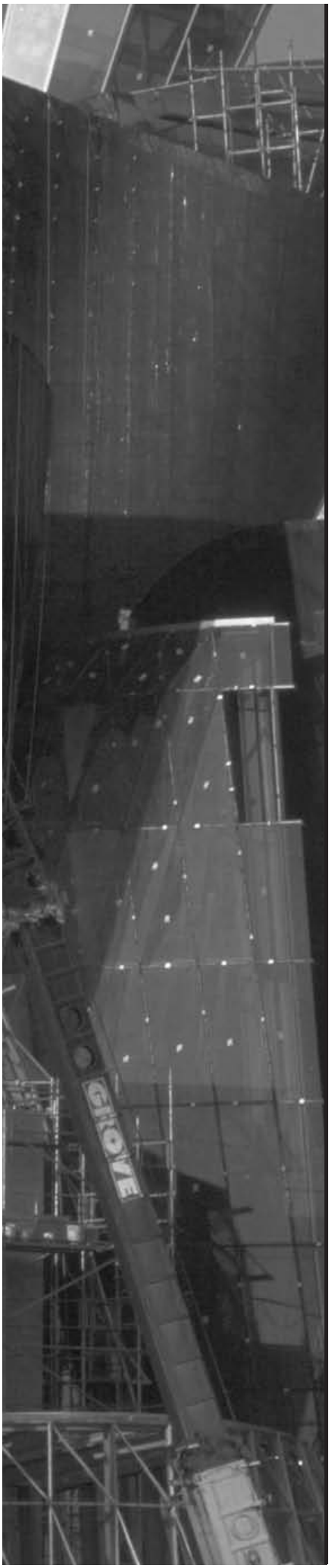

24

Gestión de la calidad

Objetivos

El objetivo de este capítulo es introducirlo a la gestión de calidad y la medición del software. Al estudiar este capítulo:

- estará al tanto del proceso de gestión de calidad y sabrá por qué es importante la planeación de calidad;
- comprenderá que la calidad del software se ve afectada por el proceso de desarrollo del software utilizado;
- conocerá la importancia de los estándares en el proceso de gestión de calidad y aprenderá cómo se usan los estándares en el aseguramiento de calidad;
- distinguirá la forma en que se utilizan las revisiones e inspecciones como mecanismo para garantizar la calidad del software;
- identificará cómo pueden ser útiles las mediciones en la valoración de algunos atributos de calidad del software y las limitaciones actuales de medición del software.

Contenido

Los problemas de calidad del software se descubrieron inicialmente en la década de 1960 con el desarrollo de los primeros grandes sistemas de software, y han continuado invadiendo la ingeniería de software a partir de esa década. El software entregado era lento y poco fiable, difícil de mantener y de reutilizar. El descontento con esta situación condujo a la adopción de técnicas formales de gestión de calidad del software, desarrolladas a partir de métodos usados en la industria manufacturera. Estas técnicas de gestión de calidad, en conjunto con nuevas tecnologías y mejores pruebas de software, llevaron a progresos significativos en el nivel general de calidad del software.

La gestión de calidad del software para los sistemas de software tiene tres intereses fundamentales:

1. A nivel de organización, la gestión de calidad se ocupa de establecer un marco de proceso y estándares de organización que conducirán a software de mejor calidad. Esto supone que el equipo de gestión de calidad debe tener la responsabilidad de definir los procesos de desarrollo del software a usar, los estándares que deben aplicarse al software y la documentación relacionada, incluyendo los requerimientos, el diseño y el código del sistema.
2. A nivel del proyecto, la gestión de calidad implica la aplicación de procesos específicos de calidad y la verificación de que continúen dichos procesos planeados; además, se ocupa de garantizar que los resultados del proyecto estén en conformidad con los estándares aplicables a dicho proyecto.
3. A nivel del proyecto, la gestión de calidad se ocupa también de establecer un plan de calidad para un proyecto. El plan de calidad debe establecer metas de calidad para el proyecto y definir cuáles procesos y estándares se usarán.

Los términos *aseguramiento de calidad* y *control de calidad* se utilizan ampliamente en la industria manufacturera. El aseguramiento de calidad (QA, por las siglas de *quality assurance*) es la definición de procesos y estándares que deben conducir a la obtención de productos de alta calidad y, en el proceso de fabricación, a la introducción de procesos de calidad. El control de calidad es la aplicación de dichos procesos de calidad para eliminar aquellos productos que no cuentan con el nivel requerido de calidad.

En la industria de software, diversas compañías y sectores industriales interpretan de maneras diferentes el aseguramiento de calidad y el control de calidad. En ocasiones, el aseguramiento de calidad representa simplemente la definición de procedimientos, procesos y estándares cuyo objetivo es asegurar el logro de calidad del software. En otros casos, el aseguramiento de calidad incluye también todas las actividades de gestión de configuración, verificación y validación aplicadas después de que un equipo de desarrollo entrega un producto. En este capítulo se usa el término *aseguramiento de calidad* para incluir verificación y validación, y los procesos de que la comprobación de procedimientos de calidad se aplicó de manera adecuada. Se evita el término "control de calidad", puesto que esta expresión no se usa mucho en la industria del software.

En la mayoría de las compañías, el equipo QA es el responsable de administrar el proceso de pruebas de liberación. Como se explicó en el capítulo 8, esto significa que se aplican las pruebas del software antes de que éste se libere a los clientes. El equipo es responsable de comprobar que las pruebas del sistema cubran los requerimientos y de mantener los registros adecuados del proceso de pruebas. Como en el capítulo 8 se estudiaron las pruebas de liberación, en este apartado no se trata este aspecto del aseguramiento de calidad.

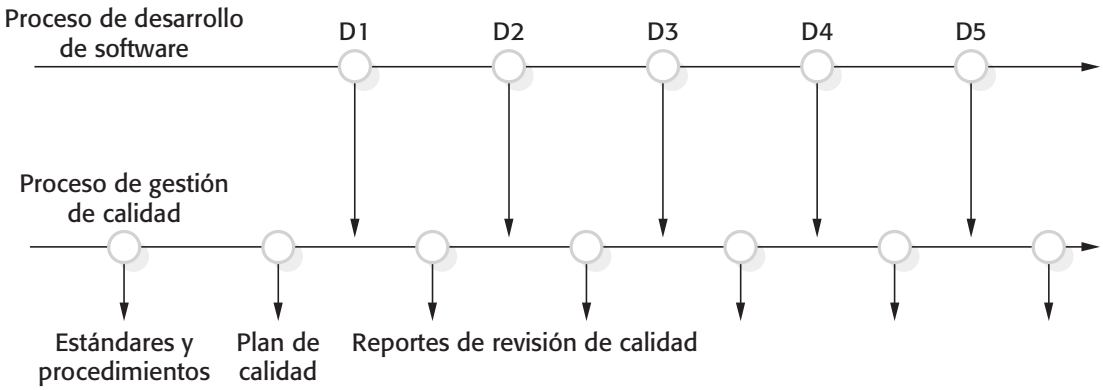

Figura 24.1
Gestión de calidad y desarrollo de software

La gestión de calidad proporciona una comprobación independiente sobre el proceso de desarrollo de software. El proceso de gestión de calidad verifica los entregables del proyecto para garantizar que sean consistentes con los estándares y las metas de la organización (figura 24.1). El equipo QA debe ser independiente del equipo de desarrollo para que pueda tener una perspectiva objetiva del software. Esto les permite reportar la calidad del software sin estar influidos por los conflictos de desarrollo del software.

De preferencia, el equipo de gestión de calidad no debe asociarse con algún grupo de desarrollo particular; sin embargo, tiene la responsabilidad ante toda la organización por la administración de la calidad. El equipo debe ser independiente y reportarse ante la administración ubicada sobre el nivel del administrador del proyecto. La razón es que los administradores de proyecto tienen que mantener el presupuesto y el calendario del proyecto. Si surgen problemas, pueden estar tentados a comprometer la calidad del producto para cumplir con el calendario. Un equipo de gestión de calidad independiente garantiza que las metas de calidad de la organización no estén comprometidas a corto plazo por consideraciones de presupuesto y calendario. Sin embargo, en compañías más pequeñas, esto es prácticamente imposible. La gestión de calidad y el desarrollo de software están inevitablemente vinculados con las personas que tienen responsabilidades tanto de desarrollo como de calidad.

La planeación de calidad es el proceso de desarrollar un plan de calidad para un proyecto. El plan de calidad debe establecer las cualidades deseadas de software y describir cómo se valorarán. Por lo tanto, define lo que realmente significa software de "alta calidad" para un sistema particular. Sin esta definición, los ingenieros pueden hacer diferentes suposiciones, algunas veces conflictivas, sobre cuáles atributos del producto reflejan las características de calidad más importantes. La planeación de calidad formalizada es parte integral de los procesos de desarrollo basados en un plan. No obstante, los métodos ágiles adoptan un enfoque menos formal para la gestión de calidad.

Humphrey (1989), en su clásico libro referente a la gestión del software, sugiere un bosquejo de estructura para un plan de calidad. Éste incluye:

1. *Introducción del producto* Una descripción del producto, la pretensión de su mercado y las expectativas de calidad para el producto.
2. *Planes del producto* Indican las fechas de entrega críticas y las responsabilidades para el producto, junto con planes para distribución y servicio al producto.

3. *Descripciones de procesos* Describen los procesos y estándares de desarrollo y servicio que deben usarse para diseño y gestión del producto.
4. *Metas de calidad* Las metas y los planes de calidad para el producto, incluyendo una identificación y justificación de los atributos esenciales de calidad del producto.
5. *Riesgos y gestión del riesgo* Los riesgos clave que pueden afectar la calidad del producto y las acciones a tomar para enfrentar dichos riesgos.

Los planes de calidad, que se desarrollan como parte del proceso de planeación general del proyecto, difieren en detalle dependiendo del tamaño y tipo de sistema que se desarrolló. Sin embargo, cuando escriba planes de calidad, debe tratar de mantenerlos tan breves como sea posible. Si el documento es demasiado amplio, las personas no lo leerán y, en consecuencia, se anulará el propósito de generar un plan de calidad.

Algunas personas consideran que la calidad del software puede lograrse mediante procesos establecidos basados en estándares de organización y procedimientos de calidad asociados que verifican el seguimiento de dichos estándares mediante el equipo de desarrollo de software. Su argumento es que los estándares se dirigen a la buena práctica de ingeniería de software y que seguir esta buena práctica conducirá a productos de alta calidad. No obstante, en la práctica, se considera que en la gestión de calidad hay mucho más que estándares y burocracia asociados para asegurar que éstos se sigan.

Aunque los estándares y procesos son importantes, los administradores de calidad deben enfocarse también a desarrollar una "cultura de calidad" en la que todo responsable del desarrollo del software se comprometa a lograr un alto nivel de calidad del producto. Deben exhortar a los equipos a asumir la responsabilidad de la calidad de su trabajo y desarrollar nuevos enfoques para el mejoramiento de la calidad. A pesar de que los estándares y procedimientos son la base de la gestión de calidad, los buenos administradores de calidad reconocen que existen aspectos intangibles a la calidad del software (elegancia, legibilidad, etcétera) que no pueden expresarse en estándares. Deben apoyar a la gente interesada en aspectos intangibles de calidad e impulsar el comportamiento profesional en todos los miembros del equipo.

