autorización.

sumidor gracias a que cumplan con sus especificaciones. Estos sistemas cubren una amplia variedad de actividades, que contemplan todo el ciclo de vida del producto, incluidos planeación, control, medición, pruebas e informes, así como la mejora de los niveles de calidad en todo el proceso de desarrollo y manufactura. La norma ISO 9000 describe en términos generales los elementos de aseguramiento de la calidad que se aplican a cualquier negocio, sin importar los productos o servicios ofrecidos.

Para registrarse en alguno de los modelos del sistema de aseguramiento de la calidad contenidos en la ISO 9000, por medio de auditores externos se revisan en detalle el sistema y las operaciones de calidad de una compañía, respecto del cumplimiento del estándar y de la operación eficaz. Después de un registro exitoso, el grupo de registro representado por los auditores emite un certificado para la compañía. Auditorías semestrales de supervisión aseguran el cumplimiento continuo de la norma.

WebRef

En la dirección **www.tantara.ab.ca/info.htm**, se encuentran muchos vínculos hacia los recursos de la norma ISO 9000/9001.

Los requerimientos esbozados por la norma ISO 9001:2000 se dirigen a temas tales como responsabilidad de la administración, sistema de calidad, revisión del contrato, control del diseño, documentación y control de datos, identificación del producto y su seguimiento, control del proceso, inspección y pruebas, acciones correctivas y preventivas, registros del control de calidad, auditorías internas de calidad, capacitación, servicio y técnicas estadísticas. A fin de que una organización de software se registre en la ISO 9001:2000, debe establecer políticas y procedimientos que cumplan cada uno de los requerimientos mencionados (y otros más), y después demostrar que sigue dichas políticas y procedimientos. Si el lector desea más información sobre la norma ISO 9001:2000, consulte [Ant06], [Mut03] o [Dob04].

INFORMACIÓN

La norma ISO 9001:2000

La descripción siguiente define los elementos básicos de la norma ISO 9001:2000. Información completa sobre la misma se obtiene en la Organización Internacional de Normas (**www.iso.ch**) y en otras fuentes de internet (como en **www.praxiom.com**).

Establecer los elementos de un sistema de administración de la calidad.
- Desarrollar, implementar y mejorar el sistema.
- Definir una política que ponga el énfasis en la importancia del sistema.

Documentar el sistema de calidad.
- Describir el proceso.
- Producir un manual de operación.
- Desarrollar métodos para controlar (actualizar) documentos.
- Establecer métodos de registro.

Apoyar el control y aseguramiento de la calidad.
- Promover la importancia de la calidad entre todos los participantes.
- Centrarse en la satisfacción del cliente.
- Definir un plan de calidad que se aboque a los objetivos, responsabilidades y autoridad.
- Definir mecanismos de comunicación entre los participantes.

Establecer mecanismos de revisión para el sistema de administración de la calidad.
- Identificar métodos de revisión y mecanismos de retroalimentación.
- Definir procedimientos para dar seguimiento.

Identificar recursos para la calidad, incluidos personal, capacitación y elementos de la infraestructura.
- Establecer mecanismos de control.
- Para la planeación
- Para los requerimientos del cliente
- Para las actividades técnicas (tales como análisis, diseño y pruebas)
- Para la vigilancia y administración del proyecto

Definir métodos de corrección.
- Evaluar datos y métricas de la calidad.
- Definir el enfoque para la mejora continua del proceso y la calidad.

16.8 EL PLAN DE ACS

El *Plan de ACS* proporciona un mapa de ruta para instituir el aseguramiento de la calidad del software. Desarrollado por el grupo de ACS (o por el equipo del software si no existe un grupo de ACS), el plan funciona como plantilla para las actividades de ACS que se instituyen para cada proyecto de software.

La IEEE [IEEE93] ha publicado una norma para el ACS. Ésta recomienda una estructura que identifica lo siguiente: 1) propósito y alcance del plan, 2) descripción de todos los productos del trabajo de ingeniería de software (tales como modelos, documentos, código fuente, etc.) que se ubiquen dentro del ámbito del ACS, 3) todas las normas y prácticas aplicables que se utilicen durante el proceso del software, 4) acciones y tareas del ACS (incluidas revisiones y auditorías) y su ubicación en el proceso del software, 5) herramientas y métodos que den apoyo a las acciones y tareas de ACS, 6) procedimientos para la administración de la configuración del software (véase el capítulo 22), 7) métodos para unificar las salvaguardas y para mantener todos los registros relacionados con el ACS y 8) roles y responsabilidades relacionados con la calidad del producto.

