TEMA

50

Calidad y documentación en entornos gráficos.

elaborado por
EL EQUIPO DE PROFESORES
DEL CENTRO DOCUMENTACIÓN

1. CALIDAD DEL SOFTWARE Y GARANTÍA DE CALIDAD DEL SOFTWARE

Hasta el realizador de software más harto estará de acuerdo con que el software de alta calidad es una meta importante. Pero, ¿como definimos la calidad? Un bromista dijo una vez: "Cualquier programa hace algo bien, lo que puede pasar es que no sea lo que nosotros queremos que haga". En los libros se han propuesto muchas definiciones de la calidad del software.

Por lo que a nosotros respecta, 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.

No hay duda de que la anterior definición puede ser modificada o ampliada. De hecho, no tendría fin una discusión sobre una definición definitiva de la calidad del software. Para los propósitos de este capítulo, 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.

La calidad del software es una compleja mezcla de ciertos factores que varían para las diferentes aplicaciones y los clientes que las solicitan. En las siguientes secciones, se identifican los factores de la calidad del software y se describen las actividades humanas requeridas para alcanzarlos.

1.1. 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/KLDC[1]/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 en su libro "Factors in Software Quality" ha propuesto una útil clasificación de los factores que afectan a la calidad del software. Estos factores de calidad del software, que aparecen en la figura 50.1. 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.

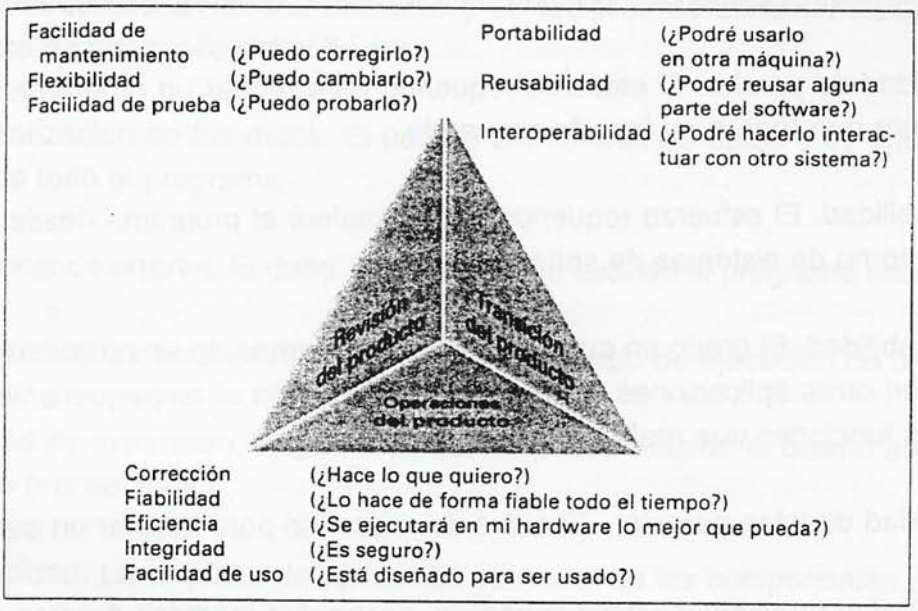

Figura 50.1. Factores de calidad de software de McCall

Para los factores indicados, McCall proporciona las siguientes descripciones:

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

[1] KLDC: kilo líneas de código.

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 computadora 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 para 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 reutilizar 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 acoplar un sistema a otro.

Es difícil, y en algunos casos imposible, desarrollar medidas directas de los anteriores factores de calidad. Por tanto, se define un conjunto de métricas usadas para desarrollar expresiones para cada uno de los factores de acuerdo con la siguiente relación:

$$F_c = c_1 \times m_1 + c_2 \times m_2 + \ldots + c_n \times m_n$$

donde F_c es un factor de calidad del software, c_i son coeficientes de regresión y m_i son las métricas que afectan al factor de calidad. Desgraciadamente, muchas de las métricas definidas por McCall sólo pueden ser medidas de forma subjetiva. Las métricas pueden estar en forma de listas de comprobaciones, usadas para "obtener el grado" de los atributos específi-

cos del software. El esquema de graduación propuesto por McCall va en una escala de 0 (bajo) a 10 (alto). En el esquema de graduación se usan las siguientes métricas:

Facilidad de auditoría. La facilidad con que se puede comprobar la conformidad con los estándares.

Exactitud. La precisión de los cálculos y del control.

