CAPÍTULO

26 GESTIÓN DE LA CALIDAD

El enfoque de ingeniería del software descrito en este libro se dirige hacia una sola meta: producir software de alta calidad. Aunque a muchos lectores les parecerá un reto la pregunta: ¿qué es calidad del software?

Philip Crosby [CRO79], en su libro fundamental acerca de calidad, ofrece una respuesta irónica a esta pregunta:

> El problema de la gestión de la calidad no es lo que la gente ignora acerca de ella. El problema es lo que creen saber...
>
> A este respecto, la calidad tiene mucho en común con el sexo. Todo el mundo lo quiere. (En ciertas condiciones, desde luego.) Todos sienten que lo entienden. (Aun cuando no quieran explicarlo.) Todos piensan que su ejecución sólo es cuestión de seguir las inclinaciones naturales. (Después de todo, la gente se las arregla de alguna forma.) Y, desde luego, la mayoría de las personas piensa que los problemas en estas áreas los provocan otras personas. (Si sólo se tomaran el tiempo para hacer las cosas bien.)

Algunos desarrolladores de software continúan creyendo que la calidad de éste es algo en lo que se debe comenzar a preocupar sólo después de que se haya generado el código. ¡Nada podría estar más alejado de la verdad! La *gestión de la calidad* (con frecuencia llamada *garantía de la calidad del software*) es una actividad protectora o de sombrilla (capítulo 2) que se aplica a lo largo del proceso de software.

La gestión de la calidad abarca 1) un proceso de garantía de la calidad del software (SQA, *por sus siglas en inglés*), 2) tareas específicas de aseguramiento y control de la calidad (que incluyen revisiones técnicas formales y una estrategia de pruebas de varios niveles); 3) prácticas efectivas de ingeniería del software (métodos y herramientas); 4) control de todos los productos de trabajo del soft-

UN VISTAZO RÁPIDO

¿Qué es? No es suficiente hablar por hablar diciendo que la calidad del software es importante. Se tiene que 1) definir explícitamente qué quiere decir cuando dice "calidad del software", 2) crear un conjunto de actividades que ayudarán a asegurar que todo producto de trabajo de ingeniería del software presentará alta calidad, 3) realizar actividades de control y aseguramiento de la calidad en cada proyecto de software, 4) usar métricas para desarrollar estrategias que mejoren el proceso de software y, como consecuencia, la calidad del producto final.

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

¿Por qué es importante? Es posible hacerlo bien o hacerlo de nuevo otra vez. Si un equipo de software subraya la calidad en todas

las actividades de ingeniería del software, ello reduce la cantidad de reelaboración que se debe realizar. Esto resulta en menores costos y, más importante, mejorará el tiempo de llegada al mercado.

¿Cuáles son los pasos? Antes de se que inicien las actividades de aseguramiento de la calidad del software es importante definir "calidad del software" en diversos grados de abstracción. Una vez que entienda qué es calidad, un equipo de software debe identificar un conjunto de actividades SQA que filtrarán los errores de los productos de trabajo antes de que se aprueben.

¿Cuál es el producto obtenido? Se crea un "Plan de aseguramiento de la calidad del software" para definir la estrategia SQA del equipo del software. Durante el análisis, diseño y generación de código el principal producto de trabajo SQA es un breve informe de la revisión técnica formal. Durante las pruebas se producen los planes de prueba y los procedimientos. También se pueden generar otros productos de trabajo asociados con la mejora del proceso.

¿Cómo puedo estar seguro de que lo he hecho correctamente? ¡Encuentre los errores antes de que se conviertan en defectos! Es decir, trabaje para mejorar su eficiencia en la eliminación de defectos (capítulo 22), con lo que se reduce la cantidad de reelaboración que su equipo de software tiene que realizar.

ware y los cambios que generan (capítulo 27); 5) un procedimiento para garantizar la concordancia con los estándares de desarrollo del software (cuando sea aplicable), y 6) mecanismos de medición e informe.

Este capítulo se centra en los temas de gestión y las actividades específicas del proceso que permiten a una organización de software garantizar que hace las cosas correctas en el momento justo y en la forma correcta.

26.1 Conceptos de calidad[1]

El control de la variación es la clave para un producto de alta calidad. En el contexto del software se lucha por controlar la variación en el proceso genérico que se aplica y el énfasis de calidad que permea el trabajo de ingeniería del software.

El *control de la variación* es el corazón del control de calidad. Un fabricante quiere minimizar la variación entre los productos que se producen, aun cuando haga algo relativamente simple como duplicar DVD. Seguramente, esto no puede ser un problema: la duplicación de los DVD es una operación simple de fabricación, y es posible garantizar que siempre se creen duplicados exactos del software.

¿Se puede? Es necesario asegurarse de que las pistas se colocan en los DVD dentro de una tolerancia especificada de modo que la mayoría abrumadora de los controladores de DVD pueda leer el medio. Las máquinas de duplicación de discos pueden, y lo hacen, aceptar y rechazar la tolerancia. Así que incluso un proceso "simple" como la duplicación de DVD puede encontrar problemas debidos a la variación entre muestras.

¿Pero cómo se aplica esto al trabajo de software? ¿Cómo puede una organización de desarrollo de software necesitar controlar la variación? De un proyecto a otro se quiere minimizar la diferencia entre los recursos predichos necesarios para comple-

1 Esta sección, escrita por Michael Stovsky, ha sido adaptada de "Fundamentals of ISO 9000", un libro de trabajo desarrollado para *Essential Software Engineering*, un video curriculum desarrollado por R. S. Pressman & Associates, Inc. Reimpreso con permiso.

tar un proyecto y los recursos reales utilizados, que incluyen personal, equipo y tiempo. En general, se quisiera estar seguro de que el programa de pruebas abarca un porcentaje conocido del software, de una liberación a otra. No sólo se quiere minimizar el número de defectos que se liberan, sino que se quiere asegurar que la varianza en el número de *bugs* también se minimiza de una liberación a otra. (Los clientes probablemente se molestarán si la tercera liberación de un producto tiene diez veces más defectos que la liberación previa.) Nos gustaría minimizar las diferencias en rapidez y precisión de las respuestas de la línea de soporte para los problemas de los clientes. La lista puede continuar indefinidamente.

26.1.1 Calidad

El *American Heritage Dictionary* define *calidad* como "una característica o atributo de algo". Como un atributo de un elemento, la calidad se refiere a características mensurables, es decir: cosas que se pueden comparar para conocer estándares, como longitud, color, propiedades eléctricas y maleabilidad. Sin embargo, el software, principalmente una entidad intelectual, es más difícil de caracterizar que los objetos físicos.

No obstante, existen las mediciones de las características de un programa. Dichas propiedades incluyen complejidad ciclomática, cohesión, número de puntos de función, líneas de código y muchas otras examinadas en el capítulo 15. Cuando se examina un elemento con base en sus características mensurables se pueden encontrar dos tipos de calidad: calidad de diseño y calidad de concordancia.

La *calidad de diseño* se refiere a las características que los diseñadores especifican para un elemento. La *calidad de concordancia* es el grado en el que las especificaciones de diseño se aplican durante la fabricación.

> "La gente olvida cuán rápido hiciste un trabajo, pero siempre recuerdan cuán bien lo hiciste."
>
> **Howard Newton**

En el desarrollo de software, la calidad del diseño incluye requisitos, especificaciones y el diseño del sistema. La calidad de concordancia es un tema enfocado principalmente en la implementación. Si ésta sigue el diseño y el sistema resultante satisface sus requisitos y metas de desempeño, la calidad de concordancia es alta.

¿Pero la calidad del diseño y la calidad de concordancia son los únicos temas que deben considerar los ingenieros de software? Robert Glass [GLA98] argumenta que es conveniente una relación más "intuitiva":

satisfacción del usuario = producto manejable + buena calidad
+ entrega dentro de presupuesto y tiempo

En el fondo, Glass afirma que la calidad es importante, pero si el usuario no está satisfecho, nada más importa en realidad. DeMarco [DEM99] refuerza esta visión cuando afirma: "La calidad de un producto es una función de cuánto cambia el mundo para mejorar". Esta visión de la calidad afirma que si un producto de software pro-

porciona beneficio sustancial a sus usuarios finales, éstos estén dispuestos a tolerar problemas ocasionales en confiabilidad y desempeño.

26.1.2 Control de calidad

¿Qué es control de calidad de software?

El control de la variación puede equipararse con el control de calidad. Pero, ¿cómo se logra el control de calidad? Éste involucra la serie de inspecciones, revisiones y pruebas empleadas a lo largo del proceso de software para garantizar que cada producto de trabajo satisfaga los requisitos que se le han asignado. El control de calidad incluye un bucle de retroalimentación con el proceso que creó el producto de trabajo. La combinación de medición y retroalimentación permite afinar el proceso cuando los productos de trabajo creados fracasan en cuanto a satisfacer sus especificaciones.

