Guide to the Software Engineering: Body of Knowledge.

Capitulo 4

TESTEANDO SOFTWARE.

INTRODUCCION

Las pruebas de software consisten en una verificación *dinámica* de un programa de software que busca, a través de un conjunto *finito* de casos, que este cumpla con una serie de comportamientos esperados en un contexto determinado.

1. **Fundamentos de las pruebas de software.**
   * 1. **Definiciones de Prueba y terminología relacionada**
     2. **Fallas vs Fracasos**

Muchos términos son usados en la literatura sobre ingeniería de software para describir un mal funcionamiento: notablemente *falla*, *fracaso* y *error*, entre otras. Esto es crucial para definir claramente la diferencia entre la *causa* de un mal funcionamiento (para el cual se usará el término *falla* aquí) y los efectos no deseados observados en el sistema (a los cuales se le denominarán *fracasos*). En efecto, puede haber fallos que nunca serán tratados como fracasos. Así, aunque las pruebas puedan detectar fracasos, son los fallos los que pueden y deben ser removidos. El término *defecto* suele ser el más utilizado y se considera válido cuando la distinción entre *falla* y *fracaso* no es importante.

Sin embargo, hay que reconocer que la causa de un *fracaso* no siempre puede identificarse inequívocamente. No existen criterios teóricos para determinar definitivamente, en general, el fallo que ha causado un *fracaso* detectado. Podría decirse que reparando o removiendo un determinado *fallo*, algún *fracaso* observado puede solucionarse, aunque este pudiera llegar a arreglarse realizando otras modificaciones. Para evitar la ambigüedad, se podría hacer referencia a los *fallos de entrada*, que no son mas que el conjunto de entradas que componen un *fracaso*.

* 1. **Cuestiones clave**
     1. **Criterios de selección de pruebas / Criterios de adecuación de pruebas.**

Un criterio de selección de pruebas es un medio para decidir cuáles deben ser los casos de prueba adecuados. Un criterio de selección se puede usar para seleccionar casos de pruebas o para comprobar si el grupo de casos de prueba es apropiado-o sea, para decidir si se puede terminar de hacer pruebas.

**1.2.2 Efectividad de las pruebas/Objetivos para las pruebas**

Realizar pruebas consiste en observar un conjunto de ejecuciones del programa. Hay diferentes objetivos que nos pueden guiar en la selección del conjunto de pruebas: la efectividad del grupo de pruebas sólo se puede evaluar en función del objetivo seleccionado.

**1.2.3 Realizar pruebas para la identificación de defectos**

Cuando realizamos pruebas para la identificación de defectos, una prueba es satisfactoria si produce un error en el sistema. Es éste un enfoque completamente diferente al de realizar pruebas para demostrar que el software satisface las especificaciones u otro conjunto de propiedades deseadas, en cuyo caso una prueba satisfactoria es aquella en la que no se observan errores (al menos significativos).

**1.2.4 El problema del oráculo**

Un oráculo es cualquier agente (humano o mecánico) que decide si un programa se comporta correctamente durante una prueba y consecuentemente produce un veredicto de superada o fallada. Hay varios tipos diferentes de oráculos, y la automatización de oráculos puede ser muy difícil y cara.

**1.2.5 Limitaciones teóricas y prácticas de las pruebas**

La teoría de pruebas advierte en contra de un nivel injustificado de confianza en una serie de pruebas superadas. Desafortunadamente, la mayor parte de los resultados establecidos en la teoría de pruebas son negativos, en el sentido de que establecen aquello que la prueba no puede conseguir, en vez de lo que consiguió. La más famosa cita a este respecto es el aforismo de Dijkstra que dice “las pruebas de un programa se pueden usar para mostrar las presencia de errores, pero nunca para demostrar su ausencia”. La razón obvia es que realizar un grupo completo de pruebas no es posible en el software real. Como consecuencia, las pruebas deben dirigirse en función de los riesgos y por tanto pueden verse como una estrategia de gestión de riesgo.

**1.2.6 El problema de los caminos no alcanzables**

Los caminos no alcanzables son aquellos caminos de control que no pueden ejecutarse para ninguna entrada de datos. Son un problema importante en las pruebas orientadas por caminos y particularmente en las derivaciones automáticas de entradas de pruebas que se emplean en las técnicas de pruebas basadas en código.

**1.2.7 Posibilidad de hacer pruebas**

El término “posibilidad de hacer pruebas” tiene dos significados relacionados pero diferentes: por un lado, se refiere al grado de facilidad del software para satisfacer un determinado criterio de cobertura de pruebas; por otro, se define como la probabilidad, posiblemente cualificada estadísticamente, de que los errores del software queden expuestos durante las pruebas. Ambos significados son importantes.

**1.3 Relación de las pruebas con otras actividades**

Las pruebas del software, aunque diferentes, están relacionadas con las técnicas de gestión de la calidad del software estático, las pruebas de validez del software, la depuración y la programación. Sin embargo, es útil considerar las pruebas desde el punto de vista del analista de calidad del software o certificador.

* Pruebas vs. técnicas de Gestión de Calidad del Software Estático.
* Pruebas vs. Pruebas de Validez y Verificación Formal.
* Pruebas vs. Depuración.
* Pruebas vs. Programación.
* Pruebas y Certificación (Wak99).

1. **Niveles de Pruebas**

**2.1 El objeto de la prueba**

Las pruebas del software se realizan normalmente a diferentes niveles durante los procesos de desarrollo y mantenimiento. Esto significa que el objeto de las pruebas puede cambiar: un módulo, un grupo de dichos módulos (relacionados por propósito, uso, comportamiento, o estructura), o un sistema completo. Conceptualmente se pueden distinguir tres grandes niveles de pruebas, llamadas de Unidad, de Integración y del Sistema. No hay un modelo de proceso implícito, ni se asume que ninguno de estos tres niveles tiene mayor importancia que los otros dos.

**2.1.1 Pruebas de Unidad**

Las pruebas de unidad verifican el funcionamiento aislado de partes del software que se pueden probar independientemente. Dependiendo del contexto, estas partes podrían ser subprogramas individuales o un componente más grande formado por unidades muy relacionadas. Hay una definición más precisa de prueba de unidad en el estándar IEEE de pruebas de unidad del software (IEEE1008-87), que también describe un método integrado para realizar y documentar pruebas de unidad sistemáticamente. Normalmente, las pruebas de unidad se realizan con acceso al código fuente y con el soporte de herramientas de depuración, pudiendo implicar a los programadores que escribieron el código.