Normalización de las comunicaciones. El grado en que se usan el ancho de banda, los protocolos y las interfaces estándar.

Completitud. El grado en que se ha conseguido la total implementación de las funciones requeridas.

Concisión. Lo compacto que es el programa en términos de líneas de código. Consistencia. El uso de un diseño uniforme y de técnicas de documentación a lo largo del proyecto de desarrollo del software.

Estandarización en los datos. El uso de estructuras de datos y de tipos estándar a lo largo de todo el programa.

Tolerancia de errores. El daño que se produce cuando el programa encuentra un error.

Eficiencia en la ejecución. El rendimiento en tiempo de ejecución de un programa.

Facilidad de expansión. El grado en que se puede ampliar el diseño arquitectónico, de datos o procedimental.

Generalidad. La amplitud de aplicación potencial de los componentes del programa.

Independencia del hardware. El grado en que el software es independiente del hardware sobre el que opera.

Instrumentación. El grado en que el programa muestra su propio funcionamiento e identifica errores que aparecen.

Modularidad. La independencia funcional de los componentes del programa.

Facilidad de operación. La facilidad de operación de un programa.

Seguridad. La disponibilidad de mecanismos que controlen o protejan los programas o los datos.

Autodocumentación. El grado en que el código fuente proporciona documentación significativa.

Simplicidad. El grado en que un programa puede ser entendido sin dificultad.

Independencia del sistema de software. El grado en que el programa es independiente de características no estándar del lenguaje de programación, de las características del sistema operativo y de otras restricciones del entorno.

Facilidad de traza. La posibilidad de seguir la pista a la representación del diseño o de los componentes reales del programa hacia atrás, hacia los requisitos.

Formación. El grado en que el software ayuda para permitir que nuevos usuarios apliquen el sistema.

Debe insistirse en que el peso dado a cada métrica dependerá de los productos particulares, así como de otros aspectos.

Los factores de calidad descritos por McCall representan una de tantas "listas de comprobación" que se han sugerido para la calidad del software. Hewlett-Packard ha desarrollado un conjunto de factores de calidad del software cuyas siglas son FURPS -por funcionalidad, facilidad de uso, fiabilidad, rendimiento y capacidad de soporte (en inglés). Los factores de calidad FURPS se han obtenido libremente de trabajos anteriores y se definen los siguientes atributos para cada uno de los cinco factores principales:

- La funcionalidad se obtiene mediante la evaluación del conjunto de características y de posibilidades del programa, la generalidad de las funciones que se entregan y la seguridad de todo el sistema.

- La facilidad de uso se calcula considerando los factores humanos, la estética global, la consistencia y la documentación.

- La fiabilidad se calcula midiendo la frecuencia de fallos y su importancia, la eficacia de los resultados de salida, el tiempo medio entre fallos (TMEF), la posibilidad de recuperarse a los fallos y la previsibilidad del programa.

• El rendimiento se mide mediante la evaluación de la velocidad de proceso, el tiempo de respuesta, el consumo de recursos, el rendimiento total de procesamiento y la eficiencia.

• La capacidad de soporte combina la posibilidad de ampliar el programa (extensibilidad), la adaptabilidad y la utilidad (estos tres atributos representan un término más común -facilidad de mantenimiento), además de la facilidad de prueba, la compatibilidad, la posibilidad de configuración [posibilidad de organizar y controlar elementos de la configuración del software], la facilidad con la que se puede instalar un sistema y la facilidad con la que se pueden localizar los problemas.

Los factores y atributos de calidad FURPS descritos anteriormente se pueden utilizar para establecer métricas de calidad en cada paso del proceso de ingeniería del software.

1.2. GARANTÍA DE CALIDAD DEL SOFTWARE

La garantía de calidad es una actividad esencial en cualquier empresa que produce productos que van a ser usados por otros. Antes del siglo veinte, la garantía de calidad era responsabilidad única de la persona que construía el producto. La primera función de control y de garantía de calidad formal fue introducida por los laboratorios Bell en 1916 y se extendió rápidamente por todo el mundo de las manufacturas. Hoy en día, cada empresa tiene un mecanismo que asegura la calidad de sus productos. De hecho, durante la pasada década se ha usado ampliamente, como táctica de mercado, la declaración explícita de mensajes que ponían de manifiesto la calidad ofrecida por las empresas.

