PARTE

Tres

ADMINISTRACIÓN DE LA CALIDAD

En esta parte de *Ingeniería del software*, aprenderá los principios, conceptos y técnicas que se aplican para administrar y controlar la calidad del software. En los próximos capítulos se responderán preguntas como las siguientes:

- ¿Cuáles son las características generales del software de alta calidad?
- ¿Cómo se revisa la calidad y de qué manera se llevan a cabo revisiones eficaces?
- ¿En qué consiste el aseguramiento de la calidad del software?
- ¿Qué estrategias son aplicables para probar el software?
- ¿Qué métodos se utilizan para diseñar casos de prueba eficaces?
- ¿Hay métodos realistas que aseguren que el software es correcto?
- ¿Cómo pueden administrarse y controlarse los cambios que siempre ocurren cuando se elabora el software?
- ¿Qué medidas y unidades de medición se usan para evaluar la calidad de los modelos de requerimientos y diseño, código fuente y casos de prueba?

Una vez respondidas estas preguntas, el lector estará mejor preparado para asegurar que se ha producido software de alta calidad.

CAPÍTULO

14 Conceptos de calidad

Conceptos clave

El redoble de tambores para mejorar la calidad del software comenzó tan luego que éste empezó a integrarse en cada faceta de nuestras vidas. En la década de 1990, las principales corporaciones reconocieron que cada año se desperdiciaban miles de millones de dólares en software que no tenía las características ni la funcionalidad que se habían prometido. Lo que era peor, tanto el gobierno como la industria se preocupaban por la posibilidad de que alguna falla de software pudiera afectar infraestructura importante y provocara pérdidas de decenas de miles de millones de dólares. Al despuntar el nuevo siglo, *CIO Magazine* [Lev01] dio la alerta: "Dejemos de desperdiciar $78 mil millones de dólares al año", y lamentaba el hecho de que "las empresas estadounidenses gastan miles de millones de dólares en software que no hace lo que se supone que debe hacer". *InformationWeek* [Ric01] se hizo eco de la misma preocupación:

> A pesar de las buenas intenciones, el código defectuoso sigue siendo el duende de la industria del software, es responsable hasta de 45% del tiempo que están fuera los sistemas basados en computadoras y costó a las empresas estadounidenses alrededor de $100 mil millones de dólares el último año en pérdidas de productividad y reparaciones, afirma Standish Group, empresa de investigación de mercados. Eso no incluye el costo que implica perder a los clientes disgustados. Como los productores de tecnologías de información escriben aplicaciones que se basan en software empacado en infraestructura, el código defectuoso también puede inutilizar aplicaciones personalizadas...
>
> Pero, ¿cuán malo es el software defectuoso? Las respuestas varían, mas los expertos dicen que sólo se requiere de tres a cuatro defectos por cada 1 000 líneas de código para que un programa tenga mal desempeño. Hay que pensar que la mayoría de los programadores cometen un error en cada 10 líneas de código que escriben, lo que, multiplicado por los millones de líneas que hay en muchos productos comerciales, permite imaginar que la corrección de los errores cuesta a los vendedores de software al menos la mitad de sus presupuestos de desarrollo durante las pruebas. ¿Comprende lo que esto significa?

Una mirada rápida

¿Qué es? La respuesta no es tan fácil como quizá se piense. La calidad se reconoce cuando se ve, por lo que puede ser algo difícil de definir. Pero para el software de computadora, la calidad es algo que debe definirse, y eso es lo que haremos en este capítulo.

¿Quién lo hace? Los involucrados en el proceso del software —ingenieros, gerentes y todos los participantes— son los responsables de la calidad.

¿Por qué es importante? Puede hacerse bien o puede repetirse. Si un equipo de software hace énfasis en la calidad de todas las actividades de la ingeniería de software, se reduce el número de repeticiones que deben hacerse. Esto da como resultado menores costos y, lo que es más importante, mejora el tiempo de llegada al mercado.

¿Cuáles son los pasos? Para lograr software de alta calidad, deben ocurrir cuatro actividades: usar procesos y prácticas probados de la ingeniería de software, administrar bien el proyecto, realizar un control de calidad exhaustivo y contar con infraestructura de aseguramiento de la calidad.

¿Cuál es el producto final? Software que satisface las necesidades del consumidor, con un desempeño apropiado y confiable, y que agrega valor para todos los que lo utilizan.

¿Cómo me aseguro de que lo hice bien? Hay que dar seguimiento a la calidad, estudiando los resultados de todas las actividades de control de calidad y midiendo ésta con el estudio de los errores antes de la entrega y de los defectos detectados en el campo.

En 2005, *ComputerWorld* [Hil05] se quejaba de que "el mal software es una plaga en casi todas las organizaciones que emplean computadoras, lo que ocasiona horas de trabajo perdidas por el tiempo que están fuera de uso las máquinas, por datos perdidos o corrompidos, oportunidades de venta perdidas, costos elevados de apoyo y mantenimiento, y poca satisfacción del cliente. Un año después, *InfoWorld* [Fos06] escribió acerca del "lamentable estado de la calidad del software" e informaba que el problema de la calidad no había mejorado.

Actualmente, la calidad del software es preocupante, pero, ¿de quién es la culpa? Los clientes culpan a los desarrolladores, pues afirman que sus prácticas descuidadas producen software de mala calidad. Los desarrolladores culpan a los clientes (y a otros participantes) con la afirmación de que las fechas de entrega irracionales y un flujo continuo de cambios los obligan a entregar software antes de haber sido validado por completo. ¿Quién tiene la razón? *Ambos*, y ése es el problema. En este capítulo se analiza el concepto de calidad del software y por qué es útil estudiarlo con seriedad siempre que se apliquen prácticas de ingeniería de software.

14.1 ¿Qué es calidad?

En su libro místico, *El zen y el arte del mantenimiento de la motocicleta*, Robert Persig [Per74] comenta lo siguiente acerca de lo que llamamos *calidad*:

> Calidad... sabes lo que es, pero no sabes lo que es. Pero eso es una contradicción. Algunas cosas son mejores que otras; es decir, tienen más calidad. Pero cuando tratas de decir lo que es la calidad, además de las cosas que la tienen, todo se desvanece... No hay nada de qué hablar. Pero si no puede decirse qué es Calidad, ¿cómo saber lo que es, o incluso saber que existe? Si nadie sabe lo que es, entonces, para todos los propósitos prácticos, no existe en absoluto. Pero para todos los propósitos prácticos, en realidad sí existe. ¿En qué otra cosa se basan las calificaciones? ¿Por qué paga fortunas la gente por algunos artículos y tira otros a la basura? Es obvio que algunas cosas son mejores que otras... pero, ¿en qué son mejores? Y así damos vueltas y más vueltas, ruedas de metal que patinan sin nada en lo que hagan tracción. ¿Qué demonios es la Calidad? ¿Qué es?

Es cierto: ¿qué es?