La gestión de la calidad formalizada es particularmente importante para los equipos que diseñan grandes sistemas de larga duración, los cuales tardan varios años en desarrollarse. La documentación de la calidad es un registro de lo que cada subgrupo realizó en el proyecto. Ayuda a las personas a comprobar que no se olvidaron tareas importantes o que un grupo no hizo suposiciones incorrectas acerca de lo que hicieron otros grupos. La documentación de la calidad también es un medio de comunicación durante la vida del sistema. Permite a los grupos responsables de la evolución del sistema encontrar las pruebas y comprobaciones que el equipo de desarrollo debe implementar.

Para sistemas más pequeños, la gestión de calidad es aún importante, aunque puede adoptarse un enfoque más informal. No se necesita tanto papeleo, puesto que un equipo de desarrollo pequeño puede comunicarse de manera informal. El aspecto de calidad clave para el desarrollo de sistemas pequeños es establecer una cultura de calidad y asegurarse de que todos los miembros del equipo tienen un enfoque positivo sobre la calidad del software.

24.1 Calidad del software

La industria manufacturera estableció los fundamentos de la gestión de calidad para mejorar ésta en los productos que se fabricaban. Como parte de ello se desarrolló una definición de calidad, que se basa en la conformidad con una especificación de producto detallada (Crosby, 1979) y la noción de tolerancia. La suposición subyacente era que los productos podían especificarse por completo y establecerse procedimientos que comprobaran si un producto manufacturado cumplía o no con su especificación. Desde luego, los productos nunca cumplirán exactamente una especificación, pues se permite cierta tolerancia. Si el producto era "casi bueno", se clasificaba como aceptable.

La calidad del software no es directamente comparable con la calidad en la fabricación. La idea de tolerancia no es aplicable a los sistemas digitales y es prácticamente imposible llegar a una conclusión objetiva sobre si un sistema de software cumple o no su especificación, por las siguientes razones:

1. Como se explicó en el capítulo 4, referente a la ingeniería de requerimientos, es difícil escribir especificaciones de software completas y sin ambigüedades. Los desarrolladores y clientes de software pueden interpretar los requerimientos de diferentes formas y tal vez sea imposible llegar a acuerdos acerca de si el software se desarrolló conforme a su especificación.
2. Por lo general, las especificaciones integran requerimientos de varias clases de participantes. Dichos requerimientos son un compromiso ineludible y tal vez no incluyan los requerimientos de todos los grupos de participantes. Por lo tanto, las partes interesadas excluidas quizá perciban el sistema como uno de mala calidad, a pesar de que implementa los requerimientos acordados.
3. Es imposible medir de manera directa ciertas características de calidad (por ejemplo, mantenibilidad) y, por ende, no pueden especificarse plenamente sin ambigüedades. En la sección 24.4 se estudian las dificultades de la medición.

Debido a estos problemas, la valoración de calidad del software es un proceso subjetivo en que el equipo de gestión de calidad tiene que usar su juicio para decidir si se logró un nivel aceptable de calidad. El equipo de gestión de calidad debe considerar si el software se ajusta o no a su propósito pretendido. Esto implica responder preguntas sobre las características del sistema. Por ejemplo:

1. ¿En el proceso de desarrollo se siguieron los estándares de programación y documentación?
2. ¿El software se verificó de manera adecuada?
3. ¿El software es suficientemente confiable para utilizarse?
4. ¿El rendimiento del software es aceptable para uso normal?

Protección	Comprensibilidad	Portabilidad
Seguridad	Comprobabilidad	Usabilidad
Fiabilidad	Adaptabilidad	Reusabilidad
Flexibilidad	Modularidad	Eficiencia
Robustez	Complejidad	Facilidad para que el usuario aprenda a utilizarlo

Figura 24.2 Atributos de calidad del software

5. ¿El software es utilizable?
6. ¿El software está bien estructurado y es comprensible?

Existe la suposición general en la gestión de calidad del software de que el sistema se pondrá a prueba en contra de sus requerimientos. La decisión acerca de si entregar o no la funcionalidad requerida debe basarse en los resultados de dichas pruebas. Por lo tanto, el equipo QA debe revisar las pruebas que se desarrollaron y examinar los registros de pruebas para verificar que éstas se hayan realizado de manera apropiada. En algunas organizaciones el equipo de gestión de calidad es responsable de las pruebas del sistema, pero, en ocasiones, un grupo de pruebas de sistema separado es responsable de esto.

La calidad subjetiva de un sistema de software se basa principalmente en sus características no funcionales. Esto refleja la experiencia práctica del usuario: Si la funcionalidad del software no es lo que se esperaba, entonces los usuarios con frecuencia sólo le darán la vuelta a este asunto y encontrarán otras formas de hacer lo que quieren. Sin embargo, si el software no es fiable o resulta muy lento, entonces es prácticamente imposible que los usuarios logren sus metas.

Por consiguiente, la calidad del software no sólo se trata de si la funcionalidad de éste se implementó correctamente, sino también depende de los atributos no funcionales del sistema. Boehm y sus colaboradores (1978) indican que existen 15 importantes atributos de calidad de software, los cuales se listan en la figura 24.2. Dichos atributos se relacionan con la confiabilidad, usabilidad, eficiencia y mantenibilidad del software. Como se estudió en el capítulo 11, por lo general se considera que los atributos de confiabilidad son los atributos de calidad más importantes de un sistema. Sin embargo, también es significativo el rendimiento del software. Los usuarios rechazarán el software que sea demasiado lento.

No es posible que algún sistema se optimice para todos esos atributos; por ejemplo, mejorar la robustez puede conducir a pérdida de rendimiento. En consecuencia, el plan de calidad debe definir los atributos de calidad más importantes para el software que se desarrollará. Tal vez la eficiencia sea crítica y tengan que sacrificarse otros factores para que se logre esto. Si lo anterior se estableció en el plan de calidad, los ingenieros que trabajan en el desarrollo pueden cooperar para lograrlo. El plan debe incluir también una definición del proceso de valoración de la calidad. Ésta debe ser una forma acordada de valorar si cierto grado de calidad, como la mantenibilidad o robustez, está presente en el producto.

Una suposición que subyace en la gestión de la calidad del software es que la calidad del software se relaciona directamente con la calidad del proceso de desarrollo del

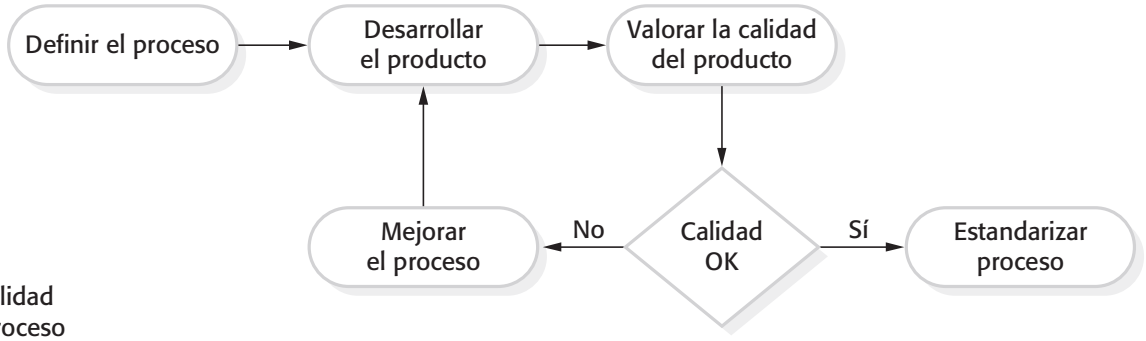

Figura 24.3 Calidad basada en el proceso

software. Esto proviene de nuevo de los sistemas fabriles, donde la calidad del producto está estrechamente relacionada con el proceso de producción. Un proceso de fabricación incluye configurar, establecer y operar las máquinas implicadas en el proceso. Una vez que las máquinas operan correctamente, se sigue de manera natural la calidad del producto. Entonces se mide la calidad del producto y el proceso se modifica hasta que se logra el nivel de calidad necesario. La figura 24.3 ilustra este enfoque basado en el proceso para obtener la calidad del producto.

En la manufactura existe un claro vínculo entre el proceso y la calidad del producto, ya que el proceso es relativamente sencillo de estandarizar y monitorizar. Una vez calibrados los sistemas de fabricación, pueden operar una y otra vez para generar productos de alta calidad; sin embargo, el software no se manufactura, se diseña. Por lo tanto, en el desarrollo del software es más compleja la relación entre calidad de proceso y calidad del producto. El diseño del software es un proceso creativo más que mecánico, pues es significativa la influencia de las habilidades y la experiencia individuales. Factores externos, como la novedad de una aplicación o la premura por el lanzamiento comercial de un producto, también afectan la calidad de éste sin importar el proceso usado.

No hay duda de que el proceso de desarrollo utilizado tiene una influencia importante sobre la calidad del software, y que los buenos procesos tienen más probabilidad de conducir a software de buena calidad. La gestión de la calidad y el mejoramiento del proceso pueden conducir a menores defectos en el software a desarrollar. Sin embargo, es difícil valorar los atributos de calidad del software, como la mantenibilidad, sin usar el software durante un largo periodo. En consecuencia, es difícil decir cómo las características del proceso influyen en dichos atributos. Más aún, debido al papel del diseño y la creatividad en el proceso de software, la estandarización del proceso en ocasiones puede exterminar la creatividad, lo cual, lejos de elevar la calidad, conducirá a un software de calidad inferior.

24.2 Estándares de software

Los estándares de software tienen una función muy importante en la gestión de calidad del software. Como se indicó, un aspecto importante del aseguramiento de calidad es la definición o selección de estándares que deben aplicarse al proceso de desarrollo de software o al producto de software. Como parte de este proceso QA, también pueden elegirse herramientas y métodos para apoyar el uso de dichos estándares. Una vez seleccionados éstos

Estándares de documentación

Los documentos del proyecto son una forma tangible de describir las diferentes representaciones de un sistema de software (requerimientos, UML, código, etcétera) y su proceso de producción. Los estándares de documentación definen la organización de diferentes tipos de documentos, así como el formato del documento. Son importantes porque facilitan la comprobación de que no se haya omitido material importante de los documentos y garantiza que los documentos del proyecto tengan una apariencia común. Los estándares pueden desarrollarse para el proceso de escribir documentos, los documentos en sí y el intercambio de documentos.

http://www.SoftwareEngineering-9.com/Web/QualityMan/docstandards.html

para su uso, deben definirse procesos específicos de proyecto para monitorizar el uso de los estándares y comprobar que éstos se siguieron.