Un concepto clave del control de calidad es que todos los productos de trabajo tienen especificaciones definidas mensurables con las cuales se puede comparar la salida de cada proceso. El bucle de retroalimentación es esencial para minimizar los defectos producidos.

26.1.3 Garantía de la calidad

Referencia Web

Útiles vínculos a recursos de SQA se pueden encontrar en **www.qualitytree.com/links/index.htm.**

La garantía de la calidad consiste en un conjunto de funciones de auditoría e información que evalúan la efectividad y qué tan completas son las actividades de control de calidad. La meta del aseguramiento de la calidad es brindarle al gestor los datos necesarios para que esté informado acerca de la calidad del producto, y por consiguiente que comprenda y confíe en que la calidad del producto está satisfaciendo sus metas. Desde luego, si los datos que ofrece el aseguramiento de la calidad identifican problemas, es responsabilidad del gestor abordarlos y aplicar los recursos necesarios para resolver los conflictos de calidad.

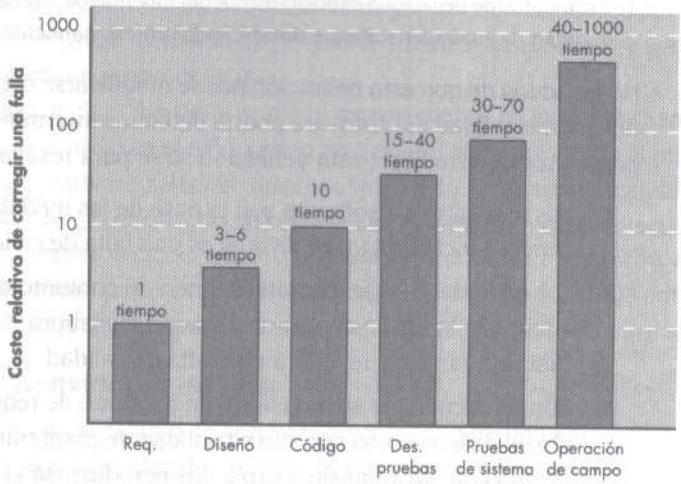
26.1.4 Costo de la calidad

El costo de la calidad incluye todos los costos que genera la búsqueda de calidad o que demanda el desarrollo de las actividades relacionadas con la calidad. Los estudios de costo de la calidad se llevan a cabo para ofrecer una línea base para el costo actual de la calidad, identificar oportunidades que reduzcan el costo de calidad y proporcionar una base normalizada de comparación. La base de la normalización casi siempre es monetaria. Una vez que se han normalizado los costos de la calidad sobre una base monetaria, se tienen los datos necesarios para evaluar dónde se encuentran las oportunidades para mejorar los procesos. Más todavía, se puede evaluar el efecto de los cambios en términos monetarios.

¿Cuáles son los componentes del costo de calidad?

Los costos de calidad se dividen en costos asociados con prevención, evaluación y fallas. Los *costos de prevención* incluyen planificación de la calidad, revisiones técnicas formales, equipo de pruebas y entrenamiento. Los *costos de evaluación* incluyen actividades para comprender mejor la condición del producto la "primera vez a través de" cada proceso. Los ejemplos de costos de evaluación incluyen inspección en el proceso y entre procesos, calibración y mantenimiento de equipo y pruebas.

FIGURA 26.1
Costo relativo de corregir una falla.



CONSEJO

No tema incurrir en significativos costos de prevención. Descanse tranquilo de que su inversión proporcionará un excelente rendimiento.

Los *costos de fallas* son aquellos que desaparecerían si no aparecieran defectos antes de enviar un producto a los clientes. Estos costos se subdividen en costos de fallas internas y externas. Se incurre en los *costos de fallas internas* cuando se detecta un defecto en el producto antes del envío. Los costos de fallas internas incluyen reelaboración, reparación y análisis en modo de falla. Los *costos de fallas externas* se asocian con defectos detectados después de que el producto ha sido enviado al cliente. Los ejemplos de costos de fallas externas son la resolución de las quejas, devolución y reemplazo del producto, soporte de ayuda en línea y trabajo de garantía.

Como se esperaba, los costos relativos para encontrar y reparar un defecto aumentan sustancialmente conforme se pasa de la prevención a detección y de los de falla interna a los de falla externa.

La figura 26.1, basada en datos recopilados por Boehm [BOE81] y otros, ilustra este fenómeno.

> "Toma menos tiempo hacer una cosa bien que explicar por qué la hiciste mal."
>
> **H. W. Longfellow**

26.2 GARANTÍA DE LA CALIDAD DEL SOFTWARE (SQA)

Incluso los desarrolladores de software más exhaustos estarán de acuerdo en que el software de alta calidad es una meta importante. Pero, ¿cómo se define calidad? Un bromista dijo una vez: "Todo programa hace algo bien, sólo que puede ser la cosa que no queremos que haga".

En la bibliografía se han propuesto muchas definiciones de la calidad del software. En cuanto a los propósitos del presente texto, la *calidad del software* se define de la siguiente manera:

¿Cómo se define la calidad del software?

Concordancia con los requisitos funcionales y de desempeño explícitamente establecidos, estándares de desarrollo explícitamente documentados y características implícitas que se esperan de cualquier software desarrollado profesionalmente.

No hay duda de que esta definición puede modificarse o extenderse. De hecho, la definición de calidad del software podría debatirse interminablemente. En cuanto a los propósitos de este libro, esta definición sirve para resaltar tres puntos importantes:

1. Los requisitos de software son la base de las medidas de la calidad. La falta de concordancia con los requisitos es una falta de calidad.
2. Los estándares especificados definen un conjunto de criterios de desarrollo que guían la forma en que el software se elabora. Si no se siguen los criterios, casi seguramente resultará una falta de calidad.
3. Con frecuencia no se menciona un conjunto de requisitos implícito (por ejemplo, el deseo de uso sencillo y facilidad de mantenimiento). Si el software concuerda con sus requisitos explícitos pero fracasa al satisfacer los requisitos implícitos, su calidad está en duda.

26.2.1 Algunos antecedentes

El control y la garantía de la calidad son actividades esenciales en cualquier negocio que elabore productos de consumo. Antes del siglo xx, el control de calidad era responsabilidad exclusiva del empresario que fabricaba un producto. La primera función formal de garantía y control de calidad la introdujeron los Laboratorios Bell en 1916 y se extendió rápidamente a través del mundo industrial. Durante el decenio de 1940 se sugirieron enfoques más formales del control de calidad, los cuales se apoyaban en la medición y la mejora continua de los procesos [DEM86] como los elementos clave de la gestión de la calidad.

> "Cometiste demasiados malos errores."
>
> **Yogi Berra**

En la actualidad, toda compañía tiene mecanismos que garantizan la calidad en sus productos. De hecho, las afirmaciones explícitas de la preocupación de una compañía por la calidad se han convertido en una táctica de mercadotecnia durante las décadas pasadas.

La historia de la garantía de la calidad en el desarrollo de software avanza paralela a la de la calidad en la fabricación de hardware. Durante los primeros días de la computación (décadas de 1950 y 1960), la calidad era responsabilidad exclusiva del programador. Los estándares de garantía de la calidad para el software se introdujeron en los contratos militares de desarrollo de software durante el decenio de 1970 y se han extendido rápidamente en el desarrollo del software en el mundo de los negocios [IEE94]. Si se extiende la definición presentada anteriormente, la garantía de la calidad del software es un "patrón de acciones sistemático y planificado" [SCH98]

que se requiere para garantizar alta calidad en el software. Numerosos y diversos participantes tienen responsabilidad en la garantía de la calidad del software: ingenieros de software, gestores de proyecto, clientes, vendedores y los individuos que participan en un grupo de SQA.

El grupo de SQA funciona como el representante en casa del cliente. Es decir, las personas que realizan el SQA deben observar el software desde el punto de vista del cliente. ¿El software satisface adecuadamente los factores de calidad señalados en el capítulo 15? ¿El desarrollo de software se ha llevado a cabo de acuerdo con los estándares preestablecidos? ¿Las disciplinas técnicas han realizado adecuadamente sus tareas como parte de la actividad de SQA? El grupo de SQA intenta responder éstas y otras preguntas para garantizar que la calidad del software se conserva.

26.2.2 Actividades de SQA

La garantía de la calidad de software se compone de una variedad de tareas asociadas con dos integrantes diferentes: los ingenieros de software que realizan el trabajo técnico y un grupo de SQA que tiene la responsabilidad de planificar, supervisar, guardar registros, analizar y reportar la garantía de calidad.

