TEMA 57

CALIDAD DEL "SOFTWARE". FACTORES Y MÉTRICAS. ESTRATEGIAS DE PRUEBA.

ÍNDICE

1. INTRODUCCIÓN

La **calidad del software** se define como: *concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente*.

La anterior definición sirve para hacer hincapié en tres puntos importantes:

1. Los requisitos del 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 se aplica la ingeniería del software. Si no se siguen esos criterios, casi siempre habrá falta de calidad.

3. Existe un conjunto de "requisitos implícitos" que a menudo no se mencionan (p. ej.: el deseo de un buen mantenimiento). Si el software se ajusta a sus requisitos explícitos pero falla en alcanzar los requisitos implícitos, la calidad del software queda en entredicho.

2. FACTORES QUE DETERMINAN LA CALIDAD DEL SOFTWARE

Los factores que afectan a la calidad del software se pueden clasificar en dos grandes grupos:

(1) Factores que pueden ser medidos directamente (p. ej.: errores/unidad de tiempo) y
(2) Factores que sólo pueden ser medidos indirectamente (p. ej.: facilidad de uso o de mantenimiento).

En cualquiera de los dos casos se puede "medir". Debemos comparar el software (documentos, programas, etc. ...) con alguna "referencia" y llegar a una indicación de la calidad.

McCall ha propuesto una útil clasificación de los factores que afectan a la calidad del software. Estos "factores de la calidad del software" se centran en tres aspectos importantes de un producto de software: *sus características operativas, su capacidad de soportar los cambios y su adaptabilidad a nuevos entornos.*

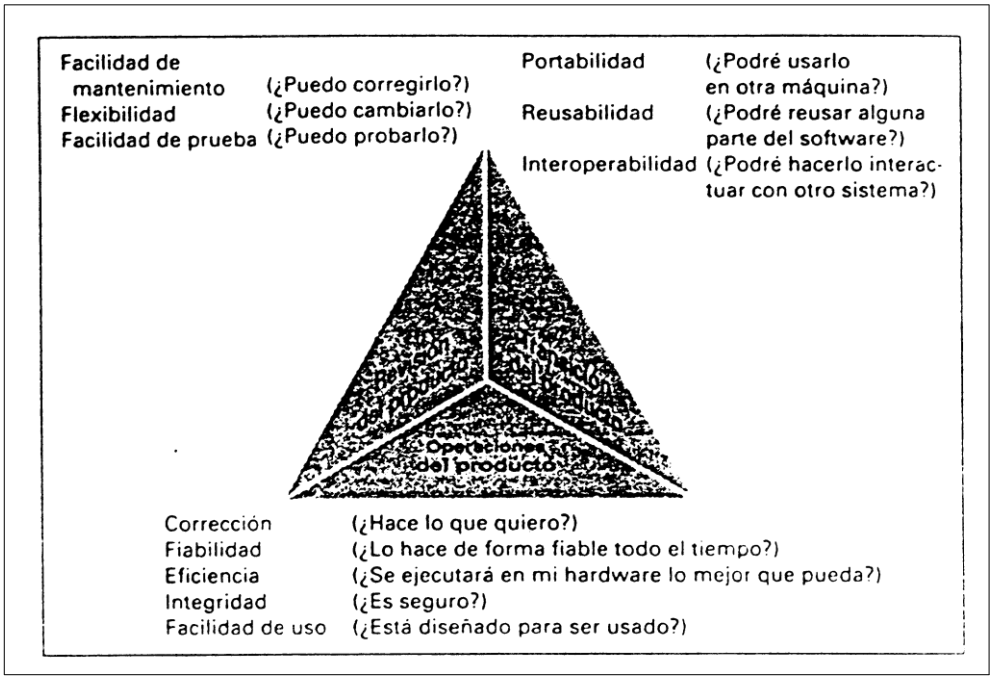

Figura 57.1. Factores de calidad del software de McCall

"Corrección". El grado en que un programa satisface sus especificaciones y consigue los objetivos de la misión encomendada por el cliente.