2.1.2 Pruebas de Integración

Una prueba de integración es el proceso de verificar la interacción entre componentes de software. Estrategias clásicas de integración, como arriba-abajo o abajo-arriba, se usan, tradicionalmente, con software estructurado jerárquicamente. Las estrategias modernas de integración están dirigidas por la arquitectura, lo que supone integrar los componentes de software o subsistemas basándose en caminos de funcionalidad identificada. Las pruebas de integración son una actividad continua, que sucede en cada fase en que los ingenieros de software tienen que hacer abstracciones de las perspectivas de bajo nivel y concentrarse en las perspectivas del nivel que están integrando. Con la excepción de software sencillo y pequeño, las estrategias de pruebas de integración sistemáticas e incrementales son preferibles a probar todos los componentes juntos al final, lo que se conoce (de forma gráfica), como pruebas en “*big bang”*.

**2.1.3 Pruebas del Sistema.**

Las pruebas del sistema se ocupan del comportamiento de un sistema completo. La mayoría de los fallos funcionales deberían haber sido identificados antes, durante las fases de pruebas de unidad y pruebas de integración. Las pruebas del sistema se consideran normalmente como las apropiadas para comparar el sistema con los requisitos no funcionales del sistema, como seguridad, velocidad, exactitud y confiabilidad. Las interconexiones externas con otras aplicaciones, utilidades, dispositivos hardware o con el sistema operativo, también se evalúan en este nivel.

**2.2 Objetivos de las pruebas**

Las pruebas se realizan en relación a conseguir un determinado objetivo, que se ha definido más o menos explícitamente y con diversos niveles de precisión. Definir el objetivo, en términos precisos y cuantitativos, permite establecer controles en el proceso de las pruebas. Las pruebas se pueden realizar para verificar propiedades distintas. Se pueden asignar casos de prueba para comprobar que las especificaciones funcionales se han implementado correctamente, a lo que la literatura se refiere como pruebas de conformidad, pruebas de corrección o pruebas de funcionalidad. Sin embargo, también se pueden hacer pruebas a otras muchas propiedades no funcionales, como rendimiento, confiabilidad y facilidad de uso, entre otras muchas.

Otros objetivos importantes de las pruebas incluyen (aunque no se limitan a) mediciones de confiabilidad, evaluación de la facilidad de uso y aceptación, para los cuales se utilizarían métodos diferentes. Se debe tener en cuenta que los objetivos de las pruebas varían con el objeto de las pruebas; en general, propósitos diferentes son tratados con diferentes niveles de pruebas. Las referencias recomendadas para este punto describen el conjunto de objetivos de pruebas potenciales. Los puntos enumerados seguidamente son los que se citan más frecuentemente en la literatura. Téngase en cuenta que algunos tipos de pruebas son más apropiados para paquetes de software hechos a medida, pruebas de instalación, por ejemplo; mientras otros son más apropiados para productos más genéricos, como pruebas beta.

**2.2.1 Pruebas de aceptación/calificación.**

Las pruebas de aceptación comparan el comportamiento del sistema con los requisitos del cliente, sea cual sea la forma en que éstos se hayan expresado. El cliente realiza, o específica, tareas típicas para comprobar que se satisfacen sus requisitos o que la organización los ha identificado para el mercado al que se destina el software. Esta actividad de pruebas puede incluir o no a los desarrolladores del sistema.

**2.2.2 Pruebas de instalación.**

Normalmente, cuando las pruebas de aceptación han terminado, el software se puede comprobar una vez instalado en el entorno final. Las pruebas de instalación se pueden ver como pruebas del sistema realizadas en relación con los requisitos de la configuración de hardware. Los procedimientos para la instalación también se podrían verificar.

**2.2.3 Pruebas alfa y beta.**

A veces, antes de poner el software en distribución, éste se proporciona a un grupo representativo de usuarios potenciales para que puedan usarlo en pruebas en las instalaciones del desarrollador (pruebas alpha) o externamente (pruebas beta). Dichos usuarios notifican problemas con el producto. Normalmente, el uso de versiones alfa y beta sucede en entornos no controlados y no siempre se le hace referencia en los planes de pruebas.

**2.2.4 Pruebas de conformidad/pruebas funcionales/pruebas de corrección.**

Las pruebas de conformidad tienen el objetivo de verificar si el comportamiento del software se corresponde con las especificaciones.

**2.2.5 Materialización de la confiabilidad y evaluación.**

Las pruebas, al ayudar a identificar errores, son un medio para mejorar la confiabilidad. Por contraste, generando casos de prueba aleatorios siguiendo el perfil de operaciones, se pueden derivar aproximaciones estadísticas de confiabilidad. Cuando se usan modelos que potencian la confiabilidad, ambos objetivos se pueden alcanzar al mismo tiempo.

**2.2.6 Pruebas de regresión.**

Según (IEEE610.12-90), las pruebas de regresión son “*pruebas selectivas que se repiten en un componente para verificar que los cambios no han producido efectos indeseados...”* En la práctica, la idea es demostrar que cierto software que previamente pasó un conjunto de pruebas, aún las pasa. Beizer las define como cualquier repetición de pruebas que tiene como objetivo demostrar que el comportamiento del software no ha cambiado, excepto en aquellos aspectos en que se haya requerido así. Por supuesto se tiene que llegar a un compromiso entre realizar pruebas de regresión cada vez que se hace un cambio y los medios de que se dispone para realizar las pruebas. Las pruebas de regresión se pueden realizar en cada uno de los niveles de pruebas descritos en el punto 2.1. El objeto de la prueba y son válidas tanto para pruebas funcionales como no funcionales.

**2.2.7 Pruebas de rendimiento.**

Estas pruebas tienen el objetivo de verificar que el software alcanza los requerimientos de rendimiento especificados, particularmente los de capacidad y tiempo de respuesta. Un tipo particular de pruebas de rendimiento son las pruebas de volumen (Per95:p185, p487; Pfl01:p401), en los que las limitaciones internas del programa o sistema se ponen a prueba.

**2.2.8 Pruebas de desgaste.**

Las pruebas de desgaste hacen funcionar el software a la máxima capacidad para la que fue diseñado, y por encima de ella.

**2.2.9 Pruebas de continuidad.**

Un grupo de pruebas se ejecuta en dos versiones diferentes de un producto software y los resultados se comparan.

**2.2.10 Pruebas de recuperación.**

El objetivo de las pruebas de recuperación es verificar la capacidad del software para reiniciarse después de un “*desastre*”.

**2.2.11 Pruebas de configuración.**

En los casos en los que el software se construye para dar servicio a distintos usuarios, las pruebas de configuración analizan el software en las diferentes configuraciones especificadas.