Los ingenieros de software abordan la calidad (y realizan actividades de aseguramiento y control de calidad) al aplicar sólidos métodos y medidas técnicas, llevar a cabo revisiones técnicas formales y desarrollar pruebas de software bien planificadas. En este capítulo sólo se examinan las revisiones. Los tópicos de tecnología se tratan en las partes 1, 2, 3 y 5 de este libro.

La misión del grupo de SQA es auxiliar al equipo de software a conseguir un producto final de alta calidad. El Software Engineering Institute (SEI) recomienda un conjunto de actividades de SQA que abordan la planificación, supervisión, conservación de registros, análisis y elaboración de informes de aseguramiento de la calidad. Dichas actividades las realiza (o facilita) un grupo de SQA independiente que se encarga de las siguientes actividades:

¿Cuál es el papel de un grupo de SQA?

Preparar un plan de SQA para un proyecto. El plan se desarrolla durante la planificación del proyecto y lo revisan todos los participantes. Las actividades de garantía de la calidad del equipo de ingeniería del software y del grupo de SQA las rige el plan. Éste identifica las evaluaciones que se realizarán, las auditorías y revisiones para llevar a cabo, los estándares aplicables al proyecto, los procedimientos para el informe y seguimiento de errores, los documentos que debe producir el grupo de SQA y la cantidad de retroalimentación proporcionada al equipo de proyecto de software.

Participar en el desarrollo de la descripción del proceso de software del proyecto. El equipo de software selecciona un proceso para el trabajo que habrá de realizarse. El grupo de SQA revisa la descripción del proceso para que concuerde con las políticas organizacionales, los estándares internos de software, los estándares impuestos de manera externa (por ejemplo, ISO-9001) y otras partes del plan de proyecto del software.

Revisar las actividades de ingeniería del software para verificar que se ajuste al proceso de software definido. El grupo de SQA identifica, documenta y sigue las desviaciones del proceso y verifica que se hayan hecho las correcciones.

Audita productos de trabajo de software seleccionados para verificar que se ajusten con los definidos como parte del proceso del software. El grupo de SQA revisa los productos de trabajo seleccionados, identifica, documenta y sigue las desviaciones; verifica que se hayan hecho las correcciones; y periódicamente informa de los resultados de su trabajo al gestor del proyecto.

Garantiza que las desviaciones en el trabajo del software y en los productos de trabajo estén documentadas y se manejen de acuerdo con el procedimiento establecido. Las desviaciones se pueden encontrar en el plan del proyecto, en la descripción del proceso, en los estándares aplicables o en los productos de trabajo técnicos.

Registra cualquier falta de ajuste y lo informa al gestor ejecutivo. A los elementos que no se ajustan se les da seguimiento hasta resolverlos.

Además de estas actividades, el grupo de SQA coordina el control y la gestión del cambio (capítulo 27) y ayuda a recopilar y analizar métricas de software.

26.3 REVISIONES DEL SOFTWARE

Las revisiones son como filtros en el flujo de trabajo del proceso de software. Muy poco y el flujo está "sucio". Demasiado y el flujo se reduce a un chorrito. Use métricas para determinar qué revisiones funcionan y resáltelas.

Las revisiones del software son un "filtro" para el proceso de software. Esto es, las revisiones se aplican en varios puntos durante la ingeniería del software y sirven para descubrir errores y defectos que luego pueden eliminarse. Las revisiones del software "purifican" las actividades de ingeniería del software que se han denominado análisis, diseño y codificación. Freddman y Weinberg [FRE90] abordan del modo siguiente la necesidad de las revisiones:

> El trabajo técnico necesita revisarse por la misma razón que los lápices necesitan gomas. *Errar es humano.* La segunda razón por la que se necesitan las revisiones técnicas es que, aunque la gente sea buena al captar algunos de sus propios errores, las grandes clases de errores escapan de su creador con más facilidad de lo que se le escapan a alguien más.

Como parte de la ingeniería del software se pueden llevar a cabo muchos tipos de revisiones. Cada uno tiene su lugar. Una reunión informal en torno a una máquina expendedora de café es una forma de revisión, si se examinan los problemas técnicos. Una presentación formal del diseño de software a un auditorio de clientes, gestores y personal técnico también es una forma de revisión. Sin embargo, este libro se enfoca sobre la *revisión técnica formal*, a veces llamada *comprobación manual del código (walkthrough)* o *inspección*. Una revisión técnica formal (RTF) es el filtro más efectivo desde un punto de vista de aseguramiento de la calidad. Dirigida por los ingenieros de software (u otras personas) para ingenieros de software, la RTF es un medio efectivo para descubrir errores y mejorar la calidad del software.

INFORMACIÓN

Bugs, errores y defectos

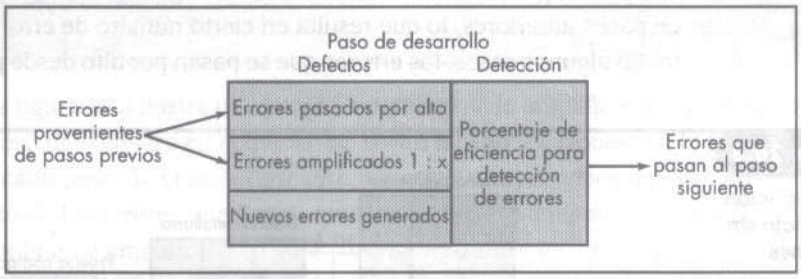
La meta del SQA es eliminar los problemas de calidad en el software. A estos problemas se les conoce con varios nombres: "bugs", "fallas", "errores" o "defectos", por mencionar unos cuantos. ¿Cada uno de éstos son términos sinónimos o existen sutiles diferencias entre ellos?

En este libro se ha hecho una clara distinción entre un *error* (un problema de calidad descubierto *antes* de que el software sea liberado entre los usuarios finales) y un *defecto* (un problema de calidad detectado sólo *después* de que el software ha sido liberado entre los usuarios finales).[2] Se ha hecho esta distinción porque los errores y defectos tienen impactos económicos, de negocios, psicológicos y humanos muy diferentes. Como ingenieros de software se quiere descubrir y corregir tantos errores como sea posible antes de que el cliente o usuario final los encuentre. Se quieren evitar los defectos porque (justificadamente) hacen ver mal a la gente de software.

Sin embargo, es importante mencionar que la distinción temporal hecha en este libro entre errores y defectos *no* es la tendencia predominante. El consenso general entre la comunidad de ingeniería del software es que defectos y errores, fallas y bugs son sinónimos. Es decir, el momento en que el problema se descubrió no tiene importancia en cuanto al término con que se describe el problema. Parte del argumento en favor de esta visión es que a veces es difícil distinguir con claridad entre preliberación y posliberación (por ejemplo, considérese un proceso incremental utilizado en el desarrollo ágil [capítulo 4]).

Sin importar cómo se elija interpretar estos términos, reconózcase que el momento en que se descubre un problema sí importa y que los ingenieros de software deben intentar duro, muy duro, detectar los problemas antes de que los clientes y usuarios finales los encuentren. Si se tiene un interés posterior en este tema, una revisión razonablemente amplia de la terminología que rodea a los "bugs" se puede encontrar en www.softwaredevelopment.ca/bugs.shtml.

FIGURA 26.2 Modelo de amplificación de defecto.



26.3.1 Impacto de los defectos de software en el costo

El objetivo principal de las revisiones técnicas formales es descubrir los errores durante el proceso, de modo que no se conviertan en defectos después de liberar el software. El beneficio obvio de las revisiones técnicas formales es el descubrimiento temprano de los errores de modo que ya no se propaguen al paso siguiente en el proceso del software.

2 Si se considera la mejora en el proceso de software, un problema de calidad que se propaga desde una actividad del marco de trabajo del proceso (por ejemplo, modelado) hacia otra (por ejemplo, construcción) también se puede llamar "defecto", porque el problema se debió haber descubierto antes de que un producto de trabajo (por ejemplo, un modelo de diseño) se hubiese "liberado" hacia la actividad siguiente.

El objetivo principal de una RTF es encontrar los errores antes de que pasen a otra actividad de ingeniería del software o sean liberados al usuario final.

Varios estudios industriales (realizados por TRW, NEC y Mitre Corp., entre otros) indican que las actividades de diseño introducen entre 50 y 65 por ciento de los errores (y, a final de cuentas, de los defectos) durante el proceso de software. Sin embargo, las técnicas de revisión formal han mostrado hasta 75 por ciento de efectividad [JON86] al descubrir fallos en el diseño. Al detectar y eliminar un gran porcentaje de dichos errores, el proceso de revisión reduce sustancialmente el costo de las actividades subsecuentes en el proceso de software.