"Fiabilidad". El grado en que se puede esperar que un programa lleve a cabo sus funciones esperadas con la precisión requerida.

"Eficiencia". La cantidad de recursos de ordenador y de código requeridos por un programa para llevar a cabo sus funciones.

"Integridad". El grado en que puede controlarse el acceso al software o a los datos, por personal no autorizado.

"Facilidad de uso". El esfuerzo requerido par aprender un programa, trabajar con él, preparar su entrada e interpretar su salida.

"Facilidad de mantenimiento". El esfuerzo requerido para localizar y arreglar un error en un programa.

"Flexibilidad". El esfuerzo requerido para modificar un programa operativo.

"Facilidad de prueba". El esfuerzo requerido para probar un programa de forma que se asegure que realiza su función requerida.

"Portabilidad". El esfuerzo requerido para transferir el programa desde un hardware y/o un entorno de sistemas de software a otro.

"Reusabilidad". El grado en que un programa (o partes de un programa) se puede rehusar en otras aplicaciones. Esto va relacionado con el empaquetamiento y el alcance de las funciones que realiza el programa.

"Facilidad de interoperación". El esfuerzo requerido para interaccionar un sistema con otro.

3. MÉTRICAS DE LA CALIDAD DEL SOFTWARE

En esta sección veremos un conjunto de métricas del software que se pueden aplicar para garantizar cuantitativamente la calidad del software. En todos los casos, las métricas representan medidas indirectas, es decir, nunca medimos realmente la "calidad", sino algunas de sus manifestaciones. El factor que lo complica es la relación precisa entre la variable que es medida y la calidad del software.

3.1. Índices de calidad del software

El US Air Force System Command ha desarrollado una serie de indicadores de calidad del software basados en las características de diseño medibles para un programa de ordenador.

Utilizan información obtenida a partir del diseño arquitectónico y de datos, para obtener un "**índice de calidad de la estructura del diseño**" (**ICED**), que va de 0 a 1. Para calcular el ICED se tienen que averiguar los siguientes valores:

S1 = número total de módulos definidos en la arquitectura del programa.

S2 = número de módulos cuya correcta función depende de la fuente de los datos de entrada o que produce datos que se usan en cualquier parte [en general, los módulos de control (entre otros) no se van a considerar como parte de S2].

S3 = número de módulos cuya correcta función depende del procesamiento previo.

S4 = número de elementos de una base de datos (incluye los objetos de datos y todos los atributos que definen objetos).

S5 = número total de elementos de base de datos únicos.

S6 = número de segmentos de base de datos (registros diferentes u objetos individuales).

S7 = número de módulos con una sola entrada y una sola salida (el procesamiento de excepciones no se considera como una salida múltiple).

Una vez determinados los valores S1 a S7 para un programa de ordenador, se pueden calcular los siguientes *valores intermedios*:

"Estructura del programa": D1, que se define de la siguiente forma: Si el diseño arquitectónico se desarrolló usando un método característico (p. ej.: diseño orientado al flujo de datos o diseño orientado a los objetos), entonces D1 = 1; en caso contrario D1 = 0.

"Independencia de módulos": D2 = 1- (S2/S1).

"Módulos no dependientes del procesamiento previo": D3 = 1- (S3/S1).

"Tamaño de la base de datos": D4 = 1- (S5/S4). "Compartimentalización

de la base de datos": D5 = 1- (S6/S4). "Característica de entrada/salida

del módulo": D6 = 1- (S7/S1).

Habiendo determinado esos valores intermedios, el ICED se calcula de la siguiente manera:

$$\textbf{ICED} = \mathbf{P_i\ D_i}$$

Donde i variará de 1 a 9, P_i es el peso relativo de la importancia de cada uno de los valores intermedios y Sumatorio de $P_i = 1$ (si todos los D_i tienen el mismo peso, entonces $P_i = 0{,}167$).