**2.2.12 Pruebas de facilidad de uso.**

Este proceso evalúa lo fácil que le resulta usar y aprender a usar el software al usuario, incluyendo la documentación del usuario, la efectividad de las funciones del software para soportar las tareas de usuario y, finalmente, la habilidad de recuperarse de errores provocados por el usuario.

**2.2.13 Desarrollo dirigido por pruebas**

El desarrollo dirigido por pruebas no es una técnica en sí misma, pero promociona el uso de pruebas como una parte subordinada al documento de especificación de requisitos en vez de una comprobación independiente de que el software implementa dichos requerimientos correctamente.

**3. Técnicas de pruebas**

Uno de los objetivos de las pruebas es revelar el máximo número posible de fallos potenciales y muchas técnicas se han desarrollado con este objetivo, intentando “*romper*” el programa ejecutando una o más pruebas seleccionadas de un cierto grupo de ejecuciones considerado equivalente. El principio subyacente de estas técnicas es tratar de ser lo más sistemático posible identificando un conjunto representativo de comportamientos del programa; por ejemplo, identificando subclases del dominio de entrada de datos, de los escenarios, de los estados y del flujo de datos. Es difícil encontrar una base homogénea para clasificar todas las técnicas, por lo que la aquí utilizada debe entenderse como un compromiso. La clasificación se basa en cómo los ingenieros del software generan las pruebas basándose en su intuición y experiencia, en las especificaciones, la estructura del código, los errores a descubrir (reales o artificiales), el uso de campos de entrada de datos o, en último término, la naturaleza de la aplicación. Algunas veces, estas técnicas se clasifican como “*de caja blanca*” (también conocidas como caja de cristal), si las pruebas están basadas en información acerca de cómo se ha diseñado o programado el software, o como “*de caja negra”* si los casos de prueba se basan solamente en el comportamiento de la entrada y salida de datos. Una última categoría se basa en el uso combinado de dos o más técnicas. Obviamente, no todo el mundo usa estas técnicas con la misma frecuencia. La siguiente lista incluye las técnicas que los ingenieros de software deberían conocer.

**3.1 Pruebas basadas en la intuición y experiencia del ingeniero de software.**

**3.1.1 Pruebas ad hoc.**

Quizás la técnica usada más globalmente continúan siendo las pruebas ad hoc: las pruebas se generan a partir la habilidad, intuición y experiencia en programas similares del ingeniero de software. Las pruebas ad hoc pueden ser útiles para identificar casos de prueba especiales, aquellos que no se pueden extraer fácilmente mediante técnicas formales.

**3.1.2 Pruebas por exploración.**

Las pruebas por exploración se definen como aprendizaje, diseño de pruebas y ejecución de pruebas al mismo tiempo. Esto significa que las pruebas no se definen primero como parte de un plan de pruebas establecido, si no que se diseñan, ejecutan y se modifican dinámicamente. La efectividad de las pruebas por exploración se basa en el conocimiento del ingeniero de software, que se puede derivar de varias fuentes: el comportamiento observado del producto durante las pruebas, su familiaridad con la aplicación, la plataforma o el proceso de fallos, los posibles tipos de errores y fallos, el riesgo asociado con un producto en particular, etc.

**3.2 Técnicas basadas en la especificación.**

**3.2.1 Particiones de equivalencia.**

El dominio de la entrada de datos se subdivide en colecciones de subconjuntos, o clases de equivalencia, las cuales se consideran equivalentes de acuerdo con la relación especificada. Un grupo representativo de pruebas (a veces solo uno) se toma de cada clase.

**3.2.2 Análisis de los valores límite.**

Casos de prueba se seleccionan en y cerca de los límites del dominio de las variables de la entrada de datos, basándose en la idea de que una gran parte de los errores se concentran cerca de los valores extremos de la entrada de datos. Una extensión de esta técnica son las pruebas de robustez, donde se seleccionan casos de prueba que se encuentran fuera del dominio de las variables de la entrada de datos, para comprobar la robustez del programa con entradas de datos erróneas e inesperadas.

**3.2.3 Tablas de decisión.**

Las tablas de decisión representan relaciones lógicas entre condiciones (mayoritariamente entradas) y acciones (mayoritariamente salidas). Los casos de prueba se derivan sistemáticamente considerando cada combinación de condiciones y acciones posibles. Una técnica relacionada es el gráfico “*causa-efecto*”.

**3.2.4 Basadas en máquinas de estado finito.**

Al modelar un programa como una máquina de estado finito, se pueden seleccionar las pruebas de manera que cubran estados y sus transiciones.

**3.2.5 Pruebas basadas en las especificaciones formales.**

Si las especificaciones se proporcionan en un lenguaje formal, es posible realizar una derivación automática de los casos de prueba funcionales y, al mismo tiempo, proporcionar unos resultados de referencia, un oráculo, que se usa para comprobar los resultados de las pruebas.

**3.2.6 Pruebas aleatorias.**

En este caso las pruebas se generan de una manera completamente aleatoria. Esta forma de realizar pruebas se incluye en la categoría de entradas basadas en la especificación, ya que el domino de las entradas de datos se debe conocer para ser capaces de seleccionar elementos aleatorios del mismo.

**3.3 Técnicas basadas en el código**

**3.3.1 Criterio basado en el flujo de control.**

Los criterios de cobertura están basados en el flujo de control se usan para cubrir todos los bloques de código o líneas de código individuales o una combinación especifica de los mismos. Hay varios criterios de cobertura propuestos, como cobertura de condición/decisión. El criterio basado en el flujo de control más efectivo son las pruebas de caminos, cuyo objetivo es verificar todos los caminos de control de tipo entrada-salida del gráfico de flujos. Como, en general, las pruebas de caminos no son posibles debido a los bucles, en la práctica se usan otros criterios menos exigentes, como pruebas de líneas de código, pruebas de condiciones y pruebas de decisión. La idoneidad de dichas pruebas se mide en porcentajes; por ejemplo, cuando las pruebas han ejecutado todas las condiciones al menos una vez, se dice que se ha conseguido una cobertura de condiciones del 100%.

**3.3.2 Criterio basado en el flujo de dato.**

En las pruebas basadas en el flujo de datos, el gráfico de flujos de control tiene anotaciones con información acerca de cómo las variables del programa se definen, usan y destruyen. El criterio más efectivo, todos los caminos de uso-definición, requiere que para cada variable, se ejecute cada uno de los segmentos del camino del flujo de control de esa variable a un usa de esa definición. Para reducir el número de caminos necesarios, se emplean estrategias menos efectivas como todas las definiciones y todos los usos.