HERRAMIENTAS DE SOFTWARE

Administración de la calidad del software

Objetivos: El objetivo de las herramientas del ACS es ayudar al equipo del proyecto a evaluar y mejorar la calidad del producto del trabajo de software.

Mecánica: La mecánica de las herramientas varía. En general, el objetivo consiste en evaluar la calidad de un producto específico. Nota: Es frecuente que dentro de la categoría de herramientas para el ACS, se incluya una amplia variedad de herramientas para someter a prueba al software (véanse los capítulos 17 a 20).

Herramientas representativas[5]

ARM, desarrollada por la NASA (**state.gsfc.nasa.gov/tools/index.html**), proporciona mediciones que se utilizan para evaluar la calidad de un documento de requerimientos de software.

QPR ProcessGuide and Scorecard, desarrollada por QPR Software (**www.qpronline.com**), da apoyo para establecer Seis Sigma y otros enfoques de administración de la calidad.

Quality Tools and Templates, desarrollada por iSixSigma (**www.isixsigma.com/tt/**), describe un amplio abanico de herramientas y métodos útiles para la administración de la calidad.

NASA Quality Resources, desarrollada por el Centro Coddard de Vuelos Espaciales (**sw-assurance.gsfc.nasa.gov/index.php**), contiene formatos, plantillas, listas de verificación y herramientas que son útiles para el ACS.

16.9 RESUMEN

El aseguramiento de la calidad del software es una actividad sombrilla de la ingeniería de software que se aplica en cada etapa del proceso del software. El ACS incluye procedimientos para la aplicación eficaz de métodos y herramientas, supervisa las actividades de control de calidad, tales como las revisiones técnicas y las pruebas del software, procedimientos para la administración del cambio, y procedimientos para asegurar el cumplimiento de las normas y mecanismos de medición y elaboración de reportes.

Para llevar a cabo el aseguramiento de la calidad del software de manera adecuada, deben recabarse, evaluarse y divulgarse datos sobre el proceso de la ingeniería de software. Los métodos estadísticos aplicados al ACS ayudan a mejorar la calidad del producto y del proceso de software mismo. Los modelos de confiabilidad del software amplían las mediciones, lo que permite que los datos obtenidos acerca de los defectos se extrapolen hacia tasas de falla proyectadas y hacia la elaboración de pronósticos de confiabilidad.

En resumen, deben tomarse en cuenta las palabras de Dunn y Ullman [Dun82]: "El aseguramiento de la calidad del software es el mapeo de los preceptos administrativos y de las disciplinas de diseño del aseguramiento de la calidad, en el ámbito administrativo y tecnológico aplicable a la ingeniería de software." La capacidad de asegurar la calidad es la medida de una

5 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.

disciplina madura de la ingeniería. Cuando el mapeo se lleva a cabo con éxito, el resultado es una ingeniería de software madura.

Problemas y puntos por evaluar

16.1. Algunas personas afirman que "el control de la variación es el corazón del control de calidad". Como todo programa que se crea es diferente de cualquier otro programa, ¿cuáles son las variaciones que se buscan y cómo se controlan?

16.2. ¿Es posible evaluar la calidad del software si el cliente cambia continuamente lo que se supone que debe hacerse?

16.3. La calidad y confiabilidad son conceptos relacionados, pero difieren en lo fundamental por varias razones. Analice las diferencias.

16.4. ¿Un programa puede corregirse y aún así ser confiable? Explique su respuesta.

16.5. ¿Un programa puede corregirse y tener buena calidad? Explique lo que responda.

16.6. ¿Por qué es frecuente que haya tensiones entre el grupo de ingeniería de software y el del aseguramiento de la calidad? ¿Es saludable eso?