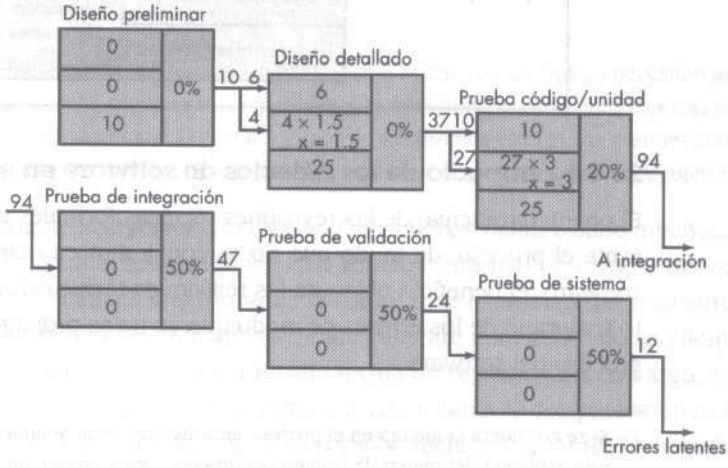
Para ilustrar el impacto en el costo de la detección temprana de errores, considérese una serie de costos relativos que se basan en datos de costo real recopilados para grandes proyectos de software [IBM81].[3] Supóngase que la corrección de un error descubierto durante el diseño costará 1.0 unidad monetaria. En relación con este costo, el mismo error descubierto justo antes de que comience el periodo de pruebas costará 6.5 unidades; durante las pruebas, 15 unidades; y después de la liberación, entre 60 y 100 unidades.

26.3.2 Amplificación y eliminación del defecto

Se puede usar un modelo de amplificación de defectos [IBM81] para ilustrar la generación y detección de errores durante el diseño preliminar, el diseño de detalles y los pasos de codificación de un proceso de ingeniería del software. En la figura 26.2 se ilustra esquemáticamente el modelo. Un recuadro representa un paso de desarrollo del software. Durante el paso, los errores se pueden generar de manera inadvertida. La revisión puede fallar en descubrir errores generados de manera reciente y errores de pasos anteriores, lo que resulta en cierto número de errores que se pasan por alto. En algunos casos, los errores que se pasan por alto desde pasos anteriores se am-

FIGURA 26.3

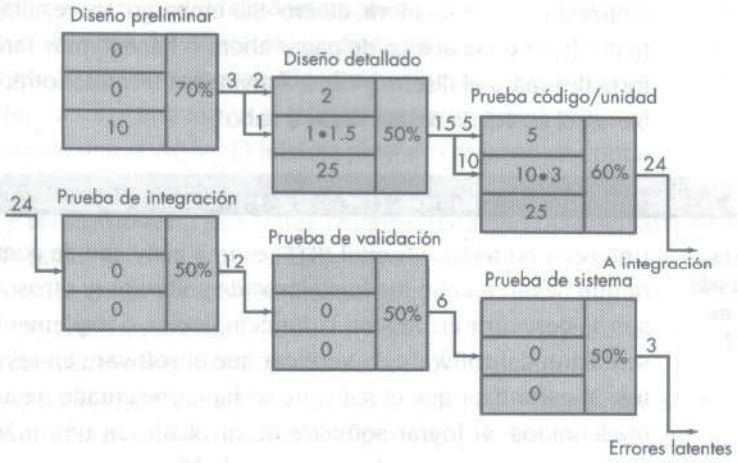
Amplificación de defecto sin revisiones.



3 Aunque estos datos tienen más de 20 años de antigüedad, aún son aplicables en un contexto moderno.

FIGURA 26.4

Amplificación de defecto con revisiones.



plifican (factor de amplificación *x*) con el trabajo actual. Las subdivisiones de los recuadros representan cada una de estas características y el porcentaje de eficiencia de la detección de errores, una función de la minuciosidad de la revisión.

> "Algunas enfermedades, dicen los médicos, son fáciles de curar en sus inicios aunque difíciles de reconocer... pero en el transcurso del tiempo, cuando no han sido reconocidas a primera vista y tratadas, se vuelven fáciles de reconocer pero difíciles de curar."
>
> **Nicolás Maquiavelo**

La figura 26.3 ilustra un ejemplo hipotético de la amplificación del defecto para un proceso de software en el que no se llevan a cabo revisiones. En la figura se supone que cada paso de prueba descubre y corrige, sin introducir nuevos errores, 50 por ciento de los errores que llegan (una suposición optimista). Diez defectos de diseño preliminar se amplificaron a 94 errores antes de comenzar las pruebas. Doce defectos latentes se liberaron al campo. La figura 26.4 considera las mismas condiciones excepto que la revisión del diseño y del código se llevaron a cabo como parte de cada paso de desarrollo. En este caso, diez errores iniciales de diseño preliminar se amplificaron a 24 errores antes de comenzar el periodo de pruebas. Sólo existen tres defectos latentes. Al considerar los costos relativos asociados con el descubrimiento y la corrección de errores se puede establecer el costo global (con revisión y sin ella para el ejemplo hipotético). El número de errores descubiertos durante cada uno de los pasos anotados en las figuras 26.3 y 26.4 se multiplica por el costo para eliminar un error (1.5 unidades de costo para diseño, 6.5 unidades de costo antes de las pruebas, 15 unidades de costo durante las pruebas, y 67 unidades de costo después de la liberación). Empleando estos datos, el costo total para desarrollo y mantenimiento es de 783 unidades cuando se realizan revisiones. Si no se realizan revisiones el costo total es de 2 177 unidades, casi tres veces más costoso.

En las revisiones un ingeniero de software debe utilizar tiempo y esfuerzo, y la organización desarrolladora, dinero. Sin embargo, los resultados del ejemplo precedente no dejan duda acerca de pagar ahora o hacerlo más tarde. Las revisiones técnicas formales (para el diseño y otras actividades técnicas) ofrecen un beneficio demostrable en el costo. Se deben llevar a cabo.

26.4 REVISIONES TÉCNICAS FORMALES

? Cuando se llevan a cabo RTF, ¿cuáles son los objetivos?

Una revisión técnica formal (RTF) es una actividad de control de calidad del software que llevan a cabo los ingenieros de software (y otros). Los objetivos de una RTF son 1) descubrir errores en la función, lógica o implementación en cualquier representación del software; 2) verificar que el software en revisión satisface sus requisitos; 3) garantizar que el software se ha representado de acuerdo con los estándares predefinidos; 4) lograr software desarrollado en una manera uniforme; y 5) hacer proyectos más manejables. Además, la RTF sirve como un campo de entrenamiento, pues permite que los ingenieros menos experimentados observen diferentes enfoques respecto del análisis, el diseño y la construcción del software. La RTF también funge como promotora del soporte y la continuidad, pues varias personas se familiarizan con las partes del software que de otra forma nunca verían.

> "No hay urgencia más grande que la que tiene un hombre por corregir el trabajo de otro."
>
> **Mark Twain**

La RTF es en realidad una clase de revisión que incluye recorridos, inspecciones, revisiones cíclicas y otro pequeño grupo de evaluaciones técnicas de software. Cada RTF se realiza en una junta y tendrá éxito sólo si se planifica, controla y atiende apropiadamente. En las siguientes secciones se presentan directrices similares a las de un recorrido (por ejemplo, [FRE90], [GIL93]) que se presenta como una revisión técnica formal representativa.

26.4.1 La junta de revisión

Sin importar el formato de RTF que se elija, cualquier junta de revisión debe atenerse a las siguientes restricciones:

- En la revisión se deben involucrar (usualmente) entre tres y cinco personas.
- Se debe preparar con anticipación, pero sin que requiera más de dos horas de trabajo de cada persona.
- La duración de la junta de revisión debe ser menor a dos horas.

Referencia Web

El *Formal Inspection Guidebook* (Libro guía de inspección formal) de NASA SATC se puede descargar de **satc.gsfc.nasa.gov/fi/fipage.html.**

Dadas estas restricciones, debe ser obvio que una RTF se enfoca en una parte específica (y pequeña) del software total. Por ejemplo, más que intentar revisar un diseño completo, se llevan a cabo recorridos para cada componente o grupo pequeño de componentes. Al estrechar el enfoque, la RTF tiene una mayor probabilidad de descubrir errores.

Una RTF se enfoca en una porción relativamente pequeña de un producto de trabajo.