**3.3.3 Modelos de referencia para pruebas basadas en el código (gráfico de flujos, gráfico de llamadas).**

Aunque no es una técnica en sí misma, la estructura de control de un programa se representa usando gráficos de flujo en las técnicas de pruebas basadas en código. Un gráfico de flujo es un gráfico dirigido cuyos nodos y arcos se corresponden con elementos del programa. Por ejemplo, los nodos podrían representar líneas de código o secuencias de líneas de código ininterrumpidas y los arcos la transferencia de control entre nodos.

**3.4 Técnicas basadas en errores**

Con diferentes niveles de formalización, las técnicas basadas en errores idean casos de prueba que están especialmente orientados a descubrir categorías de errores probables o predefinidos.

**3.4.1 Conjeturar errores.**

En la conjetura de errores, los casos de pruebas se han diseñado específicamente por ingenieros de software intentando imaginar los errores más probables en un programa determinado. La historia de errores descubiertos en proyectos anteriores es una buena fuente de información, como lo es también la experiencia del ingeniero.

**3.4.2 Pruebas por mutación.**

Un mutante es una versión ligeramente modificada de un programa al que se le está haciendo pruebas, diferenciándose tan solo en un pequeño cambio sintáctico. Cada caso de prueba se aplica al original y a los mutantes generados: si una prueba consigue identificar la diferencia entre el programa y el mutante, se dice que se ha “*matado*” al mutante. Esta técnica se concibió originalmente para evaluar un conjunto de pruebas (véase 4.2), las pruebas por mutación son un criterio de pruebas en sí mismas: o se generan pruebas aleatorias hasta que se han matado los mutantes suficientes, o se diseñan pruebas específicas para matar a los mutantes supervivientes. En el último caso, las pruebas por mutación se pueden clasificar como técnicas basadas en código. El efecto de acoplamiento, que es la base asumida en las pruebas de mutación, consiste en asumir que buscando errores sintácticos simples, se encontrarán otros más complejos pero existentes. Para que esta técnica sea efectiva, se debe poder derivar un número importante de mutantes de una manera sistemática.

**3.5 Técnicas basadas en el uso**

**3.5.1 Perfil operativo**

Durante pruebas para la evaluación de la confiabilidad, el entorno de pruebas debe reproducir el entorno operativo del software tan fielmente como sea posible. La idea es deducir la futura confiabilidad del software durante su use real desde los resultados de las pruebas.

**3.5.2 Pruebas Orientadas a la Confiabilidad del Software.**

Las pruebas orientadas a la confiabilidad del software (SRET) son un método de pruebas que forma parte del proceso de desarrollo completo, donde la realización de pruebas está “*diseñada y guiada por los objetivos de confiabilidad y el uso relativo esperado y lo críticas que sean las distintas funciones en ese ámbito*”

**3.6 Técnicas basadas en la naturaleza de la aplicación**

Las técnicas anteriores se pueden aplicar a cualquier tipo de software. Sin embargo, para algunos tipos de aplicaciones, es necesario conocimientos específicos adicionales para derivar las pruebas. La siguiente lista proporciona unas cuantas áreas de pruebas especializadas, basándose en la naturaleza de la aplicación que se está comprobando:

* Pruebas Orientadas a Objetos.
* Pruebas basadas en componentes.
* Pruebas para Internet.
* Pruebas para GUI.
* Pruebas para programas concurrentes.
* Pruebas de conformidad de protocolos.
* Pruebas para sistemas de tiempo real.
* Pruebas para sistemas de seguridad crítica (IEEE1228-94).

**3.7 Seleccionando y combinando técnicas**

**3.7.1 Funcional y estructuralmente.**

Las técnicas de pruebas basadas en las especificaciones y el código se contrastan frecuentemente como pruebas funcionales vs estructurales. Estos dos métodos de selección de pruebas no se deber ver como alternativos si no como complementarios; de hecho, usan fuentes de información diferentes y se ha comprobado que remarcan diferentes tipos de problemas. Estas técnicas se pueden combinar, dependiendo del presupuesto para pruebas.

**3.7.2 Deterministas vs aleatorias.**

Los casos de pruebas se pueden seleccionar de una forma determinista, de acuerdo con una de las varias técnicas enunciadas, o seleccionadas aleatoriamente de una distribución de entradas de datos, como se hace normalmente en las pruebas de confiabilidad. Existen varias comparaciones analíticas y empíricas que analizan las condiciones en que uno de los métodos es más efectivo que el otro.

Para conseguir esto, se le asigna una probabilidad de distribución, o perfil, a las entradas de datos, basándose en la frecuencia en que suceden durante el funcionamiento real.

**4. Medidas de las pruebas.**

Algunas veces, las técnicas de pruebas se confunden con los objetivos de las pruebas. Las técnicas de pruebas se deben ver como medios que ayudan a conseguir los objetivos de las pruebas. Por ejemplo, la cobertura de condiciones es una técnica de pruebas muy popular. Conseguir el valor de la cobertura de condiciones no debería ser un objetivo de las pruebas en sí mismo: es solo un medio para mejorar las posibilidades de encontrar fallos realizando pruebas sistemáticas en cada condición del programa para un punto de decisiones. Para prevenir dichas interpretaciones erróneas, debería hacerse una distinción muy clara entre las medidas de las pruebas, que proporcionan una evaluación del programa que se está comprobando, basada en los resultados observados de las pruebas y aquellas que evalúan la completitud del conjunto de pruebas.

Las medidas se consideran, normalmente, como esenciales en los análisis de calidad. Las medidas también se pueden utilizar para optimizar la planificación y ejecuciones de las pruebas. La gestión de pruebas puede utilizar varios procesos para medir o vigilar el progreso realizado.

**4.1 Evaluación de un programa durante las pruebas (IEEE982.1-98)**

**4.1.1 Medidas para ayudar en la planificación y diseño de pruebas de programas.**

Las medidas basadas en el tamaño de un programa (por ejemplo, número de líneas de código o métodos) o en la estructura de un programa (como la complejidad), se usan para guiar a las pruebas. Las medidas estructurales pueden incluir medidas entre módulos del programa, en términos de la frecuencia en que cada módulo llama a los otros.

**4.1.2 Tipos de errores, clasificación y estadísticas.**

La literatura de pruebas es rica a la hora de clasificar y analizar errores. Con el objetivo de hacer las pruebas más efectivas, es importante saber que tipos de errores se pueden encontrar en un programa que se está comprobando y la frecuencia relativa en que estos errores han sucedido antes. Esta información puede ser muy útil para realizar predicciones de calidad y también para mejorar el proceso.