Se puede determinar el valor del ICED para anteriores diseños y compararlo con un diseño que actualmente esté en desarrollo. Si el valor de ICED es significativamente menor que la media, significará que se va a requerir un posterior trabajo de diseño y de revisión. Igualmente, si hay que hacer cambios importantes en un diseño existente, se puede calcular el efecto de esos cambios en el ICED.

3.2. Índice de madurez del software

El estándar del IEEE 982.1-1988 sugiere un "**índice de madurez del software**" (**IMS**), que proporciona una indicación de la estabilidad de un producto de software (basada en los cambios que se producen en cada versión del producto). Se determina la siguiente información:

Mt = número de módulos en la versión actual.
Fm = número de módulos en la versión actual que han sido modificados.
Fa = número de módulos en la versión actual que han sido añadidos.
Fe = número de módulos de la versión anterior que se han eliminado en la versión actual.

El índice de madurez del software se calcula de la siguiente forma:

$$IMS = \frac{[Mt - (Fa + Fm + Fe)]}{Mt}$$

A medida que el IMS se aproxima a 1, el producto comienza a estabilizarse. El IMS también se puede utilizar como métrica para la planificación de actividades de mantenimiento del software. El tiempo medio para producir una versión de un producto de software puede tener correlación con el IMS y se pueden desarrollar modelos empíricos para el esfuerzo de mantenimiento.

4. ESTRATEGIAS DE PRUEBA

El *proceso de ingeniería del software* se puede ver como una espiral, como se ilustra en la Figura. Inicialmente, la ingeniería del sistema define el papel del software y conduce al análisis de los requisitos del software, donde se establece el campo de información, la función, el comportamiento, el rendimiento, las restricciones y los criterios de validación del software. Al movernos hacia el interior de la espiral, llegamos al diseño y, por último, a la codificación. Para desarrollar software de ordenador, damos vueltas en espiral a través de una serie de flujos o líneas que disminuyen el nivel de abstracción en cada vuelta.

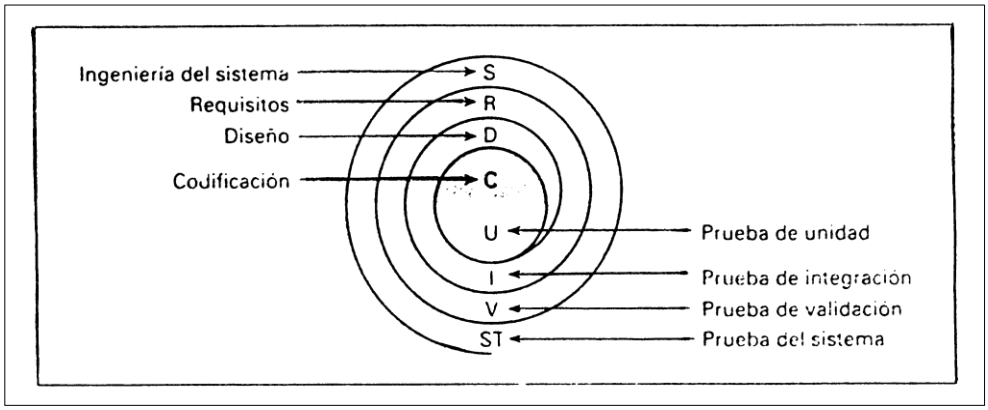

Figura 57.2. Estrategia de prueba

También podemos imaginar una **estrategia para la prueba** del software si nos movemos hacia fuera de la espiral de la Figura. La "*prueba de unidad*" comienza en el vértice de la espiral y se centra en cada unidad del software, tal como está implementada en código fuente. La prueba avanza, al movernos hacia afuera de la espiral, hasta llegar a la "*prueba de integración*", donde el foco de atención es el diseño y la construcción de la arquitectura del software. Dando otra vuelta por la espiral hacia afuera, encontramos la "*prueba de validación*", donde se validan los requisitos establecidos como parte del análisis de requisitos del software, comparándolos con el sistema que ha sido construido. Finalmente, llegamos a la "*prueba del sistema*", en la que se prueban como un todo el software y otros elementos del sistema. Para probar software de ordenador nos movemos hacia afuera por una espiral que, a cada vuelta, aumenta el alcance de la prueba.