El enfoque de la RTF se dirige a un producto de trabajo (por ejemplo, una porción de una especificación de requisitos, un diseño detallado de componente, una lista de código fuente de un componente). El individuo que ha desarrollado el producto de trabajo —el *productor*— le informa al jefe del proyecto que el producto está completo y que se requiere una revisión. El jefe del proyecto se pone en contacto con un *jefe de revisión*, quien evalúa la disponibilidad del producto, genera copias del material del producto y las distribuye a dos o tres *revisores* para que realicen sus observaciones antes de la junta. Se espera que cada revisor emplee entre una y dos horas en revisar el producto, tomar notas y familiarizarse con el trabajo. Al mismo tiempo, el jefe de revisión también revisa el producto y establece una agenda para la junta de revisión, la que usualmente se programa para el día siguiente.

En algunas situaciones es buena idea hacer que alguien distinto al productor recorra el producto que experimenta revisión. Esto conduce a una interpretación literal del producto de trabajo y a un mejor reconocimiento de los errores.

A la junta de revisión asisten el jefe de revisión, todos los revisores y el productor. Uno de los revisores asume el papel de *registrador*; es decir, el individuo que registra (por escrito) los temas importantes que surjan durante la revisión. La RTF comienza con una presentación de la agenda y una breve introducción a cargo del productor. Entonces el productor procede a recorrer el producto de trabajo y explica el material, mientras que los revisores exponen los problemas que descubrieron antes de la junta. Cuando se descubren problemas o errores válidos el registrador anota cada uno.

Al final, todos los asistentes a la RTF deben decidir si 1) aceptan el producto sin modificaciones posteriores, 2) rechazan el producto debido a errores severos (una vez corregidos se tiene que realizar otra revisión) o 3) aceptan el producto provisionalmente (se encontraron errores menores que es necesario corregir, pero no se requerirá revisión adicional). Cuando se tome la decisión, todos los asistentes a la RTF llenan un documento de finalización en el que indican su participación en la revisión y su conformidad con los hallazgos del equipo revisor.

26.4.2 Informe de la revisión y conservación de registros

Durante la RTF, un revisor (el registrador) registra activamente todos los problemas que hayan surgido. Éstos se resumen al final de la junta de revisión y se genera una *lista de problemas de revisión*. Además, se llena un *informe resumen de la revisión técnica formal*. Un informe resumen de la revisión responde tres preguntas:

1. ¿Qué se revisó?
2. ¿Quién lo revisó?
3. ¿Cuáles fueron los hallazgos y conclusiones?

El informe resumen de la revisión es un formato de una sola página (con posibles anexos). Se vuelve parte del registro histórico del proyecto y es posible distribuirlo entre el jefe del proyecto y otras partes interesadas.

La lista de problemas de la revisión cumple dos propósitos: 1) identificar áreas problema en el producto y 2) funcionar como lista de verificación de elementos de acción que guían al productor conforme se hacen las correcciones. Normalmente al informe resumen se anexa una lista de problemas.

Es importante establecer un procedimiento de seguimiento para garantizar que los elementos en la lista de problemas se han corregido adecuadamente. A menos que esto se haga, es posible que los problemas surgidos "caigan entre las grietas". Un enfoque consiste en asignar la responsabilidad del seguimiento al jefe de revisión.

> "A menudo una reunión es un suceso en el que se toman los minutos y se pierden las horas."
>
> **Anónimo**

26.4.3 Directrices de la revisión

Las directrices para realizar las revisiones técnicas formales es necesario establecerlas con anticipación, distribuirlas entre todos los revisores, suscribirlas y luego seguirlas. Una revisión descontrolada usualmente es peor que carecer de una. Las siguientes representan un conjunto mínimo de directrices para las revisiones técnicas formales:

No señale los errores de manera hiriente. Una forma de ser gentil es preguntar algo que permita al productor descubrir el error.

1. *Revisar el producto, no al productor.* Una RTF involucra personas y egos. Realizada con propiedad, la RTF debe dejar a todos los participantes con un cálido sentimiento de logro. Si se lleva a cabo de manera inadecuada, la RTF puede tomar un aura inquisitorial. Los errores se deben señalar con gentileza; el tono de la junta debe ser relajado y constructivo; la finalidad no debe ser avergonzar o menospreciar.
2. *Establecer una agenda y respetarla.* Un mal clave de las juntas de cualquier tipo es la divagación. Una RTF tiene que mantener el rumbo y seguir el programa. El jefe de revisión tiene la responsabilidad de mantener el programa de la junta y no vacilar en llamar la atención de la gente cuando se empiece a divagar.
3. *Limitar el debate y la impugnación.* Cuando un revisor plantee un problema, tal vez no haya un acuerdo universal sobre su impacto. En lugar de perder tiempo debatiendo la cuestión, el problema se debe registrar para tratarlo informalmente después.
4. *Enunciar áreas de problemas, pero no se intente resolver todos los que se hayan señalado.* Una revisión no es una sesión para resolver problemas. Esto se debe posponer hasta después de la junta de revisión.
5. *Tomar notas.* En ocasiones es buena idea que el registrador tome notas en una pizarra, de modo que las palabras y las prioridades puedan evaluarlas otros revisores conforme se registra la información.
6. *Limitar el número de participantes e insistir en la preparación anticipada.* Dos cabezas piensan mejor que una, pero 14 no necesariamente son mejores que cuatro. Manténgase el número de personas involucradas en el mínimo necesario. Sin embargo, todos los miembros del equipo revisor deben prepararse por anticipado. El jefe de revisión debe solicitar comentarios escritos (lo que ofrece un indicio de que el revisor ha analizado el material).

7. *Desarrollar una lista de verificación para cada producto que tenga probabilidad de ser revisado.* Una lista de verificación ayuda al jefe de revisión a estructurar la junta de RTF, y a cada revisor a enfocarse en los problemas importantes.
8. *Asignar recursos y programar las RTF.* Las revisiones serán efectivas si se programan como una tarea del proceso de software. Además, se debe programar tiempo para realizar las inevitables modificaciones que ocurrirán como resultado de una RTF.
9. *Realizar un entrenamiento significativo de todos los revisores.* Los participantes en la revisión serán eficientes si reciben algún entrenamiento formal. El entrenamiento debe subrayar tanto los problemas relacionados con el proceso como el lado psicológico y humano de las revisiones.
10. *Analizar las revisiones previas.* La junta es beneficiosa para descubrir problemas en el proceso de revisión mismo. El primer producto que se haya revisado debe establecer las directrices de revisión.

> "Una de las compensaciones más hermosas de la vida es que ningún hombre puede intentar sinceramente ayudar a otro sin ayudarse a sí mismo."
>
> **Ralph Waldo Emerson**

Puesto que muchas variables (por ejemplo, número de participantes, tipo de productos de trabajo, tiempo y duración, enfoque específico de revisión) inciden en una revisión provechosa, una organización de software debe experimentar para determinar qué enfoque funciona mejor en un contexto local. Porter y sus colegas [POR95] ofrecen una excelente guía para este tipo de experimentación.

26.4.4 Revisiones basadas en muestras

En un contexto ideal, cada producto de trabajo de ingeniería del software debería experimentar una revisión técnica formal. En el mundo real de los proyectos de software, los recursos son limitados y el tiempo es corto. Como consecuencia, usualmente las revisiones se soslayan, aunque se reconozca su valor como mecanismo de control de calidad. Thelin y sus colegas [THE01] abordan este tema cuando afirman:

> Las inspecciones [RTF] sólo son vistas como eficientes si se encuentran muchas fallas durante su búsqueda. Si en los artefactos [productos de trabajo] se encuentran muchas fallas, las inspecciones son necesarias. Si, por otra parte, sólo se encuentran pocas fallas, la inspección ha sido una pérdida de tiempo para muchas personas involucradas en la tarea.[4] Más aún, los proyectos de software que están atrasados con frecuencia disminuyen el tiempo de las actividades de inspección, lo que conduce a una falta de calidad. Una so-

4 Desde luego, se puede argumentar que, al llevar a cabo revisiones, se alienta a los productores a enfocarse en la calidad, incluso si no se encuentran errores.

lución sería asignar jerarquías a los recursos para las actividades de inspección y, en consecuencia, concentrar los recursos disponibles en los artefactos más proclives a las fallas.

Las revisiones toman tiempo, pero es tiempo bien empleado. Sin embargo, si el tiempo es corto y no se tiene otra opción, no se dispensen las revisiones. En su lugar utilícense revisiones basadas en muestras.

Thelin y sus colegas sugieren un proceso de revisión basado en que muestras de todos los productos de trabajo de ingeniería del software, éstas se inspeccionan para determinar qué productos de trabajo son más proclives al error. Entonces los recursos de las RTF completas se enfocan sólo en aquellos productos de trabajo con probabilidad (basándose en los datos recopilados durante el muestreo) de ser proclives al error.