Los estándares de software son importantes por tres razones:

1. Los estándares reflejan la sabiduría que es de valor para la organización. Se basan en conocimiento sobre la mejor o más adecuada práctica para la compañía. Con frecuencia, este conocimiento se adquiere sólo después de gran cantidad de ensayo y error. Configurarla dentro de un estándar, ayuda a la compañía a reutilizar esta experiencia y a evitar errores del pasado.
2. Los estándares proporcionan un marco para definir, en un escenario particular, lo que significa el término "calidad". Como se dijo, la calidad del software es subjetiva, y al usar estándares se establece una base para decidir si se logró un nivel de calidad requerido. Desde luego, esto depende del establecimiento de estándares que reflejen las expectativas del usuario para la confiabilidad, la usabilidad y el rendimiento del software.
3. Los estándares auxilian la continuidad cuando una persona retoma el trabajo iniciado por alguien más. Los estándares aseguran que todos los ingenieros dentro de una organización adopten las mismas prácticas. En consecuencia, se reduce el esfuerzo de aprendizaje requerido al iniciarse un nuevo trabajo.

Existen dos tipos de estándares de ingeniería de software relacionados que pueden definirse y usarse en la gestión de calidad del software:

1. *Estándares del producto* Se aplican al producto de software a desarrollar. Incluyen estándares de documentos (como la estructura de los documentos de requerimientos), estándares de documentación (como el encabezado de un comentario estándar para una definición de clase de objeto) y estándares de codificación, los cuales definen cómo debe usarse un lenguaje de programación.
2. *Estándares de proceso* Establecen los procesos que deben seguirse durante el desarrollo del software. Deben especificar cómo es una buena práctica de desarrollo. Los estándares de proceso pueden incluir definiciones de especificación, procesos de diseño y validación, herramientas de soporte de proceso y una descripción de los documentos que deben escribirse durante dichos procesos.

Estándares de producto	Estándares de proceso
Formato de revisión de diseño	Realizar revisión de diseño
Estructura de documento de requerimientos	Enviar nuevo código para construcción de sistema
Formato de encabezado por método	Proceso de liberación de versión
Estilo de programación Java	Proceso de aprobación del plan del proyecto
Formato de plan de proyecto	Proceso de control de cambio
Formato de solicitud de cambio	Proceso de registro de prueba

Figura 24.4 Estándares de producto y proceso

Los estándares deben entregar valor, en la forma de calidad aumentada del producto. No hay razón para definir estándares que sean costosos en términos de tiempo y esfuerzo, pues aplicarlos sólo conduce a mejoras secundarias en la calidad. Los estándares de producto deben diseñarse de forma que puedan aplicarse y comprobarse de manera efectiva en cuanto a costos, y los estándares de proceso deben incluir la definición de procesos que comprueben que se siguieron dichos estándares.

El desarrollo de estándares internacionales de ingeniería de software, por lo general, es un proceso prolongado en el que se reúnen los interesados en el estándar, elaboran borradores para comentar y, finalmente, acuerdan el estándar. Organismos nacionales e internacionales, como U.S. DoD, ANSI, BSI, OTAN y el IEEE, apoyan la determinación de estándares. Se trata de estándares generales que pueden aplicarse a través de varios proyectos. Entidades tales como la OTAN y otras organizaciones de defensa pueden requerir que sus propios estándares se usen en el desarrollo de contratos que suscriben con compañías de software.

Se han desarrollado estándares nacionales e internacionales que incluyen la terminología de ingeniería de software, lenguajes de programación como Java y C++, anotaciones como los símbolos de diagramación, procedimientos para derivar y escribir requerimientos de software, procedimientos de aseguramiento de calidad, y procesos de verificación y validación de software (IEEE, 2003). Estándares más especializados, como IEC 61508 (IEC, 1998), se desarrollaron para sistemas críticos de protección y seguridad.

Los equipos de gestión de calidad que elaboran estándares para alguna compañía, por lo general deben basar los estándares de dicha compañía en estándares nacionales e internacionales. Al usar estándares internacionales como punto de partida, el equipo de aseguramiento de calidad debe redactar un manual de estándares, el cual debe definir los estándares que necesita su organización. En la figura 24.4 se muestran ejemplos de estándares que podrían incluirse en dicho manual.

En ocasiones, los ingenieros de software consideran los estándares como demasiado prescriptivos y realmente poco relevantes para la actividad técnica del desarrollo de software. Esto es probable sobre todo cuando los estándares de proyecto requieren documentación y registro del trabajo tediosos. Aunque en general están de acuerdo con la necesidad de los estándares, los ingenieros encuentran a menudo razones para señalar que los estándares no necesariamente son adecuados para su proyecto particular. Para

minimizar el descontento y alentar la participación en los estándares, los administradores de calidad que establezcan los estándares deben dar los siguientes pasos:

1. *Involucrar a los ingenieros de software en la selección de estándares de producto* Si los desarrolladores comprenden por qué se seleccionaron los estándares, tienen más probabilidad de comprometerse con éstos. De preferencia, los documentos de estándares no deben establecer sólo el estándar a seguir, sino también deben incluir comentarios que expliquen por qué se tomaron las decisiones de estandarización.
2. *Revisar y modificar regularmente los estándares para reflejar las tecnologías cambiantes* Los estándares son costosos de desarrollar y tienden a guardarse como reliquias en un manual de estándares de una compañía. Debido a los costos y la discusión requeridos, muchas veces hay reticencia para cambiarlos. Aunque un manual de estándares es esencial, debe evolucionar para reflejar las circunstancias y la tecnología cambiantes.
3. *Ofrecer herramientas de software para dar soporte a los estándares* Los desarrolladores encuentran con frecuencia que los estándares son una pesadilla cuando la adhesión a ellos incluye un tedioso trabajo manual que podría hacerse mediante una herramienta de software. Si está disponible el soporte para herramientas, se requiere muy poco esfuerzo para seguir los estándares de desarrollo de software. Por ejemplo, los estándares de documento pueden implementarse mediante estilos de procesador de texto.

Diferentes tipos de software necesitan distintos procesos de desarrollo, puesto que los estándares deben ser adaptables. No hay razón para prescribir una forma particular de trabajar si es inadecuada para un proyecto o equipo de proyecto. Cada administrador de proyecto debe tener la autoridad de modificar los estándares de proceso de acuerdo con las circunstancias individuales. Sin embargo, cuando se hacen cambios, es importante garantizar que dichos cambios no conduzcan a una pérdida de calidad del producto. Esto afectará la relación de una empresa con sus clientes y conducirá probablemente a un aumento en los costos del proyecto.

El administrador del proyecto y el administrador de calidad pueden evitar problemas en los estándares al planear cuidadosamente la calidad oportuna en el proyecto. Deben decidir cuál de los estándares de la organización debe usarse sin cambio, cuáles deben modificarse y cuáles ignorarse. Es posible que deban crearse nuevos estándares en respuesta a requerimientos del cliente o del proyecto. Por ejemplo, tal vez se requieran estándares para especificaciones formales si no se han usado en proyectos anteriores.

24.2.1 El marco de estándares ISO 9001

Existe un conjunto internacional de estándares que pueden utilizarse en el desarrollo de los sistemas de administración de calidad en todas las industrias, llamado ISO 9000. Los estándares ISO 9000 pueden aplicarse a varias organizaciones, desde las industrias manufactureras hasta las de servicios. ISO 9001, el más general de dichos estándares, se aplica a organizaciones que diseñan, desarrollan y mantienen productos, incluido software. El estándar ISO 9001 se desarrolló originalmente en 1987, y su revisión más reciente fue en 2008.

Figura 24.5 Procesos centrales ISO 9001

El estándar ISO 9001 no es en sí mismo un estándar para el desarrollo de software, sino un marco para elaborar estándares de software. Establece principios de calidad total, describe en general el proceso de calidad, y explica los estándares y procedimientos organizacionales que deben determinarse. Éstos tienen que documentarse en un manual de calidad de la organización.

La revisión principal del estándar ISO 9001 reorientó en 2000 el estándar hacia nueve procesos centrales (figura 24.5). Si una organización quiere estar conforme con el estándar ISO 9001, debe documentar cómo se relacionan sus procesos con dichos procesos centrales. También deberá definir y mantener registros que demuestren que se siguieron los procesos organizacionales establecidos. El manual de calidad de la compañía tiene que describir los procesos relevantes y los datos de proceso que deben recopilarse y conservarse.

El estándar ISO 9001 no define ni prescribe los procesos de calidad específicos que deben usarse en una compañía. Para estar de conformidad con ISO 9001, una compañía debe especificar los tipos de proceso que se muestran en la figura 24.5 y tener procedimientos que demuestren que se siguen sus procesos de calidad. Esto permite flexibilidad a través de sectores industriales y diversos tamaños de compañías. Pueden definirse estándares de calidad que sean adecuados para el tipo de software a desarrollar. Las compañías pequeñas pueden tener procesos no burocráticos y estar en conformidad con ISO 9001. Sin embargo, esta flexibilidad significa que no es posible hacer suposiciones sobre las similitudes o diferencias entre los procesos en distintas compañías que acatan ISO 9001. Algunas compañías tienen procesos de calidad muy rígidos con registros detallados, mientras que otras son mucho menos formales, con poca documentación adicional.

En la figura 24.6 se muestran las relaciones entre ISO 9001, manuales de calidad organizacional y planes de calidad de proyecto individuales. Este diagrama se derivó de un modelo presentado por Ince (1994), quien explica cómo puede usarse el estándar general ISO 9001 como base para procesos de gestión de calidad de software. Bamford y Dielbler (2003) explican cómo puede aplicarse el más reciente estándar ISO 9001:2000 en las compañías de software.

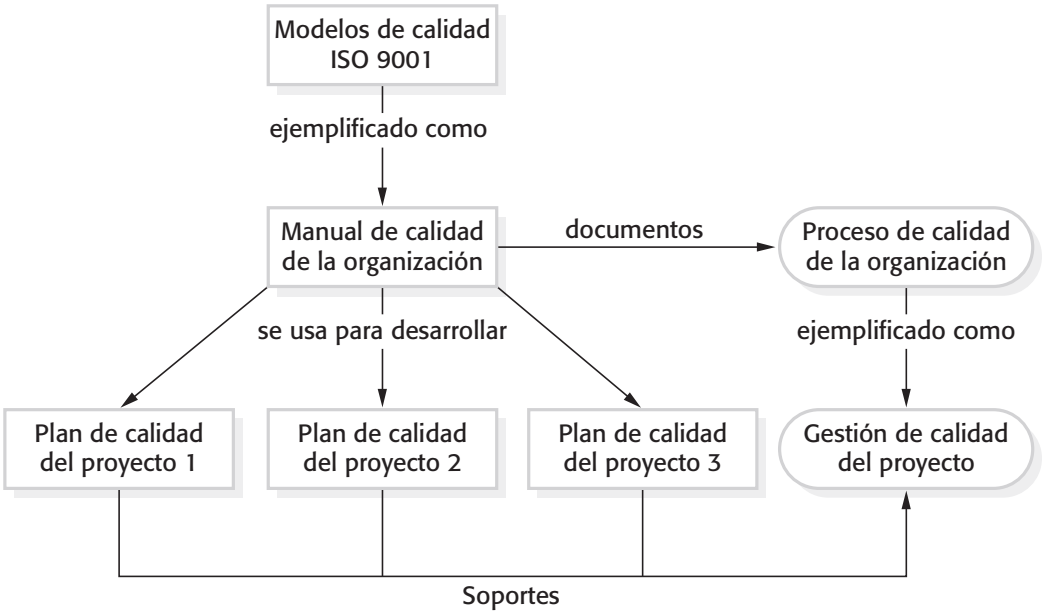

Figura 24.6
ISO 9001 y gestión de la calidad

Algunos clientes de software demandan que sus proveedores tengan la certificación ISO 9001. Así, los clientes podrán estar seguros de que la compañía que desarrolla el software tiene un sistema de gestión de calidad aprobado. Autoridades de acreditación independiente examinan los procesos de gestión de calidad y la documentación de proceso, y deciden si dichos procesos abarcan todas las áreas especificadas en ISO 9001. Si es así, certifican que los procesos de calidad de una compañía, definidos en el manual de calidad, concuerdan con el estándar ISO 9001.