16.7. El lector tiene la responsabilidad de mejorar la calidad del software en su organización. ¿Qué es lo primero que debe hacer? ¿Qué es lo siguiente?

16.8. Además de contar los errores y defectos, ¿hay otras características cuantificables de software que impliquen calidad? ¿Cuáles son y cómo podrían medirse directamente?

16.9. El concepto del tiempo medio para la falla del software es objeto de críticas. Explique por qué.

16.10. Considere dos sistemas cuya seguridad sea crítica y que estén controlados por computadora. Enliste al menos tres peligros que se relacionen directamente con fallas del software.

16.11. Obtenga una copia de las normas ISO 9001:2000 e ISO 9000-3. Prepare una presentación que analice tres requerimientos de ISO 9001 y la forma en la que se apliquen en el contexto del software.

Lecturas y fuentes de información adicionales

Los libros de Hoyle (*Quality Management Fundamentals*, Butterworth-Heinemann, 2007), Tian (*Software Quality Engineering*, Wiley-IEEE Computer Society Press, 2005), El Emam (*The ROI from Software Quality*, Auerbach, 2005) y Horch (*Practical Guide to Software Quality Management*, Artech House, 2003), y Nance y Arthur (*Managing Software Quality*, Springer, 2002) son presentaciones excelentes en el nivel de administración acerca de los beneficios de los programas formales de aseguramiento de la calidad del software de computadora. Las obras de Deming [Dem86], Juran (*Juran on Quality by Design*, Free Press, 1992) y Crosby ([Cro79], así como *Quality is Still Free*, McGraw-Hill, 1995) no se abocan al software, pero son una lectura obligada para los altos directivos que tengan responsabilidades en el desarrollo del software. Gluckman y Roome (*Everyday Heroes of the Quality Movement*, Dorset House, 1993) humanizan los aspectos de la calidad a través de la historia de los actores participantes en el proceso. Kan (*Metrics and Models in Software Quality Engineering*, Addison-Wesley, 1995) presenta un enfoque cuantitativo de la calidad del software.

Los libros de Evans (*Total Quality: Management, Organization and Strategy*, 4a. ed., South Western College Publishing, 2004), Bru (*Six Sigma for Managers*, McGraw-Hill, 2005) y Dobb (*ISO 9001:2000 Quality Registration Step-by-Step*, 3a. ed., Butterworth-Heinemann, 2004) son representativos de los muchos que se han escrito sobre Seis Sigma e ISO 9001:2000, respectivamente.

Pham (*System Software Reliability,* Springer, 2006), Musa (*Software Reliability Engineering: More Reliable Software, Faster Development and Testing*, 2a. ed., McGraw-Hill, 2004) y Peled (*Software Reliability Methods*, Springer, 2001) proporcionan guías prácticas que describen los métodos para medir y analizar la confiabilidad del software.

Vincoli (*Basic Guide to System Safety*, Wiley 2006), Dhillon (*Engineering Safety*, World Scientific Publishing Co., Inc., 2003), Hermann (*Software Safety and Reliability*, Wiley-IEEE Computer Society Press, 2000), Storey (*Safety-Critical Computer Systems*, Addison-Wesley, 1996) y Leveson [Lev95] aportan los análisis más exhaustivos que se hayan publicado hasta la fecha acerca de la seguridad del software y del sistema. Además, Van

der Meulen (*Definitions for Hardware and Software Safety Engineers*, Springer-Verlag, 2000) ofrece un compendio completo de conceptos y términos importantes para la confiabilidad y la seguridad; Gartner (*Testing Safety-Related Software*, Springer-Verlag, 1999) ofrece una guía especializada para probar sistemas cuya seguridad sea crítica; Friedman y Voas (*Software Assesment: Reliability Safety and Testability*, Wiley, 1995) proveen modelos útiles para evaluar la confiabilidad y la seguridad. Ericson (*Hazard Analysis Techniques for System Safety*, Wiley, 2005) estudia el dominio cada vez más importante del análisis de los peligros.

En internet, hay una amplia variedad de fuentes de información sobre el aseguramiento de la calidad del software y otros temas relacionados. En el sitio web del libro, **www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm**, existe una lista actualizada de referencias existentes en la red mundial que son relevantes para el ACS.