CONSEJO

¿Cuáles son las diferentes maneras en las que puede verse la calidad?

En un nivel algo pragmático, David Garvin [Gar84], de Harvard Business School, sugiere que "la calidad es un concepto complejo y de facetas múltiples" que puede describirse desde cinco diferentes puntos de vista. El *punto de vista trascendental* dice (como Persig) que la calidad es algo que se reconoce de inmediato, pero que no es posible definir explícitamente. El *punto de vista del usuario* concibe la calidad en términos de las metas específicas del usuario final. Si un producto las satisface, tiene calidad. El *punto de vista del fabricante* la define en términos de las especificaciones originales del producto. Si éste las cumple, tiene calidad. El *punto de vista del producto* sugiere que la calidad tiene que ver con las características inherentes (funciones y características) de un producto. Por último, el *punto de vista basado en el valor* la mide de acuerdo con lo que un cliente está dispuesto a pagar por un producto. En realidad, la calidad incluye todo esto y más.

Cita:

"La gente olvida cuán rápido hiciste un trabajo, pero siempre recuerda cuán bien lo realizaste."

Howard Newton

La *calidad del diseño* se refiere a las características que los diseñadores especifican para un producto. El tipo de materiales, tolerancias y especificaciones del desempeño, todo contribuye a la calidad del diseño. Si se utilizan mejores materiales, tolerancias más estrictas y se especifican mayores niveles de desempeño, la calidad del diseño de un producto se incrementa si se fabrica de acuerdo con las especificaciones.

En el desarrollo del software, la *calidad del diseño* incluye el grado en el que el diseño cumple las funciones y características especificadas en el modelo de requerimientos. La *calidad de la conformidad* se centra en el grado en el que la implementación se apega al diseño y en el que el sistema resultante cumple sus metas de requerimientos y desempeño.

Pero, ¿son la calidad del diseño y de la conformidad los únicos aspectos que deben considerar los ingenieros de software? Robert Glass [Gla98] afirma que es mejor plantear una relación más intuitiva:

satisfacción del usuario = producto que funciona + buena calidad + entrega dentro del presupuesto y plazo

En última instancia, Glass sostiene que la calidad es importante, pero que si el usuario no está satisfecho, nada de lo demás importa. DeMarco [DeM98] refuerza esta opinión al decir que "la calidad de un producto está en función de cuánto cambia al mundo para bien". Este punto de vista de la calidad afirma que si un producto de software beneficia mucho a los usuarios finales, éstos se mostrarán dispuestos a tolerar problemas ocasionales de confiabilidad o desempeño.

14.2 Calidad del software

Incluso los desarrolladores de software más experimentados estarán de acuerdo en que obtener software de alta calidad es una meta importante. Pero, ¿cómo se define la calidad del *software*? En el sentido más general se define[1] como: *Proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan*.

Hay pocas dudas acerca de que la definición anterior podría modificarse o ampliarse en un debate sin fin. Para propósitos de este libro, la misma sirve a fin de enfatizar tres puntos importantes:

1. Un proceso *eficaz de software* establece la infraestructura que da apoyo a cualquier esfuerzo de elaboración de un producto de software de alta calidad. Los aspectos de administración del proceso generan las verificaciones y equilibrios que ayudan a evitar que el proyecto caiga en el caos, contribuyente clave de la mala calidad. Las prácticas de ingeniería de software permiten al desarrollador analizar el problema y diseñar una solución sólida, ambas actividades críticas de la construcción de software de alta calidad. Por último, las actividades sombrilla, tales como administración del cambio y revisiones técnicas, tienen tanto que ver con la calidad como cualquier otra parte de la práctica de la ingeniería de software.
2. Un *producto útil* entrega contenido, funciones y características que el usuario final desea; sin embargo, de igual importancia es que entrega estos activos en forma confiable y libre de errores. Un producto útil siempre satisface los requerimientos establecidos en forma explícita por los participantes. Además, satisface el conjunto de requerimientos (por ejemplo, la facilidad de uso) con los que se espera que cuente el software de alta calidad.
3. Al *agregar valor para el productor y para el usuario* de un producto, el software de alta calidad proporciona beneficios a la organización que lo produce y a la comunidad de usuarios finales. La organización que elabora el software obtiene valor agregado porque el software de alta calidad requiere un menor esfuerzo de mantenimiento, menos errores que corregir y poca asistencia al cliente. Esto permite que los ingenieros de software dediquen más tiempo a crear nuevas aplicaciones y menos a repetir trabajos mal hechos. La comunidad de usuarios obtiene valor agregado porque la aplicación provee una capacidad útil en forma tal que agiliza algún proceso de negocios. El resultado final es 1) mayores utilidades por el producto de software, 2) más rentabilidad cuando una

1 Esta definición ha sido adaptada de [Bes04] y sustituye aquélla más orientada a la manufactura presentada en ediciones anteriores de este libro.

aplicación apoya un proceso de negocios y 3) mejor disponibilidad de información, que es crucial para el negocio.

14.2.1 Dimensiones de la calidad de Garvin

David Garvin [Gar87] sugiere que la calidad debe tomarse en cuenta, adoptando un punto de vista multidimensional que comience con la evaluación de la conformidad y termine con una visión trascendental (estética). Aunque las ocho dimensiones de Garvin de la calidad no fueron desarrolladas específicamente para el software, se aplican a la calidad de éste:

Calidad del desempeño. ¿El software entrega todo el contenido, las funciones y las características especificadas como parte del modelo de requerimientos, de manera que da valor al usuario final?

Calidad de las características. ¿El software tiene características que sorprenden y agradan la primera vez que lo emplean los usuarios finales?

Confiabilidad. ¿El software proporciona todas las características y capacidades sin fallar? ¿Está disponible cuando se necesita? ¿Entrega funcionalidad libre de errores?

Conformidad. ¿El software concuerda con los estándares locales y externos que son relevantes para la aplicación? ¿Concuerda con el diseño *de facto* y las convenciones de código? Por ejemplo, ¿la interfaz de usuario está de acuerdo con las reglas aceptadas del diseño para la selección de menú o para la entrada de datos?

Durabilidad. ¿El software puede recibir mantenimiento (cambiar) o corregirse (depurarse) sin la generación inadvertida de eventos colaterales? ¿Los cambios ocasionarán que la tasa de errores o la confiabilidad disminuyan con el tiempo?

Servicio. ¿Existe la posibilidad de que el software reciba mantenimiento (cambios) o correcciones (depuración) en un periodo de tiempo aceptablemente breve? ¿El equipo de apoyo puede adquirir toda la información necesaria para hacer cambios o corregir defectos? Douglas Adams [Ada93] hace un comentario irónico que parece pertinente: "La diferencia entre algo que puede salir mal y algo que posiblemente no salga mal es que cuando esto último sale mal, por lo general es imposible corregirlo o repararlo."