**4.1.3 Densidad de fallos.**

Un programa que se está comprobando se puede valorar contando y clasificando los errores descubiertos por su tipo. Parra cada tipo de error, la densidad de errores se mide como la razón entre el número de errores encontrados y el tamaño del programa.

**4.1.4 Vida de las pruebas, evaluación de confiabilidad.**

Una estimación estadística de la confiabilidad del software, que se puede conseguir mediante la realización y evaluación de la confiabilidad (véase punto 2.2.5), se puede usar para evaluar un producto y decidir si las pruebas se pueden detener o no.

**4.1.5 Modelos de crecimiento de la confiabilidad.**

Los modelos de crecimiento de la confiabilidad proporcionan una predicción de confiabilidad basada en los fallos observados durante la realización y evaluación de la confiabilidad (véase punto 2.2.5). Estos modelos asumen, en general, que los errores que causan los fallos observados se han arreglado (aunque algunos modelos también aceptan arreglos imperfectos), y por tanto, el producto muestra una confiabilidad incremental de promedio. Existen docenas de modelos publicados en la actualidad. Muchos se basan en algunas presunciones comunes, y otros no. Mayoritariamente, estos modelos se dividen en modelos de cuenta de fallos y tiempo entre fallos.

**4.2 Evaluación de las pruebas realizadas**

**4.2.1 Medidas de la cobertura/completitud.**

Varios criterios de idoneidad de las pruebas necesitan que los casos de pruebas ejecuten sistemáticamente un conjunto de elementos identificados en el programa o en la especificación (véase punto 3). Para evaluar la completitud de las pruebas realizadas, los ingenieros de pruebas pueden monitorizar los elementos cubiertos y su número total. Por ejemplo, es posible medir el porcentaje de condiciones cubiertas ejecutadas entre las definidas en la especificación. La idoneidad de los criterios basados en código necesita la instrumentación adecuada del programa que se está comprobando.

**4.2.2 Introducción de errores.**

Algunas veces se introducen errores artificialmente en un programa antes de comprobarlo. Cuando las pruebas se realizan, algunos de estos errores aparecerán y posiblemente algunos otros que ya estaban en el software también aparecerán. En teoría, dependiendo de cuál de los errores artificiales aparezca y cuántos de ellos, se puede evaluar la efectividad de las pruebas y se puede estimar el número restante de errores genuinos. En la práctica, los matemáticos estadísticos se cuestionan la distribución y representatividad de los errores introducidos en relación con los errores genuinos y el tamaño pequeño de la muestra en la que se basa cualquier extrapolación. Algunos incluso afirman que esta técnica debería usarse con sumo cuidado, ya que introducir errores en el software acarrea el riesgo obvio de olvidarlos allí.

**4.2.3 Puntuación de la mutación.**

En las pruebas por mutación (véase el punto 3.4.2), la razón de mutantes matados por número total de mutantes generados puede ser una medida de la efectividad del conjunto de pruebas realizadas.

**4.2.4 Comparación y efectividad relativa de las diferentes técnicas.**

Se han llevado a cabo varios estudios para comparar la efectividad relativa de las diferentes técnicas de pruebas. Es importante ser preciso acerca de la propiedad contra la cual las técnicas se han calificado; ¿cual, por ejemplo, es el significado exacto dado al término “*efectividad*”?

Las interpretaciones posibles son: el número de pruebas necesarias para encontrar el primer fallo, la razón entre el número de errores encontrados durante las pruebas y todos los errores encontrados durante y después de las pruebas, o cual fue la mejora de la confiabilidad. Se han llevado a cabo comparaciones analíticas y empíricas entre las diferentes técnicas, de acuerdo con cada uno de los significados de efectividad especificados antes.

**5. El Proceso de las Pruebas**

Los conceptos de pruebas, estrategias, técnicas y medidas han de ser integrados en un proceso definido y controlado, que debe ser gestionado por personas. El proceso de las pruebas soporta actividades y sirve de guía a los equipos de pruebas, desde la planificación de las pruebas hasta la evaluación de los resultados de las pruebas, de tal manera que se puede proporcionar una garantía justificada de que los objetivos de las pruebas se conseguirán de una manera económica.

**5.1 Consideraciones prácticas**

**5.1.1 Actitudes y programación egoless.**

Un elemento muy importante para el éxito de las pruebas es una actitud de colaboración respecto a las actividades de pruebas y garantía de calidad. Los jefes de proyecto tienen un papel fundamental en fomentar una recepción favorable en general respecto al descubrimiento de fallos durante el desarrollo y mantenimiento; particularmente, previniendo que los programadores se obsesionen con quien es el dueño del código, de tal forma que ninguno se sienta responsable por los fallos que aparezcan en su código.

**5.1.2 Guías para las pruebas.**

Se pueden guiar las fases de pruebas con varios mecanismos, por ejemplo; en pruebas basadas en el riego, que usa los riesgos en el producto para asignar prioridad y centrar la atención de las estrategias de pruebas; o en las pruebas basadas en situaciones, en las que los casos de pruebas se definen y basan en escenarios de software especificados.

**5.1.3 Gestión del proceso de las pruebas.**

Las actividades de pruebas realizadas a diferentes niveles. Se deben organizar, junto con las personas, herramientas, normas y medidas, en un proceso bien definido que será una parte integral del ciclo de vida del software. En el estándar IEEE/EIA 12207.0, las pruebas no se describen como un proceso independiente, si no que los principios de las actividades de las pruebas se encuentran incluidos con los cinco procesos primarios del ciclo de vida y con los procesos de soporte. En el estándar IEEE 1074, las pruebas se agrupan con otras actividades de evaluación como una parte integral del ciclo de vida completo.

**5.1.4 Documentación y productos de las pruebas.**

La documentación es una parte integral de la formalización del proceso de las pruebas. El estándar del IEEE Estándar para la Documentación de las Pruebas del Software (IEEE829-98) proporciona una buena descripción de los documentos de las pruebas y su relación entre cada uno y con el proceso de las pruebas. La documentación de pruebas puede incluir, entre otros, el Plan de Pruebas, la Especificación del Diseño de las Pruebas, la Especificación del Procedimiento de las Pruebas, la Especificación de los Casos de Pruebas, el Diario de las Pruebas y el Informe de Problemas o de Incidentes durante las Pruebas. El software que se está comprobando se documenta como el Artículo en Pruebas. La documentación de las pruebas se debe generar y actualizar continuamente, con el mismo nivel de calidad que cualquier otro tipo de documentación en la ingeniería del software.