Algunas personas consideran que la certificación ISO 9001 significa que la calidad del software producido por compañías certificadas será mejor que el derivado de compañías no certificadas. Esto no precisamente es cierto. El estándar ISO 9001 se enfoca en garantizar que la organización tenga procedimientos de gestión de calidad y que siga dichos procedimientos. No hay seguridad de que las compañías con certificación ISO 9001 empleen las mejores prácticas de desarrollo de software o que sus procesos conduzcan a software de alta calidad.

Por ejemplo, una compañía podría definir estándares de cobertura de pruebas que especifiquen que todos los métodos en los objetos deben llamarse al menos una vez. Lamentablemente, este estándar puede cumplirse mediante pruebas de software incompletas, que no incluyen pruebas con diferentes parámetros de métodos. En tanto se sigan los procedimientos de prueba definidos y se conserven registros de las pruebas realizadas, la compañía podría tener la certificación ISO 9001. Esta certificación define la calidad como la conformidad con estándares, y toma en cuenta la calidad como la advierten los usuarios del software.

Los métodos ágiles, que evitan la documentación y se enfocan en el código a desarrollar, tienen poco en común con los procesos de calidad formal que se examinan en ISO 9001. Se ha hecho cierto trabajo para reconciliar estos enfoques (Stalhane y Hanssen, 2008), pero la comunidad de desarrollo ágil en general se opone a lo que considera una carga burocrática de la conformidad con los estándares. Por esta razón, las compañías

que usan métodos de desarrollo ágil se preocupan pocas veces por la certificación ISO 9001.

24.3 Revisiones e inspecciones

Las revisiones e inspecciones son actividades QA que comprueban la calidad de los entregables del proyecto. Esto incluye examinar el software, su documentación y los registros del proceso para descubrir errores y omisiones, así como observar que se siguieron los estándares de calidad. Como se estudió en los capítulos 8 y 15, revisiones e inspecciones se usan junto con las pruebas del programa como parte del proceso general de verificación y validación del software.

Durante una revisión, un grupo de personas examinan el software y su documentación asociada en busca de problemas potenciales y la falta de conformidad con los estándares. El equipo de revisión realiza juicios informados sobre el nivel de calidad de un entregable de sistema o de proyecto. Entonces los administradores de proyecto pueden usar dichas valoraciones para tomar decisiones de planeación y asignar recursos al proceso de desarrollo.

Las revisiones de calidad se basan en documentos que se elaboraron durante el proceso de desarrollo del software. Al igual que las especificaciones, el diseño o el código del software, también pueden revisarse los modelos de proceso, planes de prueba, procedimientos de gestión de configuración, estándares de proceso y manuales de usuario. La revisión debe comprobar la coherencia e integridad de los documentos o el código objeto de prueba, y asegurarse de que se han seguido las normas de calidad.

Sin embargo, la revisión no sólo es acerca de la comprobación de conformidad con las normas, sino también se utiliza para ayudar a descubrir problemas y omisiones en la documentación del software o proyecto. Las conclusiones de la revisión deben registrarse formalmente como parte del proceso de gestión de calidad. Si se descubren problemas, los comentarios de los revisores deben pasar al autor del software o a quien resulte responsable de corregir los errores u omisiones.

El propósito de las revisiones e inspecciones es mejorar la calidad del software, no de valorar el rendimiento de los miembros del equipo de desarrollo. La revisión es un proceso público de detección de errores, comparado con el proceso más privado de prueba de componentes. Es necesario que los errores cometidos por los individuos se revelen a todo el equipo de programación. Para garantizar que todos los desarrolladores participen constructivamente con el proceso de revisión, los administradores de proyecto tienen que ser sensibles a las preocupaciones individuales. Deben desarrollar una cultura de trabajo que brinde apoyo y no culpar cuando se descubran errores.

Aunque una revisión de calidad ofrece a la administración datos sobre el software a desarrollar, las revisiones de calidad no son lo mismo que las revisiones de avance administrativo. Como se expuso en el capítulo 23, las revisiones de avance comparan el avance real en un proyecto de software frente al avance planeado. Su principal preocupación es si el proyecto entregará o no el software útil a tiempo y dentro del presupuesto. Las revisiones de progreso toman en cuenta factores externos, y circunstancias cambiantes pueden significar que el software en fase de desarrollo ya no se requiera o que tenga

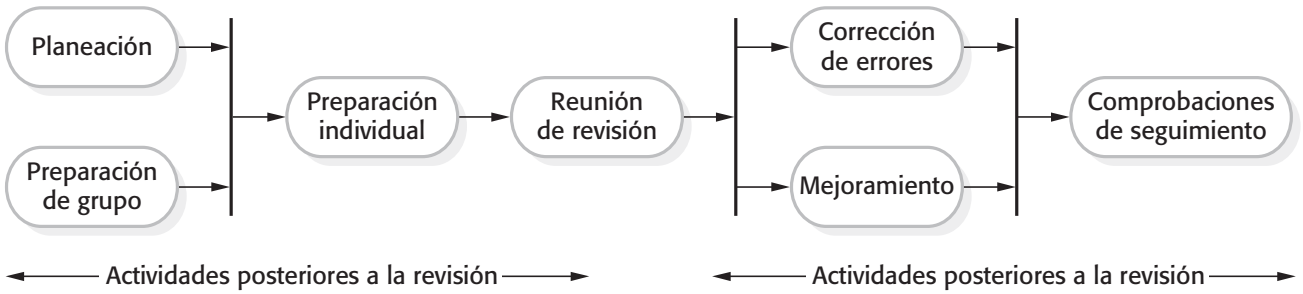

Figura 24.7
El proceso de revisión de software

que cambiarse radicalmente. Tal vez se cancelen los proyectos que desarrollaron software de alta calidad debido a cambios en la empresa o su entorno operacional.

24.3.1 El proceso de revisión

Aunque existen numerosas variaciones en los detalles de las revisiones, el proceso de revisión (figura 24.7) se estructura por lo general en tres fases:

1. *Actividades previas a la revisión* Se trata de actividades preparatorias esenciales para que sea efectiva la revisión. Por lo general, las actividades previas a la revisión se ocupan de la planeación y preparación de la revisión. La planeación de la revisión incluye establecer un equipo de revisión, organizar un tiempo, destinar un lugar para la revisión y distribuir los documentos a revisar. Durante la preparación de la revisión, el equipo puede reunirse para obtener un panorama del software a revisar. Miembros del equipo de revisión leen y entienden el software o los documentos y estándares relevantes. Trabajan de manera independiente para encontrar errores, omisiones y distanciamiento de los estándares. Si los revisores no pueden asistir a la reunión de revisión, pueden hacer sus comentarios por escrito acerca del software.

2. *La reunión de revisión* Durante la reunión de revisión un autor del documento o programa a revisar debe repasar el documento con el equipo de revisión. La revisión en sí debe ser relativamente corta, dos horas a lo sumo. Un miembro del equipo debe dirigir la revisión y otro registrar formalmente todas las decisiones y acciones de revisión a tomar. Durante la revisión, quien dirige es responsable de garantizar que se consideren todos los comentarios escritos. La dirección de la revisión debe firmar un registro de comentarios y acciones acordados durante la revisión.

3. *Actividades posteriores a la revisión* Después de terminada una reunión de revisión, deben tratarse los conflictos y problemas surgidos durante la revisión. Esto puede implicar corregir bugs de software, refactorizar el software de modo que esté conforme con los estándares de calidad, o reescribir los documentos. Algunas veces, los problemas descubiertos en una revisión de calidad son tales que es necesaria también una revisión administrativa con la finalidad de decidir si deben disponerse más recursos para corregirlos. Después de efectuar los cambios, la dirección de la revisión deberá comprobar que se hayan considerado todos los comentarios de la revisión. En ocasiones se requerirá una revisión ulterior para comprobar que los cambios realizados comprenden todos los comentarios de revisión anteriores.

Roles en el proceso de inspección

Cuando se estableció por primera vez una inspección del programa en IBM (Fagan, 1976; Fagan, 1986), se definieron algunos roles formales para los miembros del equipo de inspección. Éstos incluían: moderador, lector de código y secretario. Otros usuarios de la inspección modificaron dichos roles, pero generalmente se acepta que una inspección debe incluir al autor del código, un inspector, un secretario y un moderador que la dirija.

http://www.SoftwareEngineering-9.com/Web/QualityMan/roles.html

Con frecuencia, los equipos de revisión tienen un eje de tres o cuatro personas seleccionadas como revisores principales. Un miembro debe ser un diseñador ejecutivo, quien tendrá la responsabilidad de tomar decisiones técnicas significativas. Los revisores principales pueden invitar a otros miembros del proyecto, como los diseñadores de subsistemas relacionados, para contribuir a la revisión. Tal vez no participen en la revisión de todo el documento, pero deben concentrarse en aquellas secciones que afecten su trabajo. Como alternativa, el equipo de revisión puede hacer circular el documento y pedir comentarios por escrito de un amplio número de miembros del proyecto. El administrador del proyecto necesita participar en la revisión, a menos que se anticipen problemas que requieran cambios al plan del proyecto.

El proceso de revisión anterior se apoya en todos los miembros de un equipo de desarrollo asignado y disponible para una reunión de equipo. Sin embargo, ahora es común que los equipos de proyecto estén distribuidos, a veces a lo largo del país o en distintos continentes, así que muchas veces no es práctico que los miembros del equipo se reúnan en el mismo lugar. Ante tales situaciones, pueden usarse herramientas de edición de documentos para apoyar el proceso de revisión. Los miembros del equipo usan éstos para anotar comentarios en el documento o código fuente de software. Tales comentarios son visibles para otros miembros del equipo, quienes entonces podrán aprobarlos o rechazarlos. Sólo se requeriría una conversación telefónica cuando deban resolverse desacuerdos entre los revisores.

Por lo general, el proceso de revisión en el desarrollo de software ágil es informal. En Scrum, por ejemplo, hay una junta de revisión después de completar cada iteración del software (una revisión rápida), en la que pueden exponerse los conflictos y problemas de calidad. En la programación extrema, como se estudiará en la siguiente sección, la programación en grupos de dos personas garantiza que el código se examine y revise constantemente por otro miembro del equipo. Los conflictos de calidad general también se consideran en las reuniones diarias del equipo, pero XP se apoya en individuos que toman la iniciativa para mejorar y refactorizar el código. Por lo general, los enfoques ágiles no son dirigidos por estándares, de manera que no se consideran los asuntos de cumplimiento de estándares.