Para ser eficaz, el proceso de revisión basado en muestras debe intentar cuantificar aquellos productos de trabajo que sean objetivos principales para las RTF completas. Para lograrlo se sugieren los siguientes pasos [THE01]:

1. Inspeccionar una fracción a_i de cada producto de trabajo de software i. Registre el número de fallas f_i encontradas dentro de a_i.
2. Desarrollar una estimación bruta del número de fallas dentro del producto de trabajo i al multiplicar f_i por $1/a_i$.
3. Ordenar los productos de trabajo en forma descendente de acuerdo con la estimación bruta del número de fallas en cada uno.
4. Enfocar los recursos de revisión disponibles en aquellos productos de trabajo con el mayor número estimado de fallas.

La fracción con la que se ha hecho un muestreo del producto de trabajo debe 1) ser representativa del producto de trabajo como un todo, y 2) ser lo suficientemente grande de tal manera que sea significativa para los revisores que realicen el muestreo. Conforme a_i aumenta, la probabilidad de que la muestra sea una representación válida del producto de trabajo también crece. Sin embargo, también aumentan los recursos requeridos para levantar la muestra. Un equipo de ingeniería del software debe establecer el mejor valor para a_i según los tipos de productos de trabajo producidos.[5]

HOGARSEGURO

Problemas en el SQA

El escenario: Oficina de Doug Miller cuando comienza el proyecto de software *HogarSeguro*.

Los actores: Doug Miller (gerente del equipo de ingeniería del software *HogarSeguro*) y otros miembros del equipo de ingeniería del software.

La conversación:

Doug: Ya sé que no hemos empleado tiempo para desarrollar un plan de SQA para este proyecto, pero ya estamos en él y tenemos que considerar la calidad... ¿cierto?

5. Thelin y sus colegas han realizado una simulación detallada que puede ayudar a tomar esta determinación. Véase [THE01] para detalles.

Jamie: Claro. Ya hemos decidido que, conforme desarrollemos el modelo de requisitos [capítulos 7 y 8], Ed se ha comprometido a desarrollar un procedimiento V&V para cada requisito.

Doug: Eso es muy bueno, pero no vamos a esperar hasta hacer las pruebas para evaluar la calidad, ¿o sí?

Vinod: ¡No! Desde luego que no. Hemos programado revisiones en el plan del proyecto para este incremento de software. Comenzaremos el control de calidad con las revisiones.

Jamie: Estoy un poco preocupada de que no tengamos tiempo suficiente para realizar todas las revisiones. De hecho, sé que no podremos.

Doug: Mmmmm. ¿Qué propones?

Jamie: Sugiero que seleccionemos aquellos elementos de los modelos de análisis y diseño cruciales para *HogarSeguro* y que los revisemos.

Vinod: ¿Pero qué ocurre si perdemos algo en una parte del modelo que no revisemos?

Shakira: Leí algo acerca de una técnica de muestreo [sección 26.4.4] que puede ayudarnos a seleccionar los candidatos para revisión. (Shakira explica el enfoque.)

Jamie: Tal vez... pero no estoy segura de que incluso tengamos tiempo para tomar muestras de cada elemento de los modelos.

Vinod: ¿Qué quieres que hagamos, Doug?

Doug: Robemos algo de Programación Extrema [capítulo 4]. Desarrollaremos los elementos de cada modelo en pares —dos personas— y realizaremos una revisión informal de cada uno conforme avancemos. Luego seleccionaremos los elementos "cruciales" para una revisión en equipo más formal, pero conservaremos dichas revisiones en un mínimo. De esa forma, todo será observado por más de un conjunto de ojos, pero aún así conservaremos nuestras fechas de entrega.

Jamie: Eso significa que tendremos que revisar la calendarización.

Doug: Así debe ser. La calidad triunfa sobre la calendarización en este proyecto.

26.5 Enfoques formales acerca del SQA

Durante las dos décadas pasadas, un pequeño, pero ruidoso, segmento de la comunidad de ingeniería del software ha argumentado que se requiere un enfoque más formal de la garantía de la calidad del software. Se puede argumentar que un programa de computadora es un objeto matemático [SOM01]. En cada lenguaje de programación se definen una sintaxis y una semántica rigurosas, y existe un enfoque riguroso respecto de la especificación de requisitos de software (capítulo 28). Si el modelo de requisitos (especificación) y el lenguaje de programación se representan en una forma rigurosa, debe ser posible aplicar pruebas matemáticas de exactitud para demostrar que un programa concuerda exactamente con sus especificaciones.

Los intentos encaminados a probar la exactitud de los programas (capítulos 28 y 29) no son nuevos. Dijkstra [DIJ76] y Linger, Mills y Witt [LIN79], entre otros, aconsejaron pruebas de exactitud de programa y los vincularon con la aplicación de conceptos de programación estructurada (capítulo 11).

26.6 Garantía de la calidad estadística del software

La garantía de la calidad estadística refleja una tendencia, creciente en la industria, por adoptar un enfoque más cuantitativo acerca de la calidad. Para el software, la garantía de la calidad estadística implica los pasos siguientes:

¿Qué pasos se requieren para realizar SQA estadístico?

1. La información acerca de los defectos de software se recopila y clasifica.
2. Se intenta determinar la causa subyacente de cada defecto (por ejemplo, falta de concordancia con las especificaciones, errores de diseño, violación de estándares, deficiente comunicación con el cliente).
3. Mediante el principio de Pareto (80 por ciento de los defectos se encuentra en 20 por ciento de todas las causas posibles) se aísla un 20 por ciento (los "vitales").
4. Una vez que las causas vitales han sido identificadas, se corrigen los problemas que han provocado los defectos.

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

> "20 por ciento del código tiene 80 por ciento de los errores. ¡Encuéntrelos, corríjalos!"
>
> **Lowell Arthur**

26.6.1 Un ejemplo genérico

Para ilustrar la aplicación de los métodos estadísticos en el trabajo de ingeniería del software, supóngase que una organización de ingeniería del software recopila información acerca de defectos durante un año. Algunos de los defectos se descubren cuando el software está en desarrollo; otros, después de que se ha liberado entre sus usuarios finales. Aunque se descubren cientos de diferentes defectos, todos tienen una (o más) de las causas siguientes:

- Especificaciones incompletas o erróneas (EIE).
- Mala interpretación de la comunicación del cliente (MCC).
- Desviación intencional de las especificaciones (DIE).
- Violación de los estándares de programación (VEP).
- Errores en la representación de los datos (ERD).
- Interfaz de componente inconsistente (ICI).
- Error en la lógica del diseño (ELD).
- Prueba incompleta o errónea (PIE).
- Documentación imprecisa o incompleta (DII).
- Error en la traducción del diseño al lenguaje de programación (TLP).
- Interfaz hombre-computadora ambigua o inconsistente (IHC).
- Misceláneo (MIS).

Para aplicar el aseguramiento de la calidad estadística del software se construye la tabla de la figura 26.5. La tabla indica que EIE, MCC y ERD son las causas vitales que

FIGURA 26.5 Recolección de datos para SQA estadístico.

	Total		Serios		Moderados		Menores	
Error	Núm	%	Núm	%	Núm	%	Núm	%
EIE	205	22%	34	27%	68	18%	103	24%
MCC	156	17%	12	9%	68	18%	76	17%
DIE	48	5%	1	1%	24	6%	23	5%
VEP	25	3%	0	0%	15	4%	10	2%
ERD	130	14%	26	20%	68	18%	36	8%
ICI	58	6%	9	7%	18	5%	31	7%
ELD	45	5%	14	11%	12	3%	19	4%
PIE	95	10%	12	9%	35	9%	48	11%
DII	36	4%	2	2%	20	5%	14	3%
TLP	60	6%	15	12%	19	5%	26	6%
IHC	28	3%	3	2%	17	4%	8	2%
MIS	56	6%	0	0%	15	4%	41	9%
Totales	942	100%	128	100%	379	100%	435	100%

explican el 53 por ciento de todos los errores. Sin embargo, se debe observar que EIE, ERD, TLP y ELD se seleccionarían como las causas vitales si sólo se consideraran los errores serios. Una vez determinadas las causas vitales, la organización de ingeniería del software puede comenzar la acción correctiva. Por ejemplo, para corregir MCC, el desarrollador de software puede implementar técnicas que faciliten la recopilación de requisitos (capítulo 7) para mejorar la calidad de la comunicación y las especificaciones del cliente. Para mejorar ERD, el desarrollador puede adquirir herramientas para el modelado de datos y ejecutar revisiones de diseño de datos más rigurosas.