4.1. Prueba de Unidad

La prueba de unidad centra el proceso de verificación en la menor unidad del diseño del software: el módulo.

Se prueba la "interfaz" del módulo para asegurar que la información fluye de forma adecuada hacia y desde la unidad del programa que está siendo probada. Se examinan las "estructuras de datos" locales para asegurar que los datos que se mantienen temporalmente conservan su integridad durante todos los pasos de ejecución del algoritmo. Se prueban las "condiciones límite" para asegurar que el módulo funciona correctamente en los límites establecidos como restricciones de procesamiento. Se ejercitan todos los "caminos independientes" (caminos básicos) de la estructura de control con el fin de asegurar que todas las sentencias del módulo se ejecutan por lo menos una vez. Y, finalmente, se prueban todos los "caminos de manejo de errores".

4.2. Prueba de Integración

La prueba de integración es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la integración. El objetivo es coger los módulos probados en unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño.

A menudo hay una tendencia a intentar una "integración no incremental"; o sea, a construir el programa mediante un enfoque del "big bang". Se combinan todos los módulos por anticipado. Se prueba todo el programa en conjunto. Normalmente se llega al caos.

La "integración incremental" es la antítesis del enfoque del "big bang". El programa se construye y se prueba en pequeños segmentos en los que los errores son más fáciles de aislar y de corregir, es más probable que se puedan probar completamente las interfaces y se puede aplicar un enfoque de prueba sistemática.

Integración descendente

La integración descendente es un planteamiento incremental a la construcción de la estructura de programas. Se integran los módulos moviéndose hacia abajo por la jerarquía de control, comenzando con el módulo de control principal (programa principal). Los módulos subordinados (de cualquier modo subordinados) al módulo de control principal se van incorporando en la estructura, bien de forma "primero en profundidad", bien "primero en anchura".

Integración ascendente

La prueba de integración ascendente, como su nombre implica, empieza la construcción y la prueba con los "módulos atómicos" (o sea, módulos de los niveles más bajos de la estructura del programa). Dado que los módulos se integran de abajo hacia arriba, el procesamiento requerido de los módulos subordinados siempre está disponible y se elimina la necesidad de resguardos.

Una estrategia de integración ascendente puede ser implementada mediante los siguientes *pasos*:

1. Se combinan los módulos de bajo nivel en "grupos" (a veces denominados "construcciones") que realicen una subfunción específica del software.

2. Se escribe un conductor (un programa de control de la prueba) para coordinar la entrada y la salida de los casos de prueba.

3. Se prueba el grupo.

4. Se eliminan los conductores y se combinan los grupos moviéndose hacia arriba por la estructura del programa.

4.3. Prueba de Validación

Tras la culminación de la prueba de integración, el software está completamente ensamblado como un paquete, se han encontrado y corregido los errores de interfaz y puede comenzar una serie final de pruebas del software: la "prueba de validación". La validación puede definirse de muchas formas, pero una simple indicación es que la validación se logra cuando el software funciona de acuerdo con las expectativas razonables del cliente.

Las expectativas razonables están definidas en la "especificación de requisitos del software", un documento que describe todos los atributos del software que son visibles al usuario. La "especificación" contiene una sección denominada "criterios de validación". La información contenida en esa Sección forma la base del enfoque a la prueba de validación.

Pruebas Alfa y Beta

Cuando se realiza software a medida para un cliente, se ejecutan una serie de "pruebas de aceptación" para permitir que el cliente valide todos los requisitos. Llevado a cabo por el usuario final en lugar del equipo de desarrollo, una prueba de aceptación puede ir desde un informal "paso de prueba" hasta la ejecución sistemática de una serie de pruebas bien planificadas. De hecho, la prueba de aceptación puede tener lugar a lo largo de semanas o meses, descubriendo así errores acumulados que pueden ir degradando el sistema.