La falta de procedimientos de calidad formal en los métodos ágiles supone que puede haber problemas en el uso de enfoques ágiles en compañías que desarrollaron procedimientos de gestión de calidad detallados. Las revisiones de calidad en ocasiones aplazan el ritmo del desarrollo del software, así que éstas se emplean mejor dentro de un proceso de desarrollo dirigido por un plan. En este tipo de proceso, las revisiones pueden efectuarse mientras otros trabajos se realizan paralelamente. Esto no es práctico en enfoques ágiles que se centran de manera exclusiva en el desarrollo del código.

24.3.2 Inspecciones del programa

Las inspecciones del programa son "revisiones de pares" en las que los miembros del equipo colaboran para encontrar bugs en el programa en desarrollo. Como se explicó en el capítulo 8, las inspecciones pueden ser parte de los procesos de verificación y validación del software. Complementan las pruebas, puesto que no requieren la ejecución del programa. Esto quiere decir que es posible verificar versiones incompletas del sistema y comprobar representaciones como los modelos UML. Gilb y Graham (1993) sugieren que una de las formas más efectivas de usar las inspecciones es revisar los casos de prueba para un sistema. Las inspecciones permiten identificar problemas con las pruebas y, así, mejorar la efectividad de dichas pruebas en la detección de bugs de programa.

Las inspecciones del programa incluyen a miembros del equipo con diferentes antecedentes que realizan una cuidadosa revisión, línea por línea, del código fuente del programa. Buscan defectos y problemas, y los informan en una reunión de inspección. Los defectos pueden ser errores lógicos, anomalías en el código que indican una condición errónea o ciertas características que se hayan omitido del código. El equipo de revisión examina a detalle los modelos de diseño o el código del programa y destaca las anomalías y problemas a reparar.

Durante una inspección, con frecuencia se usa una lista de verificación de errores comunes de programación para enfocar la búsqueda de bugs. Esta lista de verificación se basa en ejemplos de libros, o bien, en el conocimiento de defectos normales en un dominio de aplicación común. Para diferentes lenguajes de programación se usan distintas listas de verificación, puesto que cada lenguaje tiene sus errores característicos. Humphrey (1989), en un amplio debate sobre inspecciones, ofrece algunos ejemplos de listas de verificación de inspección.

En la figura 24.8 se muestran las posibles comprobaciones que pueden hacerse durante el proceso de inspección. Gilb y Graham (1993) enfatizan que cada organización debe desarrollar su lista de verificación de inspección con base en estándares y prácticas locales. Dichas listas de verificación deben actualizarse regularmente, conforme se encuentren nuevos tipos de defectos. Los ítems en la lista de verificación varían según el lenguaje de programación, debido a diferentes niveles de comprobación posibles en el tiempo de compilación. Por ejemplo, un compilador Java comprueba que las funciones tengan el número correcto de parámetros; un compilador C no lo hace.

La mayoría de compañías que introdujeron las inspecciones descubrieron que éstas son muy efectivas para encontrar bugs. Fagan (1986) reportó que es posible detectar más del 60% de los errores en un programa mediante inspecciones informales de programa. Mills y sus colaboradores (1987) sugieren que un enfoque más formal a la inspección, con base en argumentos de exactitud, permite detectar más del 90% de los errores en un programa. McConnell (2004) compara las pruebas de unidad, en las que la tasa de detección de defectos es de alrededor del 25%, con las inspecciones, en las que la tasa de detección de defectos fue del 60%. También describe diversos estudios de caso, incluido un ejemplo en que la introducción de revisiones de pares condujo a un aumento en la productividad del 14% y una reducción en los defectos en el programa del 90 por ciento.

A pesar de su reconocida efectividad en términos de costos, muchas compañías de desarrollo de software se resisten a usar inspecciones o revisiones de pares. Los ingenieros de software con experiencia en pruebas de programa en ocasiones son reacios a aceptar que las inspecciones son más efectivas que las pruebas para la detección de defectos. Los administradores tal vez se muestren recelosos porque las inspecciones requieren

Clase de falla	Comprobación de inspección
Fallas de datos	• ¿Todas las variables del programa se inician antes de usar sus valores? • ¿Todas las constantes tienen nombre? • ¿La cota superior de los arreglos es igual al tamaño del arreglo o Valor – 1? • Si se usan cadenas de caracteres, ¿se asigna explícitamente un delimitador? • ¿Existe alguna posibilidad de desbordamiento de buffer?
Fallas de control	• Para cada enunciado condicional, ¿la condición es correcta? • ¿Hay certeza de que termine cada ciclo? • ¿Los enunciados compuestos están correctamente colocados entre paréntesis? • En caso de enunciados, ¿se justifican todos los casos posibles? • Si después de cada caso en los enunciados se requiere un paréntesis, ¿éste se incluyó?
Fallas de entrada/salida	• ¿Se usan todas las variables de entrada? • ¿A todas las variables de salida se les asigna un valor antes de que se produzcan? • ¿Entradas inesperadas pueden causar corrupción?
Fallas de interfaz	• ¿Todas las llamadas a función y método tienen el número correcto de parámetros? • ¿Los tipos de parámetro formal y real coinciden? • ¿Los parámetros están en el orden correcto? • Si los componentes acceden a memoria compartida, ¿tienen el mismo modelo de estructura de memoria compartida?
Fallas de gestión de almacenamiento	• Si se modifica una estructura vinculada, ¿todos los vínculos se reasignan correctamente? • Si se usa almacenamiento dinámico, ¿el espacio se asignó correctamente? • ¿El espacio se cancela explícitamente después de que ya no se requiere?
Fallas de gestión de excepción	• ¿Se tomaron en cuenta todas las posibles condiciones de error?

Figura 24.8 Lista de verificación de una inspección

costos adicionales durante el diseño y desarrollo. Quizá no quieran aceptar el riesgo de que no haya ahorros correspondientes en los costos de prueba del programa.

Los procesos ágiles pocas veces usan procesos de inspección formal o revisión de pares. En vez de ello, se apoyan en los miembros del equipo que cooperan para comprobar mutuamente el código y en lineamientos informales, tales como "comprobar antes de ingresar", lo que sugiere que los programadores deben comprobar su propio código. Los profesionales de la programación extrema argumentan que la programación en parejas es un sustituto efectivo de la inspección, ya que, en efecto, se trata de un proceso de inspección continuo. Dos personas observan cada línea de código y la comprueban antes de aceptarla.

La programación en grupos de dos conduce a un conocimiento profundo de un programa, pues ambos programadores deben entender su funcionamiento a detalle para continuar el desarrollo. En ocasiones es difícil lograr esta profundidad de conocimiento en otros procesos de inspección y, por lo tanto, la programación en grupos de dos permite

encontrar bugs que a veces no se descubrirían en inspecciones formales. Sin embargo, la programación en grupos de dos también puede conducir a malas interpretaciones de los requerimientos, en las que ambos miembros del par cometen el mismo error. Más aún, las parejas pueden tener reticencias para buscar errores, pues uno de los dos no quiere frenar el avance del proyecto. En ocasiones, las personas que participan no son tan objetivas como un equipo de inspección externo, y es probable que su habilidad para descubrir defectos esté comprometida por su cercana relación laboral.

24.4 Medición y métricas del software

La medición del software se ocupa de derivar un valor numérico o perfil para un atributo de un componente, sistema o proceso de software. Al comparar dichos valores unos con otros, y con los estándares que se aplican a través de una organización, es posible extraer conclusiones sobre la calidad del software, o valorar la efectividad de los procesos, las herramientas y los métodos de software.

Por ejemplo, suponga que una organización pretende introducir una nueva herramienta de prueba de software. Antes de introducir la herramienta, hay que registrar el número de defectos descubiertos de software en un tiempo determinado. Ésta es una línea de referencia para valorar la efectividad de la herramienta. Después de usar la herramienta durante algún tiempo, se repite este proceso. Si se descubren más defectos en el mismo lapso, después de introducida la herramienta, usted tal vez determine que ofrece apoyo útil para el proceso de validación del software.

La meta a largo plazo de la medición del software es usar la medición en lugar de revisiones para realizar juicios de la calidad del software. Al usar medición de software, un sistema podría valorarse preferentemente mediante un rango de métricas y, a partir de dichas mediciones, se podría inferir un valor de calidad del sistema. Si el software alcanzó un umbral de calidad requerido, entonces podría aprobarse sin revisión. Cuando es adecuado, las herramientas de medición pueden destacar también áreas del software susceptibles de mejora. Sin embargo, aún se está lejos de esta situación ideal y no hay señales de que la valoración automatizada de calidad será en el futuro una realidad previsible.

Una métrica de software es una característica de un sistema de software, documentación de sistema o proceso de desarrollo que puede medirse de manera objetiva. Los ejemplos de métricas incluyen el tamaño de un producto en líneas de código; el índice Fog (Gunning, 1962), que es una medida de la legibilidad de un pasaje de texto escrito; el número de fallas reportadas en un producto de software entregado, y el número de días-hombre requerido para desarrollar un componente de sistema.

Las métricas de software pueden ser métricas de control o de predicción. Como el nombre lo dice, las métricas de control apoyan la gestión del proceso, y las métricas de predicción ayudan a predecir las características del software. Las métricas de control se asocian por lo general con procesos de software. Ejemplos de las métricas de control o de proceso son el esfuerzo promedio y el tiempo requerido para reparar los defectos reportados. Las métricas de predicción se asocian con el software en sí y a veces se conocen como métricas de producto. Ejemplos de métricas de predicción son la complejidad ciclomática de un módulo (estudiado en el capítulo 8), la longitud promedio de los

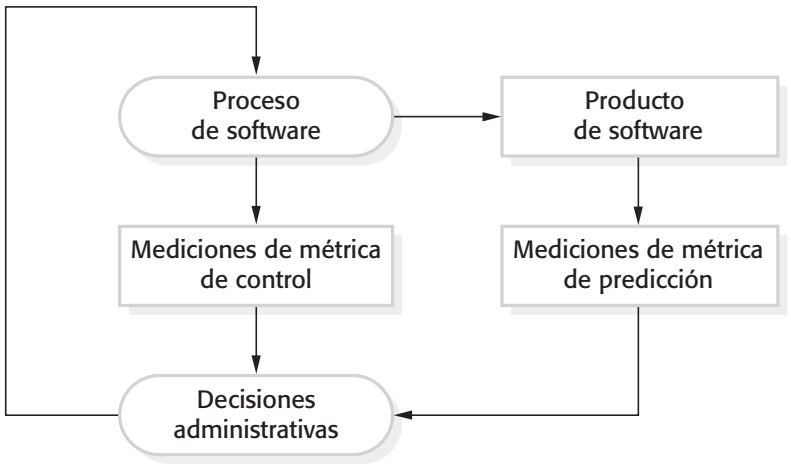

Figura 24.9 Mediciones de predicción y de control

identificadores de un programa, y el número de atributos y operaciones asociados con las clases de objetos en un diseño.

Tanto las métricas de control como las de predicción pueden influir en la toma de decisiones administrativas, como se muestra en la figura 24.9. Los administradores usan mediciones de proceso para decidir si deben hacerse cambios al proceso, y las métricas de predicción ayudan a estimar el esfuerzo requerido para hacer cambios al software. En este capítulo se estudian principalmente las métricas de predicción, cuyos valores se evalúan al analizar el código de un sistema de software. En el capítulo 26 se estudian las métricas de control y cómo se usan en el mejoramiento de procesos.