Es importante anotar que la acción correctiva se enfoca principalmente en las vitales. Conforme éstas se corrigen, nuevas candidatas ocupan la parte superior de la clasificación.

Las técnicas de garantía estadística de la calidad para software han demostrado que ofrecen una mejora sustancial en la calidad [ART97]. En algunos casos, las organizaciones de software han alcanzado 50 por ciento de reducción anual en los defectos después de aplicar estas técnicas.

La aplicación del SQA estadístico y el principio de Pareto se pueden resumir en una sola oración: *Emplee su tiempo enfocándose en las cosas que realmente importan, ¡pero primero asegúrese de entender qué es lo que realmente importa!*

Un detallado análisis del SQA estadístico está más allá del ámbito de este libro. Los lectores interesados deben consultar [GOH02], [SCH98] o [KAN95].

26.6.2 Seis sigma para ingeniería del software

Seis sigma es la estrategia más ampliamente empleada en la actualidad para el aseguramiento de la calidad estadístico en la industria. Originalmente popularizada por

Motorola en el decenio de 1980, la estrategia seis sigma "es una metodología rigurosa y disciplinada que utiliza análisis de datos y estadístico para medir y mejorar el desempeño operativo de una compañía al identificar y eliminar los 'defectos' en la fabricación y los procesos relacionados con el servicio" [ISI03]. El término "seis sigma" se deriva de seis desviaciones estándar —3.4 instancias (defectos) por millón de ocurrencias—, lo que implica un estándar de calidad extremadamente elevado. La metodología seis sigma define tres pasos centrales:

¿Cuáles son los pasos centrales de la metodología seis sigma?

- *Definir* los requisitos del cliente, entregables y metas del proyecto por medio de métodos bien definidos de comunicación con el cliente.
- *Medir* el proceso existente y su salida para determinar el desempeño de calidad actual (recopilación de métricas de defecto).
- *Analizar* las métricas de defecto y determinar las causas poco vitales.

Si un proceso de software existente está en marcha, pero se requiere mejoría, seis sigma sugiere dos pasos adicionales:

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

Estos pasos centrales y adicionales a veces se conocen como el método DMAMC (definir, medir, analizar, mejorar y controlar).

Si una organización está desarrollando un proceso de software (en lugar de mejorar un proceso existente), los pasos centrales se aumentan de la siguiente manera:

- *Diseñar* el proceso para 1) evitar las causas originales de los defectos y 2) satisfacer los requisitos del cliente.
- *Verificar* que el modelo de proceso, de hecho, evitará los defectos y satisfará los requisitos del cliente.

A esta variación a veces se le llama método DMADV (definir, medir, analizar, diseñar y verificar).

Una exposición detallada de seis sigma se encuentra en las fuentes bibliográficas dedicadas a la materia. El lector interesado debe consultar [ISI03], [SNE03] y [PAN00].

26.7 FIABILIDAD DEL SOFTWARE

La fiabilidad del software, a diferencia de otros factores de calidad, se puede medir, dirigir y estimar empleando datos históricos y de desarrollo. La *fiabilidad del software* se define en términos estadísticos como "la probabilidad de la operación libre de fallas de un programa de computadora en un entorno específico durante un tiempo específico" [MUS87]. Con fines ilustrativos, se estima que el programa *X* tiene una

fiabilidad de 0.96 durante un periodo de ocho horas de procesamiento. En otras palabras, si el programa *X* fuese ejecutado 100 veces y requiriese un total de ocho horas de tiempo de procesamiento (tiempo de ejecución), es probable que operaría correctamente (sin falla) 96 veces.

> "El precio inevitable de la fiabilidad es la simplicidad."
>
> C.A.R. Hoare

Siempre que se estudia la fiabilidad del software, surge una pregunta central: ¿qué significa el término *falla*? En el contexto de cualquier análisis de calidad y fiabilidad del software, la falla es la falta de concordancia con los requisitos del software. Sin embargo, incluso dentro de esta definición, existen gradientes. Las fallas sólo pueden ser molestas y catastróficas. Una falla puede corregirse en segundos, mientras que otra tal vez requiera semanas o incluso meses. Para complicar el tema aún más, la corrección de una falla puede, de hecho, resultar en la introducción de otros errores que a final de cuentas resultan en otras fallas.

26.7.1 Medidas de fiabilidad y disponibilidad

PUNTO CLAVE
Los problemas de fiabilidad del software casi siempre pueden seguirse a defectos en diseño o implementación.

Los primeros trabajos en la fiabilidad del software intentaron extrapolar las matemáticas de la teoría de fiabilidad del hardware (por ejemplo, [ALV64]) a la predicción de la fiabilidad del software. La mayoría de los modelos de fiabilidad relacionada con el hardware tratan acerca de las fallas debidas al uso más que a las que se deben a defectos de diseño. En el hardware, las fallas que se deben al uso físico (por ejemplo, los efectos de la temperatura, la corrosión, los choques eléctricos) son más probables que una falla relacionada con el diseño. Desdichadamente, lo opuesto es cierto para el software. De hecho, todas las fallas de software se originan en problemas de diseño o implementación, el uso (capítulo 1) no entra en el cuadro.

Ha habido debates acerca de la relación entre conceptos clave en la fiabilidad del hardware y su aplicabilidad al software (por ejemplo, [LIT89], [ROO90]). Aunque todavía se tiene que establecer un vínculo irrefutable, vale la pena considerar unos cuantos conceptos simples que se aplican a elementos de ambos sistemas.

PUNTO CLAVE
Es importante notar que el TMEF y las medidas relacionadas están basadas en tiempo de CPU, no en tiempo de reloj de pared.

Si se considera un sistema basado en computadora, una simple medida de fiabilidad es el *tiempo medio entre fallas* (TMEF), donde

$$\text{TMEF} = \text{TMDF} + \text{TMDR}$$

Las siglas TMDF y TMDR significan *tiempo medio de falla* y *tiempo medio de reparación*, respectivamente.[6]

Muchos investigadores argumentan que el TMEF es con mucho más fácil de medir que los defectos/KLDC o los defectos/PF. Establecido de manera simple, el usuario final está preocupado por las fallas, no por el conteo total de errores. Debido a

6 Aunque se pueda requerir la depuración (y correcciones relacionadas) como consecuencia de una falla, en muchos casos el software trabajará adecuadamente después de un reinicio sin otro cambio.

Algunos aspectos de la disponibilidad (no estudiados aquí) no tienen nada que ver con las fallas. Por ejemplo, los recortes en la calendarización (para funciones de soporte) provocan que el software no esté disponible.

que cada defecto contenido dentro de un programa no tiene la misma tasa de falla, la cuenta de defectos totales ofrece poca información de la fiabilidad del sistema.

Además de una medida de fiabilidad, se debe desarrollar una medida de disponibilidad. La *disponibilidad del software* es la probabilidad de que un programa opere de acuerdo con los requisitos en un punto dado del tiempo, y se define como

$$\text{Disponibilidad} = [\text{TMDF}/(\text{TMDF} + \text{TMDR})] \times 100\%$$

La medida de fiabilidad TMEF es igualmente sensible a TMDF y TMDR. La medida de disponibilidad es un poco más sensible a TMDR, y es una medida indirecta de la facilidad de mantenimiento del software.

26.7.2 Seguridad del software

La *seguridad del software* [LEV86] es una actividad de aseguramiento de la calidad del software que se enfoca en la identificación y evaluación de los peligros potenciales que pueden afectar negativamente al software y provocar una falla de todo el sistema. Si los peligros se pueden identificar temprano en el proceso de software, las características de diseño de software se pueden especificar de modo que eliminarán o controlarán los peligros potenciales.

> **"No puedo imaginar alguna condición que provoque que este barco se hunda. La industria naviera moderna ha ido más allá."**
>
> **E. I. Smith, capitán del *Titanic***

Como parte de la seguridad del software se llevan a cabo procesos de modelado y análisis. Inicialmente, los peligros se identifican y clasifican por importancia y riesgo. Por ejemplo, algunos de los peligros asociados con un control basado en computadora para la conducción de un automóvil pueden ser:

- Provoca aceleración descontrolada que no se puede detener.
- No responde a la presión del pedal de freno (al apagarlo).
- No responde cuando el interruptor se activa.
- Pierde o gana rapidez lentamente.

Una vez identificados estos peligros en el nivel del sistema, mediante técnicas de análisis se asignan severidad y probabilidad de ocurrencia.[7] Para ser eficaz, el software debe analizarse en el contexto de todo el sistema. Por ejemplo, un sutil error de entrada de usuario (las personas son componentes del sistema) tal vez lo magnifique una falla del software para producir datos de control que posicionan de manera inadecuada un dispositivo mecánico. Si se reúne un conjunto de condiciones am-

7 Este enfoque es similar a los métodos de análisis de riesgo descritos en el capítulo 25. La diferencia principal es el énfasis en los conflictos tecnológicos, más que en los tópicos relacionados con el proyecto.