**5.1.5 Equipo de pruebas interno vs equipo independiente**

La formalización del proceso de pruebas también puede formalizar la organización del equipo de pruebas. El equipo de pruebas puede estar compuesto por miembros internos (parte del equipo del proyecto, involucrados o no en la construcción del software), o de miembros externos, con la esperanza de contar con una perspectiva independiente y sin prejuicios, o, finalmente, de miembros internos y externos. La decisión puede ser afectada por consideraciones como coste, planificación, nivel de madurez de las organización involucradas y como de crítica sea la aplicación.

**5.1.6 Estimación coste/esfuerzo y otras medidas del proceso.**

Los jefes de proyectos pueden usar varias medidas, acerca de los recursos invertidos en las pruebas y de la efectividad de las varias fases de pruebas en encontrar fallos, para controlar y mejorar el proceso de las pruebas. Estas medidas de las pruebas pueden cubrir, entre otros, aspectos como el número de casos de pruebas especificados, el número de casos de pruebas ejecutados, el número de casos de pruebas superados y el número de casos de pruebas no superados. La evaluación de los informes de las fases de pruebas se puede combinar con análisis de las raíces de las causas para evaluar la efectividad del proceso de las pruebas en encontrar errores tan pronto como sea posible. Dicha evaluación se puede asociar con el análisis de riesgos. Lo que es más, los recursos que merece la pena invertir en las pruebas deberían ser proporcionales al uso/importancia de la aplicación: diferentes técnicas tienen distinto coste y proporcionan diferentes niveles de seguridad en la confiabilidad del producto.

**5.1.7 Finalización.**

Se debe tomar una decisión acerca de cuantas pruebas son suficientes y cuando la fase de pruebas se puede finalizar. Las medidas concienzudas, como las conseguidas mediante cobertura de código o completitud funcional y la estimación de densidad de errores o de confiabilidad operativa, proporcionan un suporte muy útil, pero no son suficientes por sí mismas. Esta decisión también incluye consideraciones acerca del coste y los riesgos en que se incurrirá debido a los fallos potenciales que aún queden, en vez del coste que conllevaría continuar realizando pruebas.

**5.1.8 Reutilización de pruebas y patrones de pruebas.**

Con el objetivo de realizar pruebas o mantenimiento de una forma organizada y efectiva respecto al coste, los medios usados para realizar pruebas en cada parte del software se deberían reutilizar de una forma sistemática. Dicho repositorio de material de pruebas debe estar bajo el control de un software de gestión de configuraciones, de forma que los cambios en los requerimientos del software o el diseño queden reflejados en cambios en el alcance de las pruebas realizadas. Las soluciones adoptadas para realizar pruebas en determinados tipos de aplicaciones bajo determinadas circunstancias, teniendo en cuenta los motivos detrás de las decisiones que se han tomado, forman un patrón de pruebas que se puede documentar y ser reutilizado en proyectos similares.

**5.2 Actividades de las pruebas**

En este punto, se verá una pequeña introducción a las actividades del software; gestionar con éxito las actividades relacionada con las pruebas, como la siguiente descripción da a entender, depende en gran medida del proceso de Gestión de Configuración del Software.

**5.2.1 Planificación.**

Como cualquier otro aspecto de la gestión de proyectos, las actividades de las pruebas se deben planificar. Algunos aspectos clave de la planificación de las pruebas incluyen la coordinación de personal, la gestión de instalaciones y equipos disponibles (que pueden incluir soportes magnéticos, planes de pruebas y procedimientos) y planificar en caso de posibles situaciones no deseables. Si se mantiene más de una línea base del software al mismo tiempo, una importante consideración de planificación es el tiempo y esfuerzo necesario para asegurarse de que se ha usado la configuración correcta para establecer el entorno de pruebas.

**5.2.2 Generación de casos de pruebas.**

La generación de caos de pruebas se basa en el nivel de pruebas que se vaya a realizar y en las técnicas de pruebas a usar. Los casos de pruebas deberían estar bajo el control de un software de gestión de configuraciones e incluir los resultados esperados para cada prueba.

**5.2.3 Desarrollo en el entorno de pruebas.**

El entorno usado para las pruebas debería ser compatible con las herramientas de ingeniería de software. Debería facilitar el desarrollo y control de casos de pruebas y la anotación y recuperación de los resultados esperados, los scripts y otros materiales de pruebas.

**5.2.4 Ejecución.**

La ejecución de las pruebas deberían incluir un principio básico de experimentación científica: todos los pasos durante las pruebas se deberían realizar y documentar de una forma lo suficientemente clara, que cualquier otra persona debería ser capaz de reproducir los resultados. Por tanto, las pruebas deben realizarse de acuerdo con los procedimientos documentados y usando una versión claramente definida del software que se está comprobando.

**5.2.5 Evaluación de los resultados de las pruebas.**

Los resultados de las pruebas se deben evaluar para determinar si las pruebas han sido satisfactorias o no. En la mayoría de los casos, “*satisfactorias*” significa que el software se ha ejecutado como se esperaba y no ha tenido ningún resultado inesperado importante. No todos los resultados inesperados son necesariamente errores, ya que se podría considerar que algunos son simple ruido. Antes de que se pueda arreglar un error, se necesita realizar un análisis y algún trabajo de depuración para identificarlo, aislarlo y describirlo. Cuando los resultados de las pruebas son particularmente importantes, puede que se convoque una revisión formal para evaluarlas.

**5.2.6 Notificación de problemas/Diario de pruebas.**

Las actividades de las pruebas se pueden añadir a un diario de pruebas para identificar cuando una prueba se ha ejecutado, quien la ha realizado, que configuración del software se ha utilizado y cualquier otra información relevante de identificación. Resultados inesperados o incorrectos se pueden añadir a un sistema de notificación de problemas, cuyos datos serán la base para procesos de depuración posteriormente y para arreglar los errores que causaron problemas durante las pruebas. Las anomalías no clasificadas como errores también se podrían documentar, en caso de que más tarde resulte que producen problemas más serios de lo que se pensó originalmente. Los informes de pruebas también son una entrada para los procesos de requerimientos de cambió de gestión.

5.2.7 Seguimiento de defectos.

Los fallos observados durante las pruebas son, en la mayoría de los casos, debidos a errores o defectos en el software. Dichos defectos se pueden analizar para determinar cuando fueron introducidos en el software, que clase de error produjo que se aparecieran (por ejemplo requerimientos definidos pobremente, declaraciones incorrectas de variables, fallo de memoria o errores de programación) y cuando deberían haber sido observados en el software por primera vez. La información del seguimiento de defectos se usa para determinar qué aspectos de la ingeniería del software necesitan mejorase y la efectividad de análisis y pruebas anteriores.