Existen dos formas en que pueden usarse las mediciones de un sistema de software:

1. *Para asignar un valor a los atributos de calidad del sistema* Al medir las características de los componentes del sistema, como su complejidad ciclomática, y luego agregar dichas mediciones, es posible valorar los atributos de calidad del sistema, tales como la mantenibilidad.
2. *Para identificar los componentes del sistema cuya calidad está por debajo de un estándar* Las mediciones pueden identificar componentes individuales con características que se desvían de la norma. Por ejemplo, es posible medir componentes para descubrir aquéllos con la complejidad más alta. Éstos tienen más probabilidad de tener bugs porque la complejidad los hace más difíciles de entender.

Lamentablemente, es difícil hacer mediciones directas de muchos de los atributos de calidad del software que se muestran en la figura 24.2. Los atributos de calidad, como mantenibilidad, comprensibilidad y usabilidad, son atributos externos que se refieren a cómo los desarrolladores y usuarios experimentan el software. Se ven afectados por factores subjetivos, como la experiencia y educación del usuario, y, por lo tanto, no pueden medirse de manera objetiva. Para hacer un juicio sobre estos atributos, hay que medir algunos atributos internos del software (como tamaño, complejidad, etcétera) y suponer que éstos se relacionan con las características de calidad por las que uno se interesa.

La figura 24.10 muestra algunos atributos externos de calidad del software y atributos internos que podrían, intuitivamente, relacionarse con ellos. Aunque el diagrama sugiere que pueden existir relaciones entre atributos externos e internos, no dice cómo se

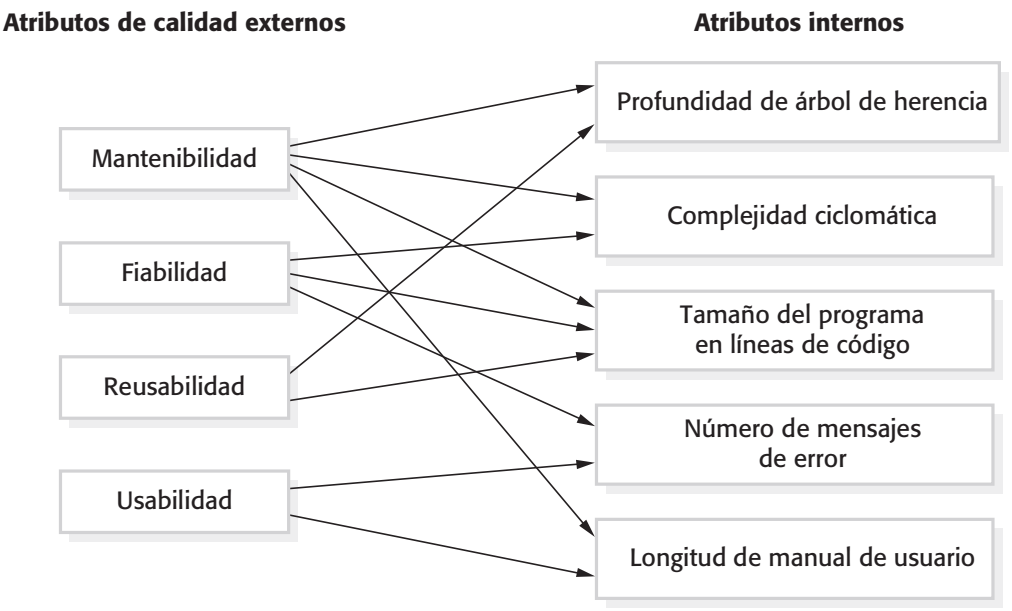

Figura 24.10 Relaciones entre software interno y externo

relacionan dichos atributos. Si la medida del atributo interno debe ser un factor de predicción útil de la característica externa del software, deben sostenerse tres condiciones (Kitchenham, 1990):

1. El atributo interno debe medirse con exactitud. Esto no siempre es un proceso directo y tal vez se requiera de herramientas de propósito especial para hacer las mediciones.
2. Debe existir una relación entre el atributo que pueda medirse y el atributo de calidad externo que es de interés. Esto es, el valor del atributo de calidad debe relacionarse, en alguna forma, con el valor del atributo que puede medirse.
3. Esta relación entre los atributos interno y externo debe comprenderse, validarse y expresarse en términos de una fórmula o un modelo. La formulación de un modelo implica identificar la manera funcional del modelo (lineal, exponencial, etcétera) mediante el análisis de datos recopilados, identificar los parámetros que se incluirán en el modelo, y calibrar dichos parámetros usando los datos existentes.

Los atributos de software internos, como la complejidad ciclomática de un componente, se miden usando herramientas de software que analizan el código fuente del software. Hay herramientas disponibles de fuente abierta que pueden utilizarse para hacer dichas mediciones. Aunque la intuición sugiere que podría existir una relación entre la complejidad de un componente de software y el número de fallas observadas en el uso, es difícil demostrar objetivamente que éste es el caso. Para probar esta hipótesis, se requieren datos de falla para un gran número de componentes y acceso al código fuente del componente para su análisis. Muy pocas compañías han establecido un compromiso a largo plazo para la recopilación de datos sobre su software, de manera que pocas veces están disponibles datos de fallas para el análisis.

En la década de 1990, numerosas y grandes compañías, como Hewlett-Packard (Grady, 1993), AT&T (Barnard y Price, 1994) y Nokia (Kilpi, 2001) introdujeron programas de métricas. Hicieron mediciones de sus productos y procesos y las usaron durante sus

procesos de gestión de calidad. La mayor parte de la atención se centró en la recolección de métricas sobre los defectos de programa y los procesos de verificación y validación. Offen y Jeffrey (1997) y Hall y Fenton (1997) tratan con más detalle la introducción en la industria de programas de métricas.

Existe escasa información disponible al público concerniente al uso actual en la industria de la medición sistemática del software. Muchas compañías reúnen información referente a su software, como el número de peticiones de cambio de requerimientos o el número de defectos descubiertos en las pruebas. Sin embargo, no es claro si usan entonces dichas mediciones de manera sistemática para comparar productos y procesos de software o para valorar el efecto de los cambios sobre los procesos y las herramientas de software. Existen algunas razones por las que esto se dificulta:

1. Es imposible cuantificar la rentabilidad de la inversión de introducir un programa de métricas organizacional. En años pasados existieron significativas mejoras en la calidad del software sin el uso de métricas, así que es difícil justificar los costos iniciales de introducir medición y valoración sistemáticas del software.
2. No hay estándares para las métricas de software o para los procesos estandarizados para medición y análisis. Muchas compañías son renuentes a introducir programas de medición hasta que se hallan disponibles tales estándares y herramientas de apoyo.
3. En gran parte de las compañías, los procesos de software no están estandarizados y se encuentran mal definidos y controlados. Por lo tanto, hay demasiada variabilidad de procesos dentro de la misma compañía para que las mediciones se usen en una forma significativa.
4. Buena parte de la investigación en la medición y métricas del software se enfoca en métricas basadas en códigos y procesos de desarrollo basados en un plan. Sin embargo, ahora cada vez más se desarrolla software mediante la configuración de sistemas ERP o COTS, o el uso de métodos ágiles. Por consiguiente, no se sabe si la investigación previa es aplicable a dichas técnicas de desarrollo de software.
5. La introducción de medición representa una carga adicional a los procesos. Esto contradice las metas de los métodos ágiles, los cuales recomiendan la eliminación de actividades de proceso que no están directamente relacionadas con el desarrollo de programas. En consecuencia, es improbable que las compañías que adoptaron los métodos ágiles aprueben un programa de métricas.

La medición y las métricas de software son la base de la ingeniería de software empírica (Endres y Rombach, 2003). Ésta es un área de investigación en la que se han usado experimentos respecto a los sistemas de software, y la recolección de datos referente a proyectos reales para formar y validar hipótesis sobre métodos y técnicas de ingeniería de software. Los investigadores que trabajan en esta área argumentan que sólo es posible confiar en el valor de los métodos y las técnicas de la ingeniería de software si se encuentra evidencia concreta de que en realidad ofrecen los beneficios que sugieren sus inventores.

Resulta lamentable que aun cuando es posible hacer mediciones objetivas y extraer conclusiones a partir de ellas, esto no necesariamente convence a quienes toman las decisiones. En vez de ello, la toma de decisiones está influida con frecuencia por factores

subjetivos, como la novedad, o la medida en que las técnicas son de interés para los profesionales. Por lo tanto, se considera que transcurrirán muchos años antes de que los resultados de la ingeniería de software empírica presenten un efecto significativo sobre la práctica de la ingeniería de software.

24.4.1 Métricas del producto

Las métricas del producto son métricas de predicción usadas para medir los atributos internos de un sistema de software. Los ejemplos de las métricas de productos incluyen el tamaño del sistema, la medida en líneas de código o el número de métodos asociados con cada clase de objeto. Por desgracia, como se explicó anteriormente en esta sección, las características del software que pueden medirse fácilmente, como el tamaño y la complejidad ciclomática, no tienen una relación clara y consistente con los atributos de calidad como comprensibilidad y mantenibilidad. Las relaciones varían dependiendo de los procesos de desarrollo, la tecnología empleada y el tipo de sistema a diseñar.

Las métricas del producto se dividen en dos clases:

1. Métricas dinámicas, que se recopilan mediante mediciones hechas de un programa en ejecución. Dichas métricas pueden recopilarse durante las pruebas del sistema o después de que el sistema está en uso. Un ejemplo es el número de reportes de bugs o el tiempo necesario para completar un cálculo.
2. Métricas estáticas, las cuales se recopilan mediante mediciones hechas de representaciones del sistema, como el diseño, el programa o la documentación. Ejemplos de mediciones estáticas son el tamaño del código y la longitud promedio de los identificadores que se usaron.

Estos tipos de métrica se relacionan con diferentes atributos de calidad. Las métricas dinámicas ayudan a valorar la eficiencia y fiabilidad de un programa. Las métricas estáticas ayudan a valorar la complejidad, comprensibilidad y mantenibilidad de un sistema de software o de los componentes del sistema.

Por lo general, existe una relación clara entre métricas dinámicas y características de calidad del software. Es muy sencillo medir el tiempo de ejecución requerido para funciones particulares y valorar el tiempo requerido con la finalidad de iniciar un sistema. Éstos se relacionan directamente con la eficiencia del sistema. De igual modo, el número de fallas del sistema y el tipo de fallas pueden registrarse y relacionarse directamente con la fiabilidad del software, que se estudió en el capítulo 15.

Como se comentó, las métricas estáticas, como las que se muestran en la figura 24.11, tienen una relación indirecta con los atributos de calidad. Se ha propuesto una gran cantidad de diferentes métricas y se han intentado muchos experimentos para derivar y validar las relaciones entre dichas métricas y atributos como complejidad y mantenibilidad. Ninguno de tales experimentos ha sido concluyente, pero el tamaño del programa y la complejidad del control parecen ser los factores de predicción más fiables de la comprensibilidad, la complejidad del sistema y la mantenibilidad.