Estética. No hay duda de que todos tenemos una visión diferente y muy subjetiva de lo que es estético. Aun así, la mayoría de nosotros estaría de acuerdo en que una entidad estética posee cierta elegancia, un flujo único y una "presencia" obvia que es difícil de cuantificar y que, no obstante, resulta evidente. El software estético tiene estas características.

Percepción. En ciertas situaciones, existen prejuicios que influirán en la percepción de la calidad por parte del usuario. Por ejemplo, si se introduce un producto de software elaborado por un proveedor que en el pasado ha demostrado mala calidad, se estará receloso y la percepción de la calidad del producto tendrá influencia negativa. De manera similar, si un vendedor tiene una reputación excelente se percibirá buena calidad, aun si ésta en realidad no existe.

Las dimensiones de la calidad de Garvin dan una visión "suave" de la calidad del software. Muchas de estas dimensiones (aunque no todas) sólo pueden considerarse de manera subjetiva. Por esta razón, también se necesita un conjunto de factores "duros" de la calidad que se clasifican en dos grandes grupos: 1) factores que pueden medirse en forma directa (por ejemplo, defectos no descubiertos durante las pruebas) y 2) factores que sólo pueden medirse indirectamente (como la usabilidad o la facilidad de recibir mantenimiento). En cada caso deben hacerse mediciones: debe compararse el software con algún dato para llegar a un indicador de la calidad.

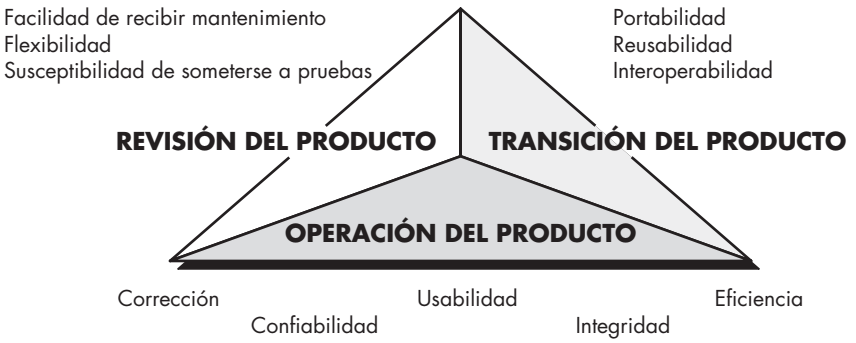


FIGURA 14.1
Factores de la calidad de McCall

14.2.2 Factores de la calidad de McCall

McCall, Richards y Walters [McC77] proponen una clasificación útil de los factores que afectan la calidad del software. Éstos se ilustran en la figura 14.1 y se centran en tres aspectos importantes del producto de software: sus características operativas, su capacidad de ser modificado y su adaptabilidad a nuevos ambientes.

En relación con los factores mencionados en la figura 14.1, McCall *et al.*, hacen las descripciones siguientes:

Corrección. Grado en el que un programa satisface sus especificaciones y en el que cumple con los objetivos de la misión del cliente.

Confiabilidad. Grado en el que se espera que un programa cumpla con su función y con la precisión requerida [debe notarse que se han propuesto otras definiciones más completas de la confiabilidad (véase el capítulo 25)].

Eficiencia. Cantidad de recursos de cómputo y de código requeridos por un programa para llevar a cabo su función.

Integridad. Grado en el que es posible controlar el acceso de personas no autorizadas al software o a los datos.

Usabilidad. Esfuerzo que se requiere para aprender, operar, preparar las entradas e interpretar las salidas de un programa.

Facilidad de recibir mantenimiento. Esfuerzo requerido para detectar y corregir un error en un programa (ésta es una definición muy limitada).

Flexibilidad. Esfuerzo necesario para modificar un programa que ya opera.

Susceptibilidad de someterse a pruebas. Esfuerzo que se requiere para probar un programa a fin de garantizar que realiza la función que se pretende.

Portabilidad. Esfuerzo que se necesita para transferir el programa de un ambiente de sistema de hardware o software a otro.

Reusabilidad. Grado en el que un programa (o partes de uno) pueden volverse a utilizar en otras aplicaciones (se relaciona con el empaque y el alcance de las funciones que lleva a cabo el programa).

Interoperabilidad. Esfuerzo requerido para acoplar un sistema con otro.

Cita:

"La amargura de la mala calidad permanece mucho tiempo después de que ya se ha olvidado la dulzura de haber cumplido el plazo programado."

Karl Weigers (cita sin acreditación)

Es difícil — y, en ciertos casos, imposible— desarrollar mediciones directas[2] de estos factores de la calidad. En realidad, muchas de las unidades de medida definidas por McCall *et al.*, sólo

2 Una *medición directa* implica que hay un solo valor cuantificable que da una indicación directa del atributo en estudio. Por ejemplo, el "tamaño" de un programa se mide directamente, contando el número de sus líneas de código.

pueden obtenerse de manera indirecta. Sin embargo, la evaluación de la calidad de una aplicación por medio de estos factores dará un indicio sólido de ella.

14.2.3 Factores de la calidad ISO 9126

El estándar ISO 9126 se desarrolló con la intención de identificar los atributos clave del software de cómputo. Este sistema identifica seis atributos clave de la calidad:

Funcionalidad. Grado en el que el software satisface las necesidades planteadas según las establecen los atributos siguientes: adaptabilidad, exactitud, interoperabilidad, cumplimiento y seguridad.

Confiabilidad. Cantidad de tiempo que el software se encuentra disponible para su uso, según lo indican los siguientes atributos: madurez, tolerancia a fallas y recuperación.

Usabilidad. Grado en el que el software es fácil de usar, según lo indican los siguientes subatributos: entendible, aprendible y operable.

Eficiencia. Grado en el que el software emplea óptimamente los recursos del sistema, según lo indican los subatributos siguientes: comportamiento del tiempo y de los recursos.

Facilidad de recibir mantenimiento. Facilidad con la que pueden efectuarse reparaciones al software, según lo indican los atributos que siguen: analizable, cambiable, estable, susceptible de someterse a pruebas.

Portabilidad. Facilidad con la que el software puede llevarse de un ambiente a otro según lo indican los siguientes atributos: adaptable, instalable, conformidad y sustituible.

Aunque resulta tentador desarrollar mediciones cuantitativas para los factores de calidad mencionados aquí, también puede crearse una lista de comprobación de atributos que den una indicación sólida de la presencia del factor.

Igual que otros factores de la calidad del software estudiados en las subsecciones anteriores, los factores ISO 9126 no necesariamente conducen a una medición directa. Sin embargo, proporcionan una base útil para hacer mediciones indirectas y una lista de comprobación excelente para evaluar la calidad del sistema.

14.2.4 Factores de calidad que se persiguen

Cita:

"Cualquier actividad se vuelve creativa cuando a quien la realiza le importa hacerla bien, o mejor."

