

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

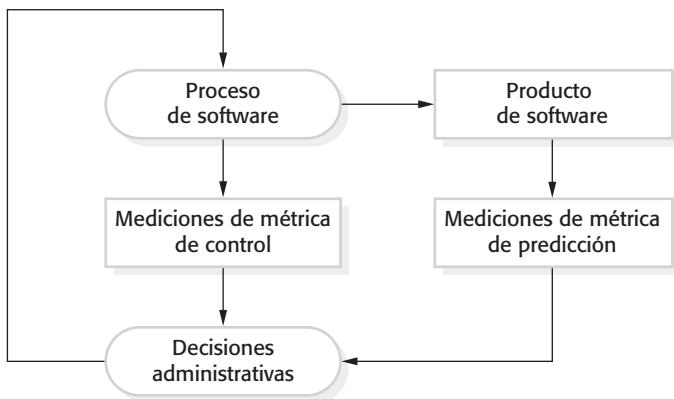


Figura 24.9 Mediciones de predicción y de control

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

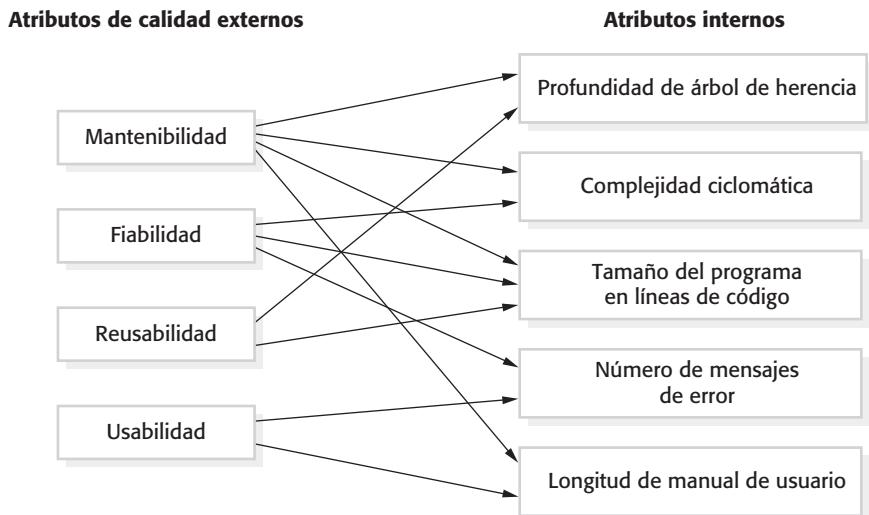


Figura 24.10
Relaciones entre software interno y externo

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

Figura 24.12 Suite de métricas CK orientadas a objetos

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

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

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

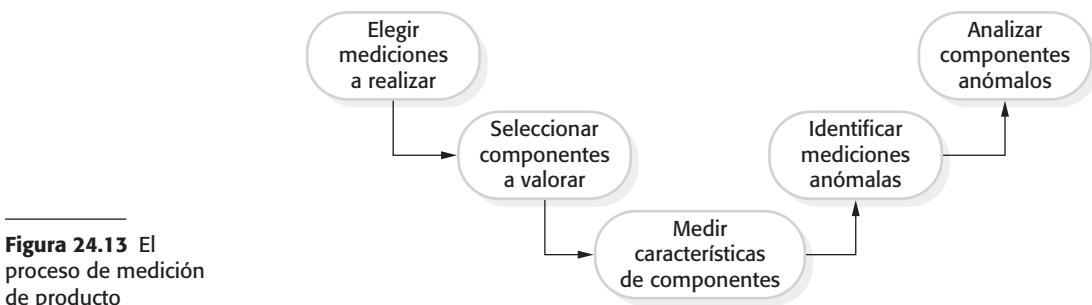


Figura 24.13 El proceso de medición de producto

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 anómalos 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.