Las métricas de la figura 24.11 son aplicables a cualquier programa, pero también se han propuesto métricas más específicas orientadas a objetos (OO). La figura 24.12 resume la suite de Chidamber y Kemerer (en ocasiones llamada suite CK) de seis métricas

Métrica de software	Descripción
Fan-in/Fan-out	Fan-in (abanico de entrada) es una medida del número de funciones o métodos que llaman a otra función o método (por ejemplo, X). Fan-out (abanico de salida) es el número de funciones a las que llama la función X. Un valor alto para fan-in significa que X está estrechamente acoplado con el resto del diseño y que los cambios a X tendrán extensos efectos dominó. Un valor alto de fan-out sugiere que la complejidad global de X puede ser alta debido a la complejidad de la lógica de control necesaria para coordinar los componentes llamados.
Longitud de código	Ésta es una medida del tamaño de un programa. Por lo general, cuanto más grande sea el tamaño del código de un componente, más probable será que el componente sea complejo y proclive a errores. Se ha demostrado que la longitud del código es una de las métricas más fiables para predecir la proclividad al error en los componentes.
Complejidad ciclomática	Ésta es una medida de la complejidad del control de un programa. Tal complejidad del control puede relacionarse con la comprensibilidad del programa. En el capítulo 8 se estudia la complejidad ciclomática.
Longitud de identificadores	Ésta es una medida de la longitud promedio de los identificadores (nombres para variables, clases, métodos, etcétera) en un programa. Cuanto más largos sean los identificadores, es más probable que sean significativos y, por ende, más comprensible será el programa.
Profundidad de anidado condicional	Ésta es una medida de la profundidad de anidado de los enunciados if en un programa. Los enunciados if profundamente anidados son difíciles de entender y proclives potencialmente a errores.
Índice Fog	Ésta es una medida de la longitud promedio de las palabras y oraciones en los documentos. Cuanto más alto sea el valor del índice Fog de un documento, más difícil será entender el documento.

Figura 24.11 Métricas estáticas de productos de software

orientadas a objetos (1994). Aunque se propusieron originalmente a principio de la década de 1990, aún son las métricas OO de más amplio uso. Algunas herramientas de diseño UML recopilan automáticamente valores para dichas métricas conforme se crean los diagramas UML.

El-Amam (2001) hace una excelente revisión de las métricas orientadas a objetos, analiza las métricas CK y otras métricas OO, y concluye que todavía no se tiene suficiente evidencia para comprender cómo estas y otras métricas orientadas a objetos se relacionan con cualidades externas de software. Esta situación no ha cambiado realmente desde su análisis en 2001. Todavía no se sabe cómo usar las mediciones de los programas orientados a objetos para extraer conclusiones fiables acerca de su calidad.

24.4.2 Análisis de componentes de software

En la figura 24.13 se ilustra un proceso de medición que puede ser parte de un proceso de valoración de calidad del software. Cada componente del sistema puede analizarse por separado mediante un rango de métricas. Los valores de dichas métricas pueden compararse entonces para diferentes componentes y, tal vez, con datos de medición históricos

Métrica orientada a objetos	Descripción
Métodos ponderados por clase (*weighted methods per class*, WMC)	Éste es el número de métodos en cada clase, ponderado por la complejidad de cada método. Por lo tanto, un método simple puede tener una complejidad de 1, y un método grande y complejo tendrá un valor mucho mayor. Cuanto más grande sea el valor para esta métrica, más compleja será la clase de objeto. Es más probable que los objetos complejos sean más difíciles de entender. Tal vez no sean lógicamente cohesivos, por lo que no pueden reutilizarse de manera efectiva como superclases en un árbol de herencia.
Profundidad de árbol de herencia (*depth of inheritance tree*, DIT)	Esto representa el número de niveles discretos en el árbol de herencia en que las subclases heredan atributos y operaciones (métodos) de las superclases. Cuanto más profundo sea el árbol de herencia, más complejo será el diseño. Es posible que tengan que comprenderse muchas clases de objetos para entender las clases de objetos en las hojas del árbol.
Número de hijos (*number of children*, NOC)	Ésta es una medida del número de subclases inmediatas en una clase. Mide la amplitud de una jerarquía de clase, mientras que DIT mide su profundidad. Un valor alto de NOC puede indicar mayor reutilización. Podría significar que debe realizarse más esfuerzo para validar las clases base, debido al número de subclases que dependen de ellas.
Acoplamiento entre clases de objetos (*coupling between object classes*, CBO)	Las clases están acopladas cuando los métodos en una clase usan los métodos o variables de instancia definidos en una clase diferente. CBO es una medida de cuánto acoplamiento existe. Un valor alto para CBO significa que las clases son estrechamente dependientes y, por lo tanto, es más probable que el hecho de cambiar una clase afecte a otras clases en el programa.
Respuesta por clase (*response for a class*, RFC)	RFC es una medida del número de métodos que potencialmente podrían ejecutarse en respuesta a un mensaje recibido por un objeto de dicha clase. Nuevamente, RFC se relaciona con la complejidad. Cuanto más alto sea el valor para RFC, más compleja será una clase y, por ende, es más probable que incluya errores.
Falta de cohesión en métodos (*lack of cohesion in methods*, LCOM)	LCOM se calcula al considerar pares de métodos en una clase. LCOM es la diferencia entre el número de pares de método sin compartir atributos y el número de pares de método con atributos compartidos. El valor de esta métrica se debate ampliamente y existe en muchas variaciones. No es claro si realmente agrega alguna información útil además de la proporcionada por otras métricas.

Figura 24.12 Suite de métricas CK orientadas a objetos

recopilados en proyectos anteriores. Las mediciones anómalas, que se desvían significativamente de la norma, pueden implicar que existen problemas con la calidad de dichos componentes.

Las etapas clave en este proceso de medición de componentes son:

1. *Elegir las mediciones a realizar* Deben formularse las preguntas que la medición busca responder, y definir las mediciones requeridas para responder a tales preguntas. Deben recopilarse las mediciones que no son directamente relevantes para dichas preguntas. El paradigma GQM (por las siglas de *Goal-Question-Metric*, es decir, Meta-Pregunta-Métrica) de Basili (Basili y Rombach, 1988), que se estudia

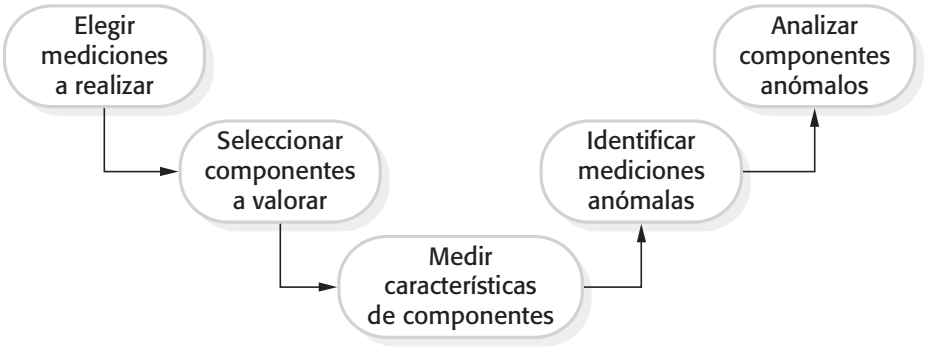

Figura 24.13 El proceso de medición de producto

en el capítulo 26, es un enfoque adecuado cuando se decide cuáles datos hay que recopilar.

2. *Seleccionar componentes a valorar* Probablemente usted no necesite estimar valores métricos para todos los componentes en un sistema de software, ya que en ocasiones podrá seleccionar una muestra representativa de componentes para medición, que le permitirá realizar una valoración global de la calidad del sistema. En otras circunstancias, tal vez desee enfocarse en los componentes centrales del sistema que están casi en uso constante. La calidad de dichos componentes es más importante que la de aquellos componentes que sólo se usan muy pocas veces.

3. *Medir las características de los componentes* Se miden los componentes seleccionados y se calculan los valores de métrica asociados. Por lo general, esto implica procesar la representación de los componentes (diseño, código, etcétera) mediante una herramienta de recolección automatizada de datos. Esta herramienta puede escribirse especialmente o ser una característica de las herramientas de diseño que ya están en uso.

4. *Identificar mediciones anómalas* Después de hacer las mediciones de componentes, se comparan entonces unas con otras y con mediciones anteriores que se hayan registrado en una base de datos de mediciones. Hay que observar los valores inusualmente altos o bajos para cada métrica, pues éstos sugieren que podría haber problemas con el componente que muestra dichos valores.

5. *Analizar componentes anómalos* Cuando identifique los componentes con valores anómalos para sus métricas seleccionadas, debe examinarlos para decidir si dichos valores de métrica anómalos significan que la calidad del componente se encuentra o no comprometida. Un valor de métrica anómalo para la complejidad (al parecer) no necesariamente significa un componente de mala calidad. Podría haber alguna otra razón para el valor alto, por lo que no necesariamente significa que haya problemas con la calidad del componente.

Siempre es conveniente mantener datos recopilados como un recurso organizacional, así como registros históricos de todos los proyectos aun cuando no se hayan usado durante un proyecto particular. Una vez establecida una base suficientemente grande de datos de medición, será posible hacer comparaciones de calidad de software a través de proyectos, además de validar las relaciones entre atributos de componentes internos y características de calidad.

24.4.3 Ambigüedad de mediciones

Cuando reúna datos cuantitativos relativos al software y los procesos de software, deberá analizar dichos datos para entender su significado. Es fácil malinterpretar los datos y hacer inferencias incorrectas. No basta con observar los datos por sí mismos, sino que también hay que considerar el contexto donde se recaban los datos.

Para ilustrar cómo pueden interpretarse los datos recopilados en diferentes formas, considere el siguiente escenario, que se ocupa del número de peticiones de cambio hechas por los usuarios de un sistema:

> *Una administradora decide monitorizar el número de peticiones de cambio enviadas por los clientes, con base en una suposición de que existe una relación entre dichas peticiones de cambio y la usabilidad y conveniencia del producto. Ella supone que cuanto más alto sea el número de peticiones de cambio, menos cumple el software las necesidades del cliente.*
>
> *Es costoso manejar las peticiones de cambio y modificar el software. Por lo tanto, la organización decide cambiar su proceso con la intención de mejorar la satisfacción del cliente y, al mismo tiempo, reducir los costos de hacer cambios. La intención es que el cambio de proceso dará como resultado mejores productos y menos peticiones de cambio.*
>
> *Los cambios de proceso se inician para aumentar la inclusión del cliente en el proceso de diseño del software. Se introducen pruebas beta de todos los productos; además, se incorporan en el producto entregado las modificaciones solicitadas por el cliente. Se entregan las nuevas versiones de los productos, que se desarrollan mediante este proceso modificado. En algunos casos se reduce el número de peticiones de cambio, aunque en otros aumenta. La administradora está confundida y descubre que es imposible valorar los efectos de los cambios de proceso sobre la calidad del producto.*

Para comprender por qué puede ocurrir este tipo de ambigüedad, hay que conocer las razones por las que los usuarios pueden hacer peticiones de cambio:

1. El software no es lo bastante bueno y no hace lo que quieren los clientes. Por lo tanto, solicitan cambios para obtener la funcionalidad que ellos requieren.
2. Como alternativa, el software puede ser muy bueno y, por consiguiente, se usa amplia e intensamente. Las peticiones de cambio pueden generarse porque existen muchos usuarios de software que piensan creativamente en nuevas ideas que podrían hacer con el software.