Capitulo 10

Calidad del Software

**INTRODUCCIÓN**

¿Qué es la calidad de software, y por qué es tan importante como para estar omnipresente en la Guía SWEBOK? Durante años, los autores y organizaciones han definido el término "calidad" de manera diferente. Para A Phil Crosby (Cro79), fue " la conformidad a las exigencias de usuario. " Watts Humphrey (Hum89) se refiere a calidad como " el alcanzar los niveles excelentes de salud para el empleo" mientras IBM acuñó la frase " la calidad conducida por el mercado, " frase basada en el objetivo de alcanzar la satisfacción de cliente total. Los criterios Bladridge para la calidad organizacional utilizan una frase similar “*calidad conducida por el cliente*”, e incluye la satisfacción del cliente como una consideración mayor. Más recientemente, la calidad se ha definido en (ISO9001-00) como “*el grado en que un conjunto de características inherentes cumple requisitos.*”

Este capítulo estudia los aspectos relativos a la calidad de software los cuales transcienden a los procesos del ciclo de vida. La calidad de software es un aspecto ubicuo en la ingeniería de software, y por lo tanto también es tratado en mucho de los KAS. En el sumario, la Guía SWEBOK describe un conjunto de modos de alcanzar la calidad del software. En particular, este KA tratará las técnicas estáticas, es decir, aquellas que no requieren la ejecución del software para su evaluación, mientras que las técnicas dinámicas son cubiertas en el KA referido a Pruebas del Software.

**DESGLOSE DE LOS TEMAS EN CALIDAD DEL SOFTWARE**

**1. Fundamentos de Calidad de Software**

Un acuerdo sobre exigencias de calidad, así como trasladar a la ingeniería del software qué constituye calidad, requiere que muchos de los aspectos del concepto calidad sean formalmente definidos y tratados.

Un ingeniero de software debería entender los significados subyacentes en los conceptos y características de calidad y su relevancia en el desarrollo o mantenimiento de software. El concepto relevante es que los requerimientos del software definen las características de calidad requeridas de ese software e influyen en los métodos de medición y criterios de aceptación para evaluar estas características.

**1.1. Ingeniería del Software Cultura y Ética.**

Los ingenieros de software esperan compartir un compromiso sobre calidad de software como una parte de su cultura. Una cultura sana sobre ingeniería del software se describe en [Wie96]. La ética puede jugar un papel significativo en la calidad de software, la cultura, y las actitudes de ingenieros de software. La IEEE Computer society y el ACM [IEEE99] han desarrollado un código de ética y práctica profesional basada en ocho principios con el objetivo de ayudar a los ingenieros de software a reforzar actitudes relacionadas con la calidad y con la independencia de su trabajo.

**1.2. Valor y coste de la calidad.**

El concepto de calidad no es tan simple como parece, para un ingeniero de productos hay muchas calidades deseadas relevantes para una perspectiva determinada de un producto, para que esto pueda ser tratado y determinado en el tiempo las exigencias de producto son puestas por escrito. Las características de calidad pueden requerirse o no, o se pueden requerir en un mayor o menor grado, y pueden hacerse compensaciones entre ellas. [Pfl01] El coste de calidad puede segmentarse en el coste de prevención, el coste de apreciación, el coste de fracaso interno, y el coste de fracaso externo. [Hou99] La motivación latente tras un proyecto de software es el deseo de crear un software que tiene valor, y este valor puede o no puede ser cuantificado como un coste. El cliente tendrá en mente algún coste máximo, a cambio del cual espera que se cumpla el objetivo básico del software. El cliente también puede tener alguna expectativa en cuanto a la calidad del software. En ocasiones los clientes pueden no haber estudiado detenidamente las cuestiones de calidad o sus gastos relacionados. ¿La calidad es meramente decorativa, o es consustancial al software? Si la respuesta se sitúa en un punto intermedio, como es casi siempre el caso, es cuestión de hacer del cliente parte del proceso de decisión y concienciarle totalmente tanto de los costes como de los beneficios. Idealmente, la mayor parte de estas decisiones serán adoptadas en el proceso de requerimientos de software (vd. El KA Requerimientos del software), sin embargo estas cuestiones pueden surgir en todas las etapas del ciclo de vida del software. No hay ninguna regla definida en cuanto a cómo deben ser adoptadas estas decisiones, pero el ingeniero de software debería ser capaz de presentar alternativas de calidad y sus correspondientes costes.

**1.3 .Modelos y Características de Calidad.**

La terminología para las características de calidad del software difiere de una taxonomía (o modelo de calidad de software) a otra, cada modelo quizás tenga un número diferente de niveles jerárquicos y un número total diferente de características. Varios autores han enunciado distintos modelos de características de calidad de software o atributos que pueden ser útiles para la negociación, planificación, y tasación de la calidad de productos software. ISO/IEC ha definido tres modelos relacionados de calidad de productos software (la calidad interna, la calidad externa, y la calidad en el empleo) (ISO9126-01) y un conjunto de partes relacionadas (ISO14598-98).

**1.3.1. La calidad del proceso en la ingeniería del software.**

La gestión de la calidad de software y la calidad de proceso en la ingeniería de software guarda relación directa con la calidad del producto software.

Los modelos y los criterios que evalúan las capacidades organizacionales en software son esencialmente la organización de proyecto y consideraciones de gestión, y, como tales, son tratados en los KAs relativos a Gestión en Ingeniería del Software y el Proceso en Ingeniería de Software. Desde luego, no es posible distinguir completamente la calidad del proceso de la calidad del producto. La calidad de proceso, tratada en el KA, de esta Guía, el Proceso en Ingeniería de Software, afecta a las características de calidad de los productos software, que a su vez repercuten en la calidad-en-el-uso tal y como es percibido por el cliente. Dos importantes estándares de calidad son TickIT [Llo03] y uno con impacto sobre la calidad de software, el estándar ISO9001-00, con sus directrices para su aplicación al software [ISO90003-04]. Otro estándar industrial en calidad del software es el CMMI [SEI02], también tratado en el KA Proceso en la Ingeniería de Software. CMMI pretende proporcionar directrices para mejorar procesos.