La historia de la garantía de calidad en el desarrollo de software ha ido paralela a la historia de la calidad en la fabricación de hardware. Durante los primeros años de la informática (los años 50 y 60), la calidad era responsabilidad únicamente del programador. Durante los años 70 se introdujeron estándares de garantía de calidad para el software en los contratos militares de desarrollo de software y se han extendido rápidamente en los desarrollos de software del mundo comercial.

La garantía de calidad del software (SQA) es un "planificado y sistemático diseño de acciones" que se requieren para asegurar la calidad del software. El alcance de la responsabilidad de la garantía de calidad se puede caracterizar de la mejor forma parafraseando un popular anuncio de automóviles: "La calidad es el trabajo N.° 1". Lo que esto implica en el desarrollo de software es que la responsabilidad de la garantía de calidad del software corresponde a muchos constituyentes de una organización -ingenieros de software, gestores del proyecto, clientes, comerciales y personas que trabajan dentro del grupo de SQA.

El grupo de SQA sirve como representación del cliente dentro de la casa. Es decir, la gente que lleva a cabo la SQA debe mirar el software desde el punto de vista del cliente. ¿Satisface de forma adecuada el software los factores de calidad apuntados en la Sección 1.1? ¿Se ha realizado el desarrollo del software de acuerdo con estándares preestablecidos? ¿Han desempeñado apropiadamente sus papeles las disciplinas técnicas como parte de la actividad de SQA? El grupo de SQA intenta responder a éstas y otras cuestiones para asegurar que se mantiene la calidad del software.

1.3. ACTIVIDADES DE SQA

La garantía de calidad de software comprende una gran variedad de tareas, asociadas con siete actividades principales: (1) aplicación de métodos técnicos; (2) realización de revisiones técnicas formales; (3) prueba del software; (4) ajuste a los estándares; (5) control de cambios; (6) mediciones; (7) registro y realización de informes.

La calidad del software debe estar diseñada para el producto o sistema; no es algo impuesto a posteriori. Por esta razón, la SQA comienza realmente con un conjunto de "herramientas y métodos técnicos" que ayudan al analista a conseguir una especificación de alta calidad y un diseño de alta calidad.

Una vez que se ha creado una especificación (o prototipo) y un diseño, debe ser garantizada su calidad. La actividad central que permite garantizar la calidad es la "revisión técnica formal". La revisión técnica formal (RTF) es una especie de reunión del personal técnico con el único propósito de descubrir problemas de calidad. En muchas situaciones se ha visto que las revisiones son tan efectivas como la prueba para descubrir defectos en el software.

La "prueba del software" combina una estrategia de múltiples pasos con una serie de métodos de diseño de casos de prueba que ayudan a asegurar una efectiva detección de errores. Muchos grupos de desarrollo de software usan la prueba del software como una "red de seguridad" para la garantía de calidad. Esto es, asumen que mediante la prueba descubrirán la mayoría de los errores, mitigando así la necesidad de otras actividades de SQA. Desgraciadamente, la prueba, incluso cuando se realiza adecuadamente, no es tan efectiva como desearíamos para todas las clases de errores.

El grado de aplicación de "procedimientos y estándares" en el proceso de la ingeniería del software varía de empresa a empresa. En muchos casos, los estándares vienen dados por los clientes o por mandamientos de regulación. En otras situaciones, los estándares se imponen por sí solos. Si existen estándares formales (escritos), se debe establecer una actividad de SQA para garantizar que se siguen. La garantía de seguimiento de estándares puede ser

llevada a cabo por los encargados del desarrollo del software como parte de una revisión técnica formal o, en situaciones en que se requiera una verificación del seguimiento independiente, por el grupo de SQA mediante su propia auditoría.

Una de las principales amenazas para la calidad del software viene de una fuente aparentemente benigna: "los cambios". Cada cambio realizado sobre el software en potencia puede introducir errores o crear efectos laterales que propaguen errores. El proceso de "control de cambios" (una tarea que es parte de la gestión de configuraciones del software) contribuye directamente a la calidad del software, al formalizar las peticiones de cambio, evaluar la naturaleza del cambio y controlar el impacto del cambio. El control de cambios se aplica durante el desarrollo del software y, posteriormente, durante la fase de mantenimiento del software.

La "medición" es una actividad integral para cualquier disciplina. Un objetivo importante de la SQA es seguir la pista a la calidad del software y evaluar el impacto de los cambios de metodología y de procedimiento que intentan mejorar la calidad del software. Para conseguir esto, se deben recolectar "métricas del software". Las métricas del software engloban un amplio conjunto de medidas técnicas orientadas a la gestión.