Por ende, aumentar la participación del cliente en el proceso puede reducir el número de peticiones de cambio para los productos con los que los clientes están descontentos. Los cambios de proceso han sido efectivos y han hecho al software más útil y adecuado. Sin embargo, alternativamente, los cambios al proceso pueden no haber funcionado, y los clientes tal vez decidieron buscar un sistema opcional. El número de peticiones de cambio disminuye porque el producto perdió participación en el mercado frente a un producto rival y, en consecuencia, hay menos usuarios del producto.

Por otra parte, los cambios al proceso pueden conducir a muchos nuevos clientes satisfechos que deseen participar en el proceso de desarrollo del producto. Por lo tanto, generan más peticiones de cambio. Los cambios al proceso de manejar las peticiones de cambio contribuyen a este aumento. Si la compañía tiene mayor capacidad de respuesta con los clientes, ellos generarán más peticiones de cambio porque saben que éstas se tomarán con seriedad. Creen que sus sugerencias se incorporarán quizás en versiones posteriores del software. O bien, el número de peticiones de cambio puede aumentar porque los sitios de prueba beta no eran los típicos del mayor uso del programa.

Para analizar los datos de petición de cambio, no basta con conocer el número de peticiones de cambio, sino que se necesita conocer quién hizo la petición, cómo usa el software y por qué hizo la petición. También se requiere información sobre los factores externos, como modificaciones al procedimiento de petición de cambio o cambios al mercado que puedan tener un efecto. Con esta información, es posible averiguar si los cambios al proceso fueron efectivos para aumentar la calidad del producto.

Esto ilustra las dificultades de entender los efectos de los cambios, y el enfoque "científico" a este problema es reducir el número de factores que tiendan a afectar las mediciones hechas. Sin embargo, los procesos y productos que se miden no están aislados de su entorno. El ambiente empresarial cambia constantemente y es imposible evitar los cambios a la práctica laboral sólo porque pueden hacerse comparaciones de datos inválidos. Como tales, los datos cuantitativos sobre las actividades humanas no siempre deben tomarse en serio. Las razones por las que cambia un valor medido con frecuencia son ambiguas. Dichas razones deben investigarse a profundidad antes de extraer conclusiones de cualquier medición que se haya realizado.

PUNTOS CLAVE

- La gestión de calidad del software se ocupa de garantizar que el software tenga un número menor de defectos y que alcance los estándares requeridos de mantenibilidad, fiabilidad, portabilidad, etcétera. Incluye definir estándares para procesos y productos, y establecer procesos para comprobar que se siguieron dichos estándares.
- Los estándares de software son importantes para el aseguramiento de la calidad, pues representan una identificación de las "mejores prácticas". Al desarrollar el software, los estándares proporcionan un cimiento sólido para diseñar software de buena calidad.
- Es necesario documentar un conjunto de procedimientos de aseguramiento de la calidad en un manual de calidad organizacional. Esto puede basarse en el modelo genérico para un manual de calidad sugerido en el estándar ISO 9001.
- Las revisiones de los entregables del proceso de software incluyen a un equipo de personas que verifican que se siguieron los estándares de calidad. Las revisiones son la técnica usada más ampliamente para valorar la calidad.
- En una inspección de programa o revisión de pares, un reducido equipo comprueba sistemáticamente el código. Ellos leen el código a detalle y buscan posibles errores y omisiones. Entonces los problemas detectados se discuten en una reunión de revisión del código.

- La medición del software puede usarse para recopilar datos cuantitativos tanto del software como del proceso de software. Se usan los valores de las métricas de software recopilados para hacer inferencias referentes a la calidad del producto y el proceso.
- Las métricas de calidad del producto son particularmente útiles para resaltar los componentes anómalos que pudieran tener problemas de calidad. Dichos componentes deben entonces analizarse con más detalle.

LECTURAS SUGERIDAS

Metrics and Models for Software Quality Engineering, 2nd edition. Éste es un análisis muy completo de las métricas del software que incluyen métricas de proceso, de producto y orientadas a objetos. También contiene cierto conocimiento matemático requerido para desarrollar y comprender modelos basados en medición de software. (S. H. Kan, Addison-Wesley, 2003.)

Software Quality Assurance: From Theory to Implementation. Un excelente vistazo actualizado a los principios y la práctica del aseguramiento de la calidad del software. Incluye un análisis de los estándares, como el ISO 9001. (D. Galin, Addison-Wesley, 2004.)

"A Practical Approach for Quality-Driven Inspections". En la actualidad muchos artículos concernientes a las inspecciones son más bien anticuados, ya que no consideran la práctica moderna del desarrollo de software. Este texto relativamente reciente describe un método de inspección que se ocupa de algunos de los problemas al utilizar la inspección y sugiere cómo pueden usarse las inspecciones en un entorno moderno de desarrollo. (C. Denger, F. Shull, *IEEE Software*, **24** (2), marzo-abril de 2007.) http://dx.doi.org/10.1109/MS.2007.31

"Misleading Metrics and Unsound Analyses". Un excelente artículo de los principales investigadores de métricas, quienes analizan las dificultades de comprensión de lo que significan realmente las métricas. (B. Kitchenham, R. Jeffrey y C. Connaughton, *IEEE Software*, **24** (2), marzo-abril de 2007.) http://dx.doi.org/10.1109/MS.2007.49.

"The Case for Quantitative Project Management". Ésta es una introducción a una sección especial de la revista que incluye otros dos artículos sobre administración cuantitativa de proyectos. Plantea razones para una mayor investigación en métricas y medición con la finalidad de mejorar la administración de proyectos de software. (B. Curtis et al., *IEEE Software*, **25** (3), mayo-junio de 2008.) http://dx.doi.org/10.1109/MS.2008.80.

EJERCICIOS

24.1. Explique por qué un proceso de software de alta calidad debería conducir a productos de alta calidad de software. Discuta los posibles problemas con este sistema de gestión de calidad.

24.2. Exponga cómo pueden usarse los estándares para obtener conocimiento de la organización sobre los métodos efectivos de desarrollo de software. Sugiera cuatro tipos de conocimiento que puedan reflejarse en los estándares de la organización.

24.3. Discuta la valoración de calidad del software según los atributos de calidad mostrados en la figura 24.2. Debe considerar a la vez cada atributo y explicar cómo puede valorarse.

24.4. Diseñe un formato electrónico que pueda usar para registrar comentarios de revisión y para enviar comentarios por correo electrónico a los revisores.

24.5. Describa brevemente posibles estándares que podría utilizar para:

- El uso de sentencias de control en C, C# o Java;
- enviar reportes para un proyecto final en una universidad;
- el proceso de hacer y aprobar cambios al programa (véase el capítulo 26);
- el proceso de comprar e instalar una nueva computadora.

24.6. Suponga que trabaja en una organización que desarrolla productos de bases de datos para individuos y empresas pequeñas. Esta organización está interesada en cuantificar su desarrollo de software. Escriba un reporte que sugiera métricas adecuadas y mencione cómo pueden recopilarse.

24.7. Exprese por qué las inspecciones de programa son una técnica efectiva para descubrir errores en un programa. ¿Qué tipos de errores tienen escasa probabilidad de descubrirse mediante inspecciones?

24.8. Diga por qué las métricas de diseño son, por sí mismas, un método inadecuado para predecir la calidad del diseño.

24.9. Exponga por qué es difícil validar las relaciones entre atributos de producto internos (como la complejidad ciclomática) y los atributos externos (como la mantenibilidad).

24.10. Un colega que es muy buen programador elabora software con un bajo número de defectos, pero siempre pasa por alto los estándares de calidad de la organización. ¿Cómo deberían reaccionar sus administradores ante este comportamiento?

REFERENCIAS

Bamford, R. y Deibler, W. J. (eds.) (2003). "ISO 9001:2000 for Software and Systems Providers: An Engineering Approach". Boca Raton, Fla.: CRC Press.

Barnard, J. y Price, A. (1994). "Managing Code Inspection Information". *IEEE Software*, **11** (2), 59–69.

Basili, V. R. y Rombach, H. D. (1988). "The TAME project: Towards Improvement-Oriented Software Environments". *IEEE Trans. on Software Eng.*, **14** (6), 758–773.

Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., Macleod, G. y Merrit, M. (1978). *Characteristics of Software Quality*. Amsterdam: North-Holland.

Chidamber, S. y Kemerer, C. (1994). "A Metrics Suite for Object-Oriented Design". *IEEE Trans. on Software Eng.*, **20** (6), 476–93.

Crosby, P. (1979). *Quality is Free*. Nueva York: McGraw-Hill.

El-Amam, K. 2001. "Object-oriented Metrics: A Review of Theory and Practice". National Research Council of Canada. http://seg.iit.nrc.ca/English/abstracts/NRC44190.html .

Endres, A. y Rombach, D. (2003). *Empirical Software Engineering: A Handbook of Observations, Laws and Theories*. Harlow, UK: Addison-Wesley.

Fagan, M. E. (1976). "Design and code inspections to reduce errors in program development". *IBM Systems J.*, **15** (3), 182–211.

Fagan, M. E. (1986). "Advances in Software Inspections". *IEEE Trans. on Software Eng.*, **SE-12** (7), 744–51.

Gilb, T. y Graham, D. (1993). *Software Inspection*. Wokingham: Addison-Wesley.

Grady, R. B. (1993). "Practical Results from Measuring Software Quality". *Comm. ACM*, **36** (11), 62–8.

Gunning, R. (1962). *Techniques of Clear Writing*. Nueva York: McGraw-Hill.

Hall, T. y Fenton, N. (1997). "Implementing Effective Software Metrics Programs". *IEEE Software*, **14** (2), 55–64.

Humphrey, W. (1989). *Managing the Software Process*. Reading, Mass.: Addison-Wesley.

IEC. 1998. "Standard IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems". International Electrotechnical Commission: Ginebra.

IEEE. (2003). *IEEE Software Engineering Standards Collection on CD-ROM*. Los Alamitos, Ca.: IEEE Computer Society Press.

Ince, D. (1994). *ISO 9001 and Software Quality Assurance*. Londres: McGraw-Hill.

Kilpi, T. (2001). "Implementing a Software Metrics Program at Nokia". *IEEE Software*, **18** (6), 72–7.

Kitchenham, B. (1990). "Measuring Software Development". En *Software Reliability Handbook*. Rook, P. (ed.). Amsterdam: Elsevier, 303–31.

McConnell, S. (2004). *Code Complete: A Practical Handbook of Software Construction, 2nd edition*. Seattle: Microsoft Press.

Mills, H. D., Dyer, M. y Linger, R. (1987). "Cleanroom Software Engineering". *IEEE Software*, **4** (5), 19–25.

Offen, R. J. y Jeffrey, R. (1997). "Establishing Software Measurement Programs". *IEEE Software*, **14** (2), 45–54.

Stalhane, T. y Hanssen, G. K. (2008). "The application of ISO 9001 to agile software development". *9th International Conference on Product Focused Software Process Improvement, PROFES 2008*, Monte Porzio Catone, Italy: Springer.