Referencia Web

Una colección valiosa de ensayos acerca de seguridad de software se puede encontrar en **www.safeware-eng.com/**.

bientales externas (y sólo si ellas se reúnen), la posición inadecuada del dispositivo mecánico provocará una falla desastrosa. Las técnicas de análisis, como el análisis del árbol de fallas [VES81], la lógica de tiempo real [JAN86] o los modelos de red de Petri [LEV87], se emplean para predecir la cadena de eventos que pueden provocar peligros y la probabilidad de que cada uno de los eventos ocurrirá para crear la cadena.

Una vez identificados y analizados los peligros, se especifican los requisitos relacionados con la seguridad del software. Esto es, la especificación puede contener una lista de eventos indeseables y las respuestas deseables del sistema ante dichos eventos. Entonces se indica el papel del software en la gestión de los eventos indeseables.

Aunque la confiabilidad del software y su seguridad están estrechamente relacionadas, es importante entender las sutiles diferencias entre ellas. La confiabilidad del software utiliza análisis estadístico para determinar la probabilidad de que ocurrirá una falla del software. Sin embargo, el hecho de que ocurra una falla no necesariamente resulta en un peligro o percance. La seguridad del software examina las formas en las cuales las fallas resultan en condiciones que pueden conducir a un percance. Esto es, las fallas no son consideradas en el vacío, sino que se evalúan en el contexto de todo un sistema basado en computadora y en su entorno. Aquellos lectores con mayor interés deben remitirse al libro de Leveson [LEV95] para profundizar en el tema.

26.8 Los estándares de calidad ISO 9000[8]

PUNTO CLAVE

ISO 9000 describe lo que se debe hacer para ser manejable, pero no describe cómo se debe hacer.

Es posible definir un *sistema de garantía de la calidad* como la estructura organizacional, responsabilidades, procedimientos, procesos y recursos para implementar la gestión de la calidad [ANS87]. Los sistemas de garantía de calidad fueron creados para ayudar a las organizaciones a garantizar que sus productos y servicios satisfacen las expectativas de los clientes al cumplir sus especificaciones. El estándar ISO 9000 describe un sistema de garantía de la calidad en términos genéricos que se aplican a cualquier negocio sin importar los productos o servicios ofrecidos.

El registro en uno de los modelos de sistema de garantía de la calidad contenidos en ISO 9000 requiere que los sistemas y operaciones de calidad de una compañía los sometan a escrutinio auditores de una tercera entidad respecto de su concordancia con el estándar y de su funcionamiento eficaz. Antes del registro exitoso, los auditores le extienden a la compañía un certificado de la organización de registro que representan. Entrevistas de auditoría semianuales garantizan la concordancia continua con el estándar.

8 Esta sección, escrita por Michael Stovsky, se ha adaptado de "Fundamentals of ISO 9000", un libro de trabajo desarrollado por *Essential Software Engineering*, un video curriculum elaborado por R. S. Pressman & Associates, Inc. Reimpreso con permiso.

Referencia Web

Extensos vínculos hacia recursos ISO 9000/9001 se pueden encontrar en **www.tantara.ab.ca/info.htm.**

El estándar de garantía de la calidad que se aplica a la ingeniería del software es el ISO 9001:2000. El estándar contiene 20 requisitos que deben estar presentes para un sistema eficiente de garantía de la calidad. Puesto que el estándar ISO 9001:2000 es aplicable a todas las disciplinas de ingeniería, se ha desarrollado un conjunto especial de directrices ISO (ISO 9000-3) que ayudan a interpretar el estándar para emplearlo en el proceso de software.

Los requisitos que delinea ISO 9001:2000 abordan tópicos como responsabilidad de la gestión, sistema de calidad, revisión de contrato, control de diseño, control de documentos y datos, identificación y seguimiento de producto, control de proceso, inspección y pruebas, acciones correctivas y preventivas, control de registros de calidad, auditorías de calidad interna, entrenamiento, servicio y técnicas estadísticas. Una organización de software obtendrá el registro ISO 9001:2000 si establece políticas y procedimientos para abordar cada uno de los requisitos anotados líneas arriba (y otros) y, además, ser capaz de demostrar que se siguen dichas políticas y procedimientos. Para mayor información acerca de ISO 9001, el lector interesado debe consultar [HOY02], [GAA01] o [CIA01].

INFORMACIÓN

El estándar ISO 9001:2000

Las siguientes líneas generales definen los elementos básicos del estándar ISO 9001:2000. Información más amplia acerca del estándar se puede obtener de International Organization for Standardization (www.iso.ch) y en otras fuentes de Internet (por ejemplo, www.praxiom.com).

Establecer los elementos de un sistema de gestión de calidad.
- Desarrollar, implementar y mejorar el sistema.
- Definir una política que enfatice la importancia del sistema.

Documentar el sistema de calidad.
- Describir el proceso.
- Producir un manual operativo.
- Desarrollar métodos para controlar (actualizar) los documentos.
- Establecer métodos para la conservación de registros.

Soporte del control y la garantía de calidad.
- Promover la importancia de la calidad entre todos los participantes.
- Enfocarse en la satisfacción del cliente.
- Definir un plan de calidad que aborde objetivos, responsabilidades y autoridad.
- Definir mecanismos de comunicación entre los participantes.

Establecer mecanismos de revisión para el sistema de gestión de calidad.
- Identificar métodos de revisión y mecanismos de retroalimentación.
- Definir procedimientos de seguimiento.
- Identificar recursos de calidad que incluyan personal, entrenamiento, elementos de infraestructura.

Establecer mecanismos de control.
- Para planeación.
- Para requisitos del cliente.
- Para actividades técnicas (por ejemplo, análisis, diseño, pruebas).
- Para supervisión y gestión del proyecto.

Definir métodos para corrección.
- Valorar los datos y métricas de calidad.
- Definir enfoques para procesos continuos y mejora de la calidad.

26.9 El plan de SQA

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

En el IEEE [IEE94] se ha publicado un estándar para planes de SQA. El estándar recomienda una estructura que identifica: 1) el propósito y ámbito del plan; 2) una descripción de todos los productos de trabajo de ingeniería del software (por ejemplo, modelos, documentos, código fuente) que caen dentro del alcance del SQA; 3) todos los estándares y prácticas aplicables que se aprovechan durante el proceso de software; 4) acciones y tareas de SQA (incluso revisiones y auditorías) y su ubicación a través del proceso de software; 5) las herramientas y métodos que soportan las acciones y tareas de SQA; 6) procedimientos de gestión de configuración de software (capítulo 27) para gestionar el cambio; 7) métodos para ensamblar, salvaguardar y mantener todos los registros relacionados con el SQA; y 8) papeles y responsabilidades en la organización relativas a la calidad de producto.

Herramientas de software

Gestión de la calidad del software

Objetivo: El objetivo de las herramientas de SQA es auxiliar al equipo de proyecto para valorar y mejorar la calidad del producto de trabajo de software.

Mecánica: La mecánica de las herramientas varía. En general, la finalidad es valorar la calidad de un producto de trabajo específico. Nota: con frecuencia, dentro de la categoría de herramientas de SQA, se incluye una amplia variedad de herramientas de prueba de software (capítulos 13 y 14).

Herramientas representativas[9]

ARM, desarrollado por la NASA (satc.gsfc.nasa.gov/tools/index.html), ofrece medidas con las cuales se evalúa la calidad de un documento de requisitos de software.

QPR ProcessGuide and Scorecard, desarrollada por QPR Software (www.qpronline.com), ofrece soporte para seis sigma y otros enfoques de gestión de calidad.

Quality Tools Cookbook, desarrollado por Sytsma and Manley (www.sytsma.com/tqmtools/tqmtoolmenu.html), proporciona descripciones útiles de herramientas clásicas de gestión de calidad tales como los diagramas de control, diagramas de dispersión, diagramas de afinidad y diagramas de matriz.

Quality Tools and Templates, desarrolladas por iSixSigma (http://www.isixsigma.com/tt/), describe una amplia variedad de herramientas y métodos útiles para gestión de calidad.

TQM Tools, desarrollado por Bain & Company (www.bain.com), brinda descripciones útiles de una variedad de herramientas de gestión usadas por TQM y relacionadas con los métodos de gestión de calidad.

9 Las herramientas expuestas sólo representan una muestra de esta categoría. En la mayoría de los casos los nombres de las mismas son marcas registradas por sus respectivos desarrolladores.