John Updike

Las dimensiones y factores de la calidad presentados en las secciones 14.2.1 y 14.2.2 se centran en el software como un todo y pueden utilizarse como indicación general de la calidad de una aplicación. Un equipo de software puede desarrollar un conjunto de características de la calidad y las preguntas asociadas correspondientes que demuestren[3] el grado en el que se satisface cada factor. Por ejemplo, McCall identifica la *usabilidad* como un factor importante de la calidad. Si se pidiera revisar una interfaz de usuario para evaluar su usabilidad, ¿cómo se haría? Se comenzaría con los subatributos propuestos por McCall —entendible, aprendible y operable— pero en un sentido práctico: ¿qué significan éstos?

Para hacer la evaluación, se necesita determinar atributos específicos y medibles (o al menos reconocibles) de la interfaz. Por ejemplo [Bro03]:

Intuitiva. Grado en el que la interfaz sigue patrones esperados de uso, de modo que hasta un novato la pueda utilizar sin mucha capacitación.

- ¿La interfaz lleva hacia una comprensión fácil?
- ¿Todas las operaciones son fáciles de localizar e iniciar?
- ¿La interfaz usa una metáfora reconocible?
- ¿La entrada está especificada de modo que economiza el uso del teclado o del ratón?

3 Estas características y preguntas se plantearían como parte de la revisión del software (véase el capítulo 15).

- ¿La entrada sigue las tres reglas de oro? (véase el capítulo 11)
- ¿La estética ayuda a la comprensión y uso?

Eficiencia. Grado en el que es posible localizar o iniciar las operaciones y la información.

- ¿La distribución y estilo de la interfaz permite que un usuario introduzca con eficiencia las operaciones y la información?
- ¿Una secuencia de operaciones (o entrada de datos) puede realizarse con economía de movimientos?
- ¿Los datos de salida o el contenido están presentados de modo que se entienden de inmediato?
- ¿Las operaciones jerárquicas están organizadas de manera que minimizan la profundidad con la que debe navegar el usuario para hacer que alguna se ejecute?

Robustez. Grado en el que el software maneja entradas erróneas de datos o en el que se presenta interacción inapropiada por parte del usuario.

- ¿El software reconocerá el error si entran datos en el límite de lo permitido o más allá y, lo que es más importante, continuará operando sin fallar ni degradarse?
- ¿La interfaz reconocerá los errores cognitivos o de manipulación y guiará en forma explícita al usuario de vuelta al camino correcto?
- ¿La interfaz da un diagnóstico y guía útiles cuando se descubre una condición de error (asociada con la funcionalidad del software)?

Riqueza. Grado en el que la interfaz provee un conjunto abundante de características.

- ¿Puede personalizarse la interfaz según las necesidades específicas del usuario?
- ¿La interfaz tiene gran capacidad para permitir al usuario identificar una secuencia de operaciones comunes con una sola acción o comando?

A medida que se desarrolla el diseño de la interfaz, el equipo del software revisa el prototipo del diseño y plantea las preguntas anteriores. Si la respuesta a la mayor parte de éstas es "sí", es probable que la interfaz de usuario sea de buena calidad. Para cada factor de la calidad que se desee evaluar se desarrollan preguntas similares.

14.2.5 Transición a un punto de vista cuantitativo

En las subsecciones anteriores se presentaron varios factores cualitativos para la "medición" de la calidad del software. La comunidad de la ingeniería de software trata de obtener mediciones precisas de la calidad de éste y a veces se ve frustrada por la naturaleza subjetiva de la actividad. Cavano y McCall [Cav78] analizan esta situación:

> La determinación de la calidad es un factor clave en los eventos cotidianos: concursos para catar vinos, eventos deportivos [como gimnasia], competencias de talento, etc. En estas situaciones se juzga la calidad del modo más fundamental y directo: la comparación directa de objetos en condiciones idénticas y con conceptos predeterminados. El vino se juzga de acuerdo con su claridad, color, buqué, sabor, etc. Sin embargo, este tipo de juicio es muy subjetivo; para que tenga algún valor, debe ser hecho por un experto.
>
> La subjetividad y la especialización también se aplican a la determinación de la calidad del software. Para ayudar a resolver este problema, es necesario tener una definición más precisa de la calidad del software, así como una forma de realizar mediciones cuantitativas de la calidad a fin de hacer análisis objetivos... Como no existe algo parecido al conocimiento absoluto, no debe esperarse medir con toda exactitud la calidad del software, porque toda medición es imperfecta. Jacob Bronkowski

describió esta paradoja del conocimiento del modo siguiente: "Año con año desarrollamos instrumentos más precisos para observar la naturaleza con más nitidez. Y cuando vemos las observaciones, nos decepcionamos porque son borrosas y sentimos que son tan inciertas como siempre".

En el capítulo 23 se presenta un conjunto de unidades de medida aplicables a la evaluación cuantitativa de la calidad del software. En todos los casos, las unidades representan mediciones indirectas, es decir, nunca miden realmente la *calidad*, sino alguna manifestación de ella. El factor que complica todo es la relación precisa entre la variable que se mide y la calidad del software.

14.3 El dilema de la calidad del software

Cuando se enfrente al dilema de la calidad (y todos lo hacen en un momento u otro), trate de alcanzar el balance: suficiente esfuerzo para producir una calidad aceptable sin que sepulte al proyecto.

En una entrevista [Ven03] publicada en la web, Bertrand Meyer analiza lo que se denomina el *dilema de la calidad:*

> Si produce un sistema de software de mala calidad, usted pierde porque nadie lo querrá comprar. Por otro lado, si dedica un tiempo infinito, demasiado esfuerzo y enormes sumas de dinero para obtener un elemento perfecto de software, entonces tomará tanto tiempo terminarlo y será tan caro de producir que de todos modos quedará fuera del negocio. En cualquier caso, habrá perdido la ventana de mercado, o simplemente habrá agotado sus recursos. De modo que las personas de la industria tratan de situarse en ese punto medio mágico donde el producto es suficientemente bueno para no ser rechazado de inmediato, no en la evaluación, pero tampoco es un objeto perfeccionista ni con demasiado trabajo que lo convierta en algo que requiera demasiado tiempo o dinero para ser terminado.

Es correcto afirmar que los ingenieros de software deben tratar de producir sistemas de alta calidad. Es mejor aplicar buenas prácticas al intento de lograrlo. Pero la situación descrita por Meyer proviene de la vida real y representa un dilema incluso para las mejores organizaciones de ingeniería de software.

14.3.1 Software "suficientemente bueno"

En palabras sencillas, si damos por válido el argumento de Meyer, ¿es aceptable producir software "suficientemente bueno"? La respuesta a esta pregunta debe ser "sí", porque las principales compañías de software lo hacen a diario. Crean software con errores detectados y lo distribuyen a una gran población de usuarios finales. Reconocen que algunas de las funciones y características de la versión 1.0 tal vez no sean de la calidad más alta y planean hacer mejoras en la versión 2.0. Hacen esto, sabiendo que algunos clientes se quejarán; reconocen que el tiempo para llegar al mercado actúa contra la mejor calidad, y liberan el software, siempre y cuando el producto entregado sea "suficientemente bueno".