El "registro de información" y la "generación de informes" para la garantía de calidad del software dan procedimientos para la recolección y divulgación de información de SQA. Los resultados de las revisiones, auditorías, control de cambios, prueba y otras actividades de SQA deben convertirse en una parte del registro histórico de un proyecto y deben ser divulgados a la plantilla de desarrollo para que tengan conocimiento de ellos. Por ejemplo, los resultados de cada RTF de un diseño procedimental se registran y se guardan en una "carpeta" que contenga toda la información técnica y de SQA sobre cada módulo.

2. 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.

2.1. 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 computadora.

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 computadora, 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:

ICED = Sumatorio Pi Di

donde i variará de 1 a 9, Pi es el peso relativo de la importancia de cada uno de los valores intermedios y Sumatorio de Pi = 1 (si todos los Di tienen el mismo peso, entonces Pi = 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.

2.2. 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.

3. ESTRATEGIAS DE PRUEBA

Tema 32

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 informa-

ción, 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 computadora, damos vueltas en espiral a través de una serie de flujos o líneas que disminuyen el nivel de abstracción en cada vuelta.

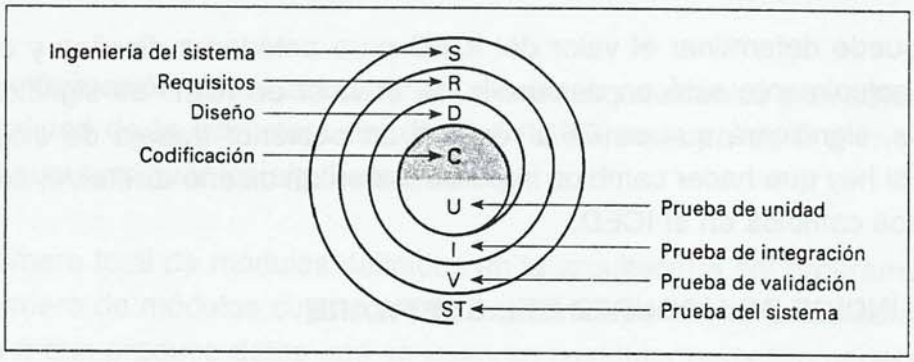

Figura 50.2. Estrategia de prueba

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 computadora nos movemos hacia afuera por una espiral que, a cada vuelta, aumenta el alcance de la prueba.

3.1. 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 examina 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 le 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 senten-

cias del módulo se ejecutan por lo menos una vez. Y, finalmente, se prueban todos los "caminos de manejo de errores".

3.2. 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.

3.2.1. 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".

3.2.2. 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.

3.3. 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.

3.3.1. Pruebas alfa y beta

Cuando software a medida para un cliente, se lleva a cabo 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.

3.4. 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 computadora. 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.

3.4.1. 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.

3.4.2. 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.

3.4.3. 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 tirar abajo el programa.

3.4.4. 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.

4. DOCUMENTACIÓN

Tema 52

La documentación la compone toda la información escrita que acompaña al programa cuya finalidad es facilitar el aprendizaje y uso de la aplicación.

La documentación, según su diseño, se agrupa en dos bloques:

- Manuales.
- Ayudas.

Los manuales son libros que describen de forma exhaustiva las características del programa así como su utilización. En la actualidad, los programas distribuidos en CD-ROM incluyen copia de los manuales en éste.

Las ayudas son textos incluidos en el propio programa que el usuario puede reclamar durante la ejecución del mismo. Vienen a ser un resumen de los manuales.

4.1. MANUALES

Cuando se diseñan los manuales, se ha de tener en cuenta el tipo de usuario que potencialmente hará uso de la aplicación (véase tipos de usuarios en tema 45). Atendiendo a los posibles tipos de usuarios se deben diseñar los siguientes manuales:

- Guía de usuario.
- Manual de referencia.
- Guía rápida.

Las guías de usuario debe diseñarse para usuarios "novicios" y debe tener la estructura de un Tutorial. Al usuario novicio hay que "llevarlo de la mano" para mostrarle la aplicación.

Una guía de usuario debe empezar por una "Descripción general" de la aplicación y sus objetivos. Nunca debe hacerse una descripción inicial de todos sus elementos y tareas. Por ejemplo, un manual de usuario de un Procesador de Texto cuando describa la tarea de escribir una carta básica no hablará de valores por omisión para márgenes, justificaciones, tipo de letra...