Si el software se desarrolla como un producto que se va a usar por muchos clientes, no es práctico realizar pruebas de aceptación formales para cada uno de ellos. La mayoría de los constructores de productos de software llevan a cabo un proceso denominado "prueba alfa y beta" para descubrir errores que parezca que sólo el usuario final puede descubrir.

La **prueba alfa** es conducida por un cliente en el lugar de desarrollo. Se usa el software de forma natural, con el encargado del desarrollo "mirando por encima del hombro" del usuario y registrando errores y problemas de uso. Las pruebas alfa se llevan a cabo en un entorno controlado.

La **prueba beta** se lleva a cabo en uno o más lugares de clientes por los usuarios finales del software. A diferencia de la prueba alfa, el encargado del desarrollo, normalmente, no está presente. Así, la prueba beta es una aplicación "en vivo" del software en un entorno que no puede ser controlado por el equipo de desarrollo. El cliente registra todos los problemas (reales o imaginarios) que encuentra durante la prueba beta e informa a intervalos regulares al equipo de desarrollo. Como resultado de los problemas anotados durante la prueba beta, el equipo de desarrollo del software lleva a cabo modificaciones y así prepara una versión del producto de software para toda la base de clientes.

4.4. Prueba del Sistema

La prueba del sistema, está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejecutar profundamente el sistema basado en ordenador. Aunque cada prueba tiene un propósito distinto, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas. En las siguientes secciones discutimos los tipos de pruebas del sistema que merecen la pena para sistemas basados en software.

Prueba de recuperación

La "prueba de recuperación" es una prueba del sistema que refuerza el fallo del software de muchas formas y verifica que la recuperación se lleva a cabo apropiadamente. Si la recuperación es automática (llevada a cabo por el propio sistema) hay que evaluar la corrección de la reinicialización, de los mecanismos de recuperación del estado del sistema, de la recuperación de datos y del rearranque. Si la recuperación requiere la intervención humana, hay que evaluar los tiempos medios de reparación para determinar si están dentro de unos límites aceptables.

Prueba de seguridad

La "prueba de seguridad" intenta verificar que los mecanismos de protección incorporados en el sistema lo protegerán, de hecho, de la penetración impropia.

Durante la prueba de seguridad, el encargado de la prueba desempeña el papel de un individuo que desea penetrar en el sistema. Debe intentar hacerse con las claves de acceso por cualquier medio externo del oficio; puede atacar al sistema con software a medida, diseñado para romper cualquier defensa que haya construido; sabe bloquear el sistema, negando así el servicio a otras personas; debe producir a propósito errores del sistema, intentando penetrar durante la recuperación; o debe curiosear en los datos públicos, intentando encontrar la clave de acceso al sistema.

Prueba de resistencia

La prueba de resistencia ejecuta un sistema de forma que demande recursos en cantidad, frecuencia o volúmenes anormales. Por *ejemplo*: **(1)** diseñar pruebas especiales que generen diez interrupciones por

segundo, cuando las normales son una o dos; **(2)** incrementar las frecuencias de datos de entrada en un orden de magnitud con el fin de comprobar cómo responden las funciones de entrada; **(3)** ejecutar casos de prueba que requieran el máximo de memoria o de otros recursos; **(4)** diseñar casos de prueba que puedan dar problemas con el esquema de gestión de memoria virtual; **(5)** diseñar casos de prueba que produzcan excesivas búsquedas de datos resistentes en disco. Esencialmente, el encargado de la prueba intenta hacer que falle el programa.

Prueba de rendimiento

Para sistemas de tiempo real o sistemas empotrados, es inaceptable el software que proporciona las funciones requeridas pero que no se ajusta a los requisitos de rendimiento. La "prueba de rendimiento" está diseñada para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado. La prueba de rendimiento se da durante todos los pasos del proceso de prueba. Incluso al nivel de unidad, se debe asegurar el rendimiento de los módulos individuales.

5. BIBLIOGRAFÍA

Pressman, Roger S.
Ingeniería del Software
Mc Graw-Hill, 1993

López, Antonio
Metodologías de Desarrollo
Ra-ma, 1990