Exactamente, ¿qué significa "suficientemente bueno"? El software suficientemente bueno contiene las funciones y características de alta calidad que desean los usuarios, pero al mismo tiempo tiene otras más oscuras y especializadas que contienen errores conocidos. El vendedor de software espera que la gran mayoría de usuarios finales perdone los errores gracias a que estén muy contentos con la funcionalidad de la aplicación.

Esta idea resulta familiar para muchos lectores. Si usted es uno de ellos, le pido que considere algunos de los argumentos contra lo "suficientemente bueno".

Es verdad que lo "suficientemente bueno" puede funcionar en ciertos dominios de aplicación y para unas cuantas compañías grandes de software. Después de todo, si una empresa tiene un presupuesto enorme para mercadotecnia y convence a suficientes personas de que compren la versión 1.0, habrá tenido éxito en capturarlos. Como ya se dijo, puede sostener que en las versiones posteriores mejorará la calidad. Al entregar la versión 1.0 suficientemente buena, habrá capturado al mercado.

Si el lector trabaja para una compañía pequeña, debe tener cuidado con esta filosofía. Al entregar un producto suficientemente bueno (defectuoso), corre el riesgo de causar un daño permanente a la reputación de su compañía. Tal vez nunca tenga la oportunidad de entregar una versión 2.0 porque los malos comentarios quizá ocasionen que las ventas se desplomen y que la empresa desaparezca.

Si trabaja en ciertos dominios de aplicación (por ejemplo, software incrustado en tiempo real) o si construye software de aplicación integrado con hardware (como el software automotriz o de telecomunicaciones), entregar software con errores conocidos es una negligencia y deja expuesta a su compañía a litigios costosos. En ciertos casos, incluso, puede ser un delito. ¡Nadie quiere tener software suficientemente bueno en los aviones!

Así que proceda con cautela si piensa que lo "suficientemente bueno" es un atajo que puede resolver los problemas de calidad de su software. Tal vez funcione, pero sólo para unos cuantos y en un conjunto limitado de dominios de aplicación.[4]

14.3.2 El costo de la calidad

El argumento es algo parecido a esto: *sabemos que la calidad es importante, pero cuesta tiempo y dinero —demasiado tiempo y dinero— lograr el nivel de calidad en el software que en realidad queremos.* Visto así, este argumento parece razonable (véanse los comentarios anteriores de Meyer en esta sección). No hay duda de que la calidad tiene un costo, pero la mala calidad también lo tiene —no sólo para los usuarios finales que deban vivir con el software defectuoso, sino también para la organización del software que lo elaboró y que debe darle mantenimiento—. La pregunta real es ésta: *¿por cuál costo debemos preocuparnos?* Para responder a esta pregunta debe entenderse tanto el costo de tener calidad como el del software de mala calidad.

El *costo de la calidad* incluye todos los costos en los que se incurre al buscar la calidad o al realizar actividades relacionadas con ella y los costos posteriores de la falta de calidad. Para entender estos costos, una organización debe contar con unidades de medición que provean el fundamento del costo actual de la calidad, que identifiquen las oportunidades para reducir dichos costos y que den una base normalizada de comparación. El costo de la calidad puede dividirse en los costos que están asociados con la prevención, la evaluación y la falla.

No tema incurrir en costos significativos por la prevención. Esté seguro de que su inversión tendrá un rendimiento excelente.

Los *costos de prevención* incluyen lo siguiente: 1) el costo de las actividades de administración requeridas para planear y coordinar todas las actividades de control y aseguramiento de la calidad, 2) el costo de las actividades técnicas agregadas para desarrollar modelos completos de los requerimientos y del diseño, 3) los costos de planear las pruebas y 4) el costo de toda la capacitación asociada con estas actividades.

Los *costos de evaluación* incluyen las actividades de investigación de la condición del producto la "primera vez" que pasa por cada proceso. Algunos ejemplos de costos de evaluación incluyen los siguientes:

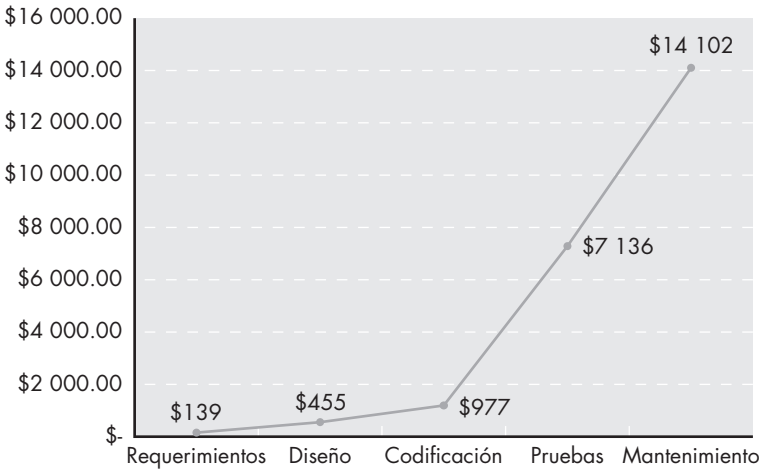
- El costo de efectuar revisiones técnicas (véase el capítulo 15) de los productos del trabajo de la ingeniería de software.
- El costo de recabar datos y unidades de medida para la evaluación (véase el capítulo 23)
- El costo de hacer las pruebas y depurar (véanse los capítulos 18 a 21)

Los *costos de falla* son aquellos que se eliminarían si no hubiera errores antes o después de enviar el producto a los consumidores. Los costos de falla se subdividen en internos y externos. Se incurre en *costos internos de falla* cuando se detecta un error en un producto antes del envío. Los costos internos de falla incluyen los siguientes:

4 Un análisis útil de los pros y contras del software "suficientemente bueno" se encuentra en [Bre02].

FIGURA 14.2

Costo relativo de corregir errores y defectos (cifras en dólares estadounidenses)
Fuente: Adaptado de [Boe01b].



- El costo requerido por efectuar repeticiones (reparaciones para corregir un error).
- El costo en el que se incurre cuando una repetición genera inadvertidamente efectos colaterales que deban mitigarse.
- Los costos asociados con la colección de las unidades de medida de la calidad que permitan que una organización evalúe los modos de la falla.

Cita:

"Toma menos tiempo hacer algo bien que explicar por qué se hizo mal."

H. W. Longfellow

Los *costos externos de falla* se asocian con defectos encontrados después de que el producto se envió a los consumidores. Algunos ejemplos de costos externos de falla son los de solución de quejas, devolución y sustitución del producto, ayuda en línea y trabajo asociado con la garantía. La mala reputación y la pérdida resultante de negocios es otro costo externo de falla que resulta difícil de cuantificar y que, sin embargo, es real. Cuando se produce software de mala calidad, suceden cosas malas.