Al diseñar esta guía se debe reflexionar sobre el mundo extrainformático para trasladarlo a nuestra aplicación. Sobre un ejemplo inicial se plantean sucesivamente las tareas de nuestra aplicación. En esta fase son necesarias tanto una cuidada redacción como la pedagogía. El desarrollo de ejemplos debe ir acompañado de "volcados" de pantalla para facilitar al lector su ubicación dentro del programa.

Los Manuales de Referencia deben diseñarse pensando en usuarios "intermitentes", los cuales tienen un conocimiento de las prestaciones de la aplicación pero no suelen recordar los pasos a seguir para realizar una tarea. Por tanto, la organización de estos manuales no debe ser un diccionario, sino un Tesauro (organizado por significados -tareas-). El criterio de ordenación será por tareas y dentro de la tarea por profundidad en la misma. Deben incluir pequeños ejemplos para la aclaración de la sintaxis.

Las Guías Rápidas tienen como finalidad recordar al usuario experto la sintaxis exacta de un comando o una orden a un usuario "experto" en nuestra aplicación. Su organización debe ser por tareas y dentro de cada tarea por orden alfabético; se pretende con ello facilitar el acceso a una consulta puntual del usuario. La organización según tareas debe ser lo más

aproximada a la categorización que se haya hecho en el programa a través de los diferentes menús de opciones.

4.2. AYUDAS

Como se ha comentado, las ayudas son texto escritos incluidos en el programa accesibles durante la ejecución. Su finalidad es evitar al usuario tener que recurrir a los manuales para una aclaración sobre el funcionamiento del programa.

En este sentido, la documentación se divide en tres grupos:

- Ayuda inmediata.
- Ayuda en línea.
- Ayuda general.

La ayuda inmediata es la información que aparece sin que el usuario la reclame (retroalimentación). Esta compuesta de frases cortas, siendo su misión:

- Ayudar en la navegación del programa.
- Aclarar la entrada de datos de inmediata ejecución.
- Aclarar las opciones de menú.

Un ejemplo de esta ayuda es el texto que aparece al ubicar el puntero de ratón en un botón.

La ayuda en línea la componen textos que explican brevemente los pasos que ha de seguir el usuario para desarrollar la tarea que este realizando en un momento dado. Se deben incluir pequeños ejemplos. Esta ayuda, también denominada "sensible al contexto", aparece a petición del usuario, el cual pulsará una tecla predefinida cuando desee este tipo de ayuda.

La Ayuda General es un resumen de los manuales de usuario y de referencia. Debe estar estructurada de forma que el usuario pueda "navegar por ella" utilizando técnicas de "hipertexto". La entrada a esta ayuda se hará a través de una opción de menú destinada a este fin, pudiéndose realizar la entrada a través de un índice de contenidos o escribiendo un texto sobre el que el programa ha de buscar temas de ayuda asociados.

El índice de contenidos estará estructurado como el índice del manual de usuario (por tareas). El usuario deberá seleccionar el tema sobre el que desea información. La información presentada en pantalla no debe ser demasiado extensa a fin de facilitar su lectura.

Cuando sobre una palabra o frase del texto de la pantalla existe información adicional, debe ser definida como palabra "clave", de forma que si el usuario la selecciona con el dispositivo apuntador, aparezca una nueva página de texto relacionado con esta palabra.

La navegación debe incluir funciones para facilitar la ubicación del usuario:

- Anterior. para que la ayuda vuelva a la página de texto anterior.
- Siguiente. para que la ayuda pase a la página siguiente de la ayuda.
- Historial, para que el usuario vea la secuencia de páginas de ayuda que ha seguido y pueda cambiar a alguna de ellas. En este historial aparecen los encabezados de las páginas.
- Buscar, para que el usuario escriba un texto sobre el que requiere ayuda.

RESUMEN

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.

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.

Se han visto 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.

Las estrategias de pruebas sigue la espiral de desarrollo del sistema pero en sentido inverso. 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 computadora nos movemos hacia afuera por una espiral que, a cada vuelta, aumenta el alcance de la prueba.

La documentación de un programa la compone toda la información escrita que acompaña al programa cuya finalidad es facilitar el aprendizaje y uso de la aplicación. Según su diseño, se agrupa en dos bloques:

- Manuales.
- Ayudas.

Atendiendo a los posibles tipos de usuarios se deben diseñar los siguientes manuales:

- Guía de usuario.
- Manual de referencia.
- Guía rápida.

La documentación de ayuda se divide en tres grupos:

- Ayuda inmediata.
- Ayuda en línea.
- Ayuda general.