En lo que constituye una acusación contra los desarrolladores de software que se rehúsan a considerar los costos de falla externos, Cem Kaner [Kan95] afirma lo siguiente:

> Muchos de los costos de falla externos, tales como los fondos de comercialización, son difíciles de cuantificar, por lo que muchas compañías los ignoran cuando calculan sus relaciones costo-beneficio. Otros costos externos de falla pueden reducirse (al dar un apoyo barato debido a la mala calidad después de hacer la venta, o al cobrar el apoyo a los consumidores) sin que se incremente la satisfacción del cliente. Al ignorar los costos que los malos productos generan a nuestros compradores, los ingenieros de la calidad estimulan una toma de decisiones que los hace víctimas en lugar de satisfacerlos.

Como es de esperar, los costos relacionados con la detección y la corrección de errores o defectos se incrementan en forma abrupta cuando se pasa de la prevención a la detección, a la falla interna y a la externa. La figura 14.2, basada en datos obtenidos por Boehm y Basili [Boe01b] y elaborada por Cigital, Inc. [Cig07], ilustra este fenómeno.

El costo promedio de la industria por corregir un defecto durante la generación de código es aproximadamente de US$977 por error. El promedio del costo en el que incurre la industria por corregir el mismo error si se descubre durante las pruebas del sistema es de US$7 136. Cigital, Inc. [Cig07] tome en cuenta que una aplicación grande contiene 200 errores introducidos durante la codificación.

> De acuerdo con datos promedio, el costo de encontrar y corregir defectos durante la fase de codificación es de US$977 por defecto. Entonces, el costo total por corregir los 200 errores "críticos" durante esta fase es de (200 × US$977) US$195 400, aproximadamente.

> Los datos promedio de la industria indican que el costo de encontrar y corregir defectos durante la fase de pruebas del sistema es de US$7 136 por cada uno. En este caso, si se supone que en dicha fase se descubren aproximadamente 50 defectos críticos (tan sólo 25% de los descubiertos por Cigital en la fase de codificación), el costo de encontrarlos y corregirlos (50 × US$7 136) sería aproximadamente de US$356 800. Esto también habría resultado en 150 errores críticos no detectados ni corregidos. El costo de encontrar y corregir estos 150 defectos en la fase de mantenimiento (150 × US$14 102) habría sido de US$2 115 300. Entonces, el costo total de encontrar y corregir los 200 defectos (US$2 115 300 + US$356 800) después de la fase de codificación habría sido de US$2 472 100.

Aun si la organización de software tuviera costos que fueran la mitad del promedio de la industria (la mayor parte de compañías no tiene idea de cuáles son sus costos), los ahorros asociados con el control de calidad temprano y las actividades para su aseguramiento (efectuadas durante el análisis de los requerimientos y el diseño) serían notables.

14.3.3 Riesgos

En el capítulo 1 de este libro se dijo que "la gente basa su trabajo, confort, seguridad, entretenimiento, decisiones y su propia vida, en software de cómputo. Más vale que esté bien hecho". La implicación es que el software de mala calidad aumenta los riesgos tanto para el desarrollador como para el usuario final. En la subsección anterior se analizó uno de dichos riesgos (el costo). Pero lo perjudicial de las aplicaciones mal diseñadas e implementadas no siempre se mide en dólares y tiempo. Un ejemplo extremo [Gag04] servirá para ilustrar esto.

En el mes de noviembre de 2000, en un hospital de Panamá, 28 pacientes recibieron dosis masivas de rayos gama durante su tratamiento contra diversos tipos de cáncer. En los meses que siguieron, 5 de estos pacientes murieron por envenenamiento radiactivo y 15 más sufrieron complicaciones serias. ¿Qué fue lo que ocasionó esta tragedia? Un paquete de software, desarrollado por una compañía estadounidense, que fue modificado por técnicos del hospital para calcular las dosis de radiación para cada paciente.

Los tres médicos panameños que "pellizcaron" el software para que diera capacidad adicional fueron acusados de asesinato en segundo grado. La empresa de Estados Unidos enfrentó litigios serios en los dos países. Gage y McCormick comentan lo siguiente:

> Éste no es un relato para prevenir a los médicos, aun cuando luchen por estar fuera de la cárcel si no entienden o hacen mal uso de la tecnología. Tampoco es la narración de cómo pueden salir heridos, o algo peor, los seres humanos a causa del software mal diseñado o poco explicado, aunque hay muchos ejemplos al respecto. Ésta es la alerta para cualquier creador de programas de cómputo: la calidad del software importa, las aplicaciones deben ser a prueba de tontos y el código mal desplegado —ya sea incrustado en el motor de un automóvil, un brazo robótico o un dispositivo de curación en un hospital— puede matar.

La mala calidad conlleva riesgos, algunos muy serios.

14.3.4 Negligencia y responsabilidad

La historia es muy común. Una entidad gubernamental o corporativa contrata a una compañía importante de desarrollo de software o a una consultoría para que analice los requerimientos y luego diseñe y construya un "sistema" basado en software para apoyar alguna actividad de importancia. El sistema debe auxiliar a una función corporativa principal (como la administración de pensiones) o a alguna función gubernamental (por ejemplo, la administración del cuidado de la salud o los créditos hipotecarios).

El trabajo comienza con las mejores intenciones por ambas partes, pero en el momento en el que el sistema se entrega, las cosas han marchado mal. El sistema va retrasado, no da los resultados y funciones deseadas, comete errores y no cuenta con la aprobación del cliente. Comienzan los litigios.

En la mayor parte de los casos, el cliente afirma que el desarrollador ha sido negligente (en cuanto a la manera en la que aplicó las prácticas del software), por lo que no merece el pago. Es frecuente que el desarrollador diga que el cliente ha cambiado repetidamente sus requerimientos y trastornado de diversas maneras los acuerdos para el trabajo. En cualquier caso, es la calidad del sistema lo que está en entredicho.

14.3.5 Calidad y seguridad

A medida que aumenta la importancia crítica de los sistemas y aplicaciones basados en web, la seguridad de las aplicaciones se ha vuelto más importante. En pocas palabras, el software que no tiene alta calidad es fácil de penetrar por parte de intrusos y, en consecuencia, el software de mala calidad aumenta indirectamente el riesgo de la seguridad, con todos los costos y problemas que eso conlleva.

En una entrevista para *ComputerWorld*, el autor y experto en seguridad Gary McGraw comenta lo siguiente [Wil05]:

> La seguridad del software se relaciona por completo con la calidad. Debe pensarse en seguridad, confiabilidad, disponibilidad y dependencia, en la fase inicial, en la de diseño, en la de arquitectura, pruebas y codificación, durante todo el ciclo de vida del software [proceso]. Incluso las personas conscientes del problema de la seguridad del software se centran en las etapas finales del ciclo de vida. Entre más pronto se detecte un problema en el software, mejor. Y hay dos clases de problemas. Uno son los errores, que son problemas de implementación. El otro son las fallas del software: problemas de arquitectura en el diseño. La gente presta demasiada atención a los errores pero no la suficiente a las fallas.

Para construir un sistema seguro hay que centrarse en la calidad, y eso debe comenzar durante el diseño. Los conceptos y métodos analizados en la parte 2 del libro llevan a una arquitectura del software que reduce las "fallas". Al eliminar las fallas de arquitectura (con lo que mejora la calidad del software) será más difícil que intrusos penetren en el software.

14.3.6 El efecto de las acciones de la administración

Es frecuente que la calidad del software reciba influencia tanto de las decisiones administrativas como de las tecnológicas. Incluso las mejores prácticas de la ingeniería de software pueden ser arruinadas por malas decisiones gerenciales y por acciones cuestionables de la administración del proyecto.

En la parte 4 de este libro se analiza la administración del proyecto en el contexto del proceso del software. Al iniciar toda tarea del proyecto, el líder de éste tomará decisiones que tienen un efecto significativo en la calidad del producto.

Decisiones de estimación. Como se dice en el capítulo 26, un equipo de software rara vez puede darse el lujo de dar una estimación para el proyecto *antes* de que se hayan establecido las fechas de entrega y especificado un presupuesto general. En vez de ello, el equipo realiza un "filtro sanitario" para garantizar que las fechas de entrega y puntos de revisión son racionales. En muchos casos, hay una presión enorme del tiempo para entrar al mercado que fuerza al equipo a aceptar fechas de entrega irreales. En consecuencia, se toman atajos, se pasan por alto las actividades que elevan la calidad del software y disminuye la calidad del producto. Si una fecha de entrega es irracional, es importante poner los pies sobre la tierra. Explique por qué se necesita más tiempo o, alternativamente, sugiera un subconjunto de funciones que puedan entregarse (sin demasiada calidad) en el tiempo programado.

Decisiones de programación. Cuando se establece un programa de desarrollo de un proyecto de software (véase el capítulo 27), se establece la secuencia de las tareas con base en dependencias. Por ejemplo, como el componente **A** depende del procesamiento que ocurra

dentro de los componentes **B**, **C** y **D**, el componente **A** no puede programarse para ser probado hasta que los componentes **B**, **C** y **D** no hayan sido probados por completo. La programación del proyecto reflejaría esto. Pero si el tiempo es demasiado escaso y debe disponerse de **A** para realizar pruebas de importancia crítica, puede decidirse a probar **A** sin sus componentes subordinados (que están un poco retrasados) a fin de que esté disponible para otras pruebas que se realicen antes de la entrega. Después de todo, el plazo final se acerca. En consecuencia, **A** podría tener defectos ocultos que sólo se descubrirían mucho tiempo después. La calidad bajaría.

Decisiones orientadas al riesgo. La administración del riesgo (véase el capítulo 28) es uno de los atributos clave de un proyecto exitoso de software. En realidad se necesita saber lo que puede salir mal y establecer un plan de contingencia para ese caso. Demasiados equipos de software prefieren un optimismo ciego y establecen un programa de desarrollo con la suposición de que nada saldrá mal. Lo que es peor, no tienen manera de manejar las cosas que salgan mal. En consecuencia, cuando un riesgo se convierte en realidad, reina el caos y aumenta el grado de locuras que se cometen, con lo que invariablemente la calidad se desploma.

El dilema de la calidad del software se resume mejor con el enunciado de la Ley de Meskimen: *Nunca hay tiempo para hacerlo bien, pero siempre hay tiempo para hacerlo otra vez.* Mi consejo es: tomarse el tiempo para hacerlo bien casi nunca es la decisión equivocada.

14.4 Lograr la calidad del software

La calidad del software no sólo se ve. Es el resultado de la buena administración del proyecto y de una correcta práctica de la ingeniería de software. La administración y práctica se aplican en el contexto de cuatro actividades principales que ayudan al equipo de software a lograr una alta calidad en éste: métodos de la ingeniería de software, técnicas de administración de proyectos, acciones de control de calidad y aseguramiento de la calidad del software.

14.4.1 Métodos de la ingeniería de software

? ¿Qué necesito hacer para influir en la calidad de manera positiva?

Si espera construir software de alta calidad, debe entender el problema que se quiere resolver. También debe ser capaz de crear un diseño que esté de acuerdo con el problema y que al mismo tiempo tenga características que lleven al software a las dimensiones y factores de calidad que se estudiaron en la sección 14.2.

En la parte 2 de este libro se presentó una amplia variedad de conceptos y métodos que conducen a una comprensión razonablemente completa del problema y al diseño exhaustivo que establece un fundamento sólido para la actividad de construcción. Si el lector aplica estos conceptos y adopta métodos apropiados de análisis y diseño, se eleva sustancialmente la probabilidad de crear software de alta calidad.

14.4.2 Técnicas de administración de proyectos

El efecto de las malas decisiones de administración sobre la calidad del software se estudió en la sección 14.3.6. Las implicaciones son claras: si 1) un gerente de proyecto usa estimaciones para verificar que las fechas pueden cumplirse, 2) se comprenden las dependencias de las actividades programadas y el equipo resiste la tentación de usar atajos, 3) la planeación del riesgo se lleva a cabo de manera que los problemas no alienten el caos, entonces la calidad del software se verá influida de manera positiva.

Además, el plan del proyecto debe incluir técnicas explícitas para la administración de la calidad y el cambio. Las técnicas que llevan a buenas prácticas de administración de proyectos se estudian en la parte 4 de este libro.

14.4.3 Control de calidad

? ¿Qué es el control de calidad del software?

El control de calidad incluye un conjunto de acciones de ingeniería de software que ayudan a asegurar que todo producto del trabajo cumpla sus metas de calidad. Los modelos se revisan para garantizar que están completos y que son consistentes. El código se inspecciona con objeto de descubrir y corregir errores antes de que comiencen las pruebas. Se aplica una serie de etapas de prueba para detectar los errores en procesamiento lógico, manipulación de datos y comunicación con la interfaz. La combinación de mediciones con retroalimentación permite que el equipo del software sintonice el proceso cuando cualquiera de estos productos del trabajo falla en el cumplimiento de las metas de calidad. Las actividades de control de calidad se estudian en detalle en lo que resta de la parte 3 de este libro.

14.4.4 Aseguramiento de la calidad

WebRef

Pueden encontrarse vínculos útiles acerca de técnicas de aseguramiento de la calidad en la dirección **www.niwotridge.com/Resources/PM-SWEResources/SoftwareQualityAssurance.htm**

El aseguramiento de la calidad establece la infraestructura de apoyo a los métodos sólidos de la ingeniería de software, la administración racional de proyectos y las acciones de control de calidad, todo de importancia crucial si se trata de elaborar software de alta calidad. Además, el aseguramiento de la calidad consiste en un conjunto de funciones de auditoría y reportes para evaluar la eficacia y completitud de las acciones de control de calidad. La meta del aseguramiento de la calidad es proveer al equipo administrativo y técnico los datos necesarios para mantenerlo informado sobre la calidad del producto, con lo que obtiene perspectiva y confianza en que las acciones necesarias para lograr la calidad del producto funcionan. Por supuesto, si los datos provistos a través del aseguramiento de la calidad identifican los problemas, es responsabilidad de la administración enfrentarlos y aplicar los recursos necesarios para resolver los correspondientes a la calidad. En el capítulo 16 se estudia en detalle el aseguramiento de la calidad del software.

14.5 Resumen

La preocupación por la calidad de los sistemas basados en software ha aumentado a medida que éste se integra en cada aspecto de nuestras vidas cotidianas. Pero es difícil hacer la descripción exhaustiva de la calidad del software. En este capítulo se define la calidad como un proceso eficaz del software aplicado de modo que crea un producto útil que da un valor medible a quienes lo generan y a quienes lo utilizan.

Con el tiempo se han propuesto varias dimensiones y factores de calidad del software. Todos ellos tratan de definir un conjunto de características que, si se logran, llevarán a un software de alta calidad. McCall y los factores de calidad de la norma ISO 9126 establecen características tales como confiabilidad, usabilidad, facilidad de dar mantenimiento, funcionalidad y portabilidad, como indicadores de la existencia de calidad.

Toda organización de software se enfrenta al dilema de la calidad del software. En esencia, todos quieren elaborar sistemas de alta calidad, pero en un mundo dirigido por el mercado, sencillamente no se dispone del tiempo y el esfuerzo requeridos para producir software "perfecto". La cuestión es la siguiente: ¿debe elaborarse software que sea "suficientemente bueno"? Aunque muchas compañías hacen eso, hay una desventaja notable que debe tomarse en cuenta.

Sin importar el enfoque que se elija, la calidad tiene un costo que puede estudiarse en términos de prevención, evaluación y falla. Los costos de prevención incluyen todas las acciones de la ingeniería de software diseñadas para prevenir los defectos. Los costos de evaluación están asociados con aquellas acciones que evalúan los productos del trabajo de software para determinar su calidad. Los costos de falla incluyen el precio interno de fallar y los efectos externos que precipitan la mala calidad.

La calidad del software se consigue por medio de la aplicación de métodos de ingeniería de software, prácticas adecuadas de administración y un control de calidad exhaustivo, todo lo cual es apoyado por la infraestructura de aseguramiento de la calidad. En los capítulos que siguen se estudian con cierto detalle el control y aseguramiento de la calidad.

Problemas y puntos por evaluar

14.1. Describa cómo evaluaría la calidad de una universidad antes de inscribirse. ¿Cuáles factores serían importantes? ¿Cuáles tendrían importancia crítica?

14.2. Garvin [Gar84] describe cinco puntos de vista distintos sobre la calidad. Dé un ejemplo de cada uno con el uso de uno o más productos electrónicos conocidos con los que esté familiarizado.

14.3. Con el uso de la definición de calidad del software propuesta en la sección 14.2, diga si cree posible crear un producto útil que genere valor medible sin el uso de un proceso eficaz. Explique su respuesta.

14.4. Agregue dos preguntas adicionales a cada una de las dimensiones de la calidad de Garvin presentadas en la sección 14.2.1.

14.5. Los factores de calidad de McCall se desarrollaron en la década de 1970. Casi todos los aspectos de la computación han cambiado mucho desde entonces, no obstante lo cual aún se aplican al software moderno. ¿Qué conclusiones saca con base en ello?

14.6. Con el empleo de los subatributos mencionados en la sección 14.2.3 para el factor de calidad llamado "facilidad de recibir mantenimiento", de la ISO 9126, desarrolle preguntas que exploren si estos atributos existen o no. Continúe el ejemplo presentado en la sección 14.2.4.

14.7. Describa con sus propias palabras el dilema de la calidad del software.

14.8. ¿Qué es un software "suficientemente bueno"? Mencione una compañía dada y productos específicos que crea que fueron desarrollados con el uso de la filosofía de lo suficientemente bueno.

14.9. Considere cada uno de los cuatro aspectos de la calidad y diga cuál piensa que es el más caro y por qué.

14.10. Haga una búsqueda en web y encuentre otros tres ejemplos de "riesgos" para el público que puedan atribuirse directamente a la mala calidad de un software. Comience la búsqueda en **http://catless.ncl.ac.uk/risks.**

14.11. ¿Son lo mismo *calidad* y *seguridad*? Explique su respuesta.

14.12. Explique por qué es que muchos de nosotros utilizamos la ley de Meskimen. ¿Qué ocurre con el software de negocios que causa esto?

Lecturas y fuentes de información adicionales

Los conceptos básicos de calidad del software se estudian en los libros de Henry y Hanlon (*Software Quality Assurance*, Prentice-Hall, 2008), Kahn *et al.* (*Software Quality: Concepts and Practice*, Alpha Science International, Ltd., 2006), O'Regan (*A Practical Approach to Software Quality*, Springer, 2002) y Daughtrey (*Fundamental Concepts for the Software Quality Engineer*, ASQ Quality Press, 2001).

Duvall *et al.* (*Continuous Integration: Improving Software Quality and Reducing Risk*, Addison-Wesley, 2007), Tian (*Software Quality Engineering*, Wiley-IEEE Computer Society Press, 2005), Kandt (*Software Engineering Quality Practices*, Auerbach, 2005), Godbole (*Software Quality Assurance: Principles and Practice*, Alpha Science International, Ltd., 2004) y Galin (*Software Quality Assurance: From Theory to Implementation*, Addison-Wesley, 2003) presentan estudios detallados del aseguramiento de la calidad del software. Stamelos y Sfetsos (*Agile Software Development Quality Assurance*, IGI Global, 2007) estudian el aseguramiento de la calidad en el contexto del proceso ágil.

El diseño sólido conduce a una alta calidad del software. Jayasawal y Patton (*Design for Trustworthy Software*, Prentice-Hall, 2006) y Ploesch (*Contracts, Scenarios and Prototypes*, Springer, 2004) analizan las herramientas y técnicas para desarrollar software "robusto".

La medición es un componente importante de la ingeniería de calidad del software. Ejiogu (*Software Metrics: The Discipline of Software Quality*, BookSurge Publishing, 2005), Kan (*Metrics and Models in Software*

Quality Engineering, Addison-Wesley, 2002) y Nance y Arthur (*Managing Software Quality*, Springer, 2002) estudian unidades de medida y modelos importantes relacionados con la calidad. Los aspectos de la calidad del software orientados al equipo los estudia Evans (*Achieving Software Quality through Teamwork*, Artech House Publishers, 2004).

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