Específicamente las áreas de procesos relacionadas con la gestión de calidad son: a) Aseguramiento de la calidad en el proceso y el producto, (b) la verificación de proceso, y c) la validación de proceso. CMMI clasifica revisiones y auditorias como los métodos de verificación, y no como procesos específicos como (IEEE12207.0-96). Hubo inicialmente algún debate sobre si ISO9001 O CMMI deberían ser usados por ingenieros de software para asegurar la calidad. Este debate ha sido profusamente publicado, y, como resultado, se ha concluido que los dos resultan complementarios y que tener la certificación ISO9001 puede ayudar enormemente para alcanzar los niveles de madurez más altos del CMMI. [Dac01]. 1.3.2. Calidad de producto software. El ingeniero de software, ante todo, necesita determinar el Objetivo verdadero del software. En cuanto a esto, es de capital importancia tener presente los requerimientos del cliente y aquellos que estos incluyen como requerimientos de calidad, no únicamente los requerimientos funcionales. Así, el ingeniero de software tiene como responsabilidad obtener los requerimientos de calidad, que pueden no estar explícitos en un principio, tratar su importancia así como el nivel dificultad para alcanzarlos. Todos los procesos asociados a la Calidad de software (como por ejemplo, construcción, pruebas, mejora de la calidad) serán diseñados con estas exigencias en mente, y ello conlleva gastos adicionales. El estándar (ISO9126-01) define, para dos de sus tres modelos de calidad, las características de calidad mencionadas, las Sub-características, y las medidas que son útiles para Evaluación de calidad de producto de software. El significado del término "producto" es ampliado para incluir cualquier artefacto que es la salida de cualquier proceso empleado para construir el producto de software final. Como ejemplos de un producto cabe incluir, aunque no con carácter limitativo, una completa especificación del sistema, una especificación de requerimientos de software para un componente de software de un sistema, un módulo de diseño, código, documentación de prueba, o los informes producidos como consecuencia de tareas de análisis de calidad. Mientras la mayor parte del tratamiento de la calidad es descrito en términos del software final y funcionamiento del sistema, una ingeniería práctica responsable requiere que los productos intermedios relevantes para la calidad sean evaluados a lo largo de todo el proceso de ingeniería de software.

**1.4. Mejora de Calidad.**

La calidad de los productos software puede ser mejorada mediante un proceso iterativo de mejora continua que requiere control de dirección, coordinación, y retroalimentación de muchos procesos simultáneos: (1) los procesos de ciclo de vida de software, (2) El proceso de detección de error/defecto, retirada de los mismo y prevención, (y 3) el proceso de mejora de calidad. (Kin92) La teoría y conceptos presentes detrás de mejora de calidad, tales como la construcción en calidad, mediante la prevención y detección temprana de errores, mejora continua y enfoque en el cliente, son adecuados para la ingeniería de software. Estos conceptos están basados en el trabajo de expertos en calidad los cuales ha afirmado que la calidad de un producto está directamente conectada con la calidad del proceso empleado para crearlo. Aproximaciones tales como Total Quality Management (TQM) process of Plan, Do, Check, and Act (PDCA) son Instrumentos mediante los cuales conocer los objetivos de calidad. El apoyo a la gestión sustenta el proceso y la evaluación del producto así como las conclusiones resultantes. Entonces se desarrolla un programa de mejora identificando acciones detalladas y proyectos de mejora para ser gestionados en un plazo de tiempo factible. El apoyo a la gestión implica que cada proyecto de mejora tiene suficientes recursos para alcanzar el objetivo definido. El apoyo a la gestión ha ser solicitado con frecuencia mediante la implementación proactiva de actividades de comunicación. La participación de los equipos de trabajo, así como el apoyo a la gerencia media y los recursos asignados en el nivel de proyecto, son tratados en el KA Proceso de Ingeniería de Software.

**2. Procesos de Gestión de Calidad del Software**

La gestión de calidad de software (SQM) resulta de aplicación a todas las perspectivas de procesos de software, productos, y recursos. Esto define procesos, propietarios de proceso, y requerimientos para aquellos procesos, medidas del Proceso y sus correspondientes salidas, y canales de retroalimentación. (Art93) Los procesos de gestión de calidad del software consisten en numerosas actividades. Algunos de ellos pueden encontrar defectos directamente, mientras otros indican donde pueden resultar valiosas más revisiones. Estos últimos también son conocidos como actividades de "direct-defect-finding". Muchas actividades a menudo sirven para ambos propósitos. La planificación para la calidad de software implica: (1) Definición del producto requerido en términos de sus características calidad (descrito más detalladamente en, por ejemplo, El KA Gestión en Ingeniería del Software). (2) Planificación de los procesos para alcanzar el producto requerido (Descrito, por ejemplo, en los Kas, Diseño de Software y Construcción de Software). Estos aspectos difieren de, por ejemplo, los procesos mismos de planificación SQM, que evalúan las características de calidad planificadas versus la implementación actual de esa planificación.

**Los procesos de gestión de calidad de software deben dirigirse a como los buenos productos software van a satisfacer o satisfacen al cliente y las exigencias del personal implicado, a como proporcionan valor a los clientes y demás personal implicado, y proveen la calidad de software precisa para conocer los requerimientos del software.**

El SQM puede ser utilizado para evaluar productos intermedios así como el producto final. Algunos de los procesos específicos SQM están definidos en el estándar (IEEE12207.0-96):

* Procesos de aseguramiento de calidad.
* Procesos de verificación.
* Procesos de validación.
* Procesos de revisión.
* Procesos de auditoría.

Estos procesos incentivan la calidad y también permiten encontrar posibles problemas. Sin embargo presentan diferencias en cuanto a su énfasis. Los procesos SQM ayudan a asegurar una calidad de software óptima en un proyecto dado. Además proveen, como un subproducto, información general sobre gestión, incluyendo directrices de calidad para todo el proceso de ingeniería del software. Las Kas Proceso de la Ingeniería del Software y Gestión en Ingeniería del Software tratan sobre la calidad de los programas para la organización que desarrolla el software. El SQM puede proporcionar retroalimentación relevante para estas áreas. Los procesos SQM consisten en tareas y técnicas para indicar como los proyectos de software (por ejemplo, la gestión, el desarrollo, la gestión de configuración) están siendo puestos en práctica y la mejor manera para que los productos intermedios y los finales encuentren sus requerimientos especificados. Los resultados de estas tareas son recopilados en informes para la dirección antes de que sea tomada la acción correctiva. La gestión de un proceso SQM se desarrolla con la certeza de que los resultados de estos informes son exactos. Como se describe en este KA, los procesos SQM están estrechamente relacionados; pueden solaparse y hasta, en ocasiones, estar combinados. En su mayor parte parecen de naturaleza reactiva ya que toman los procesos como practicados y los productos como producidos; sin embargo tienen un papel principal en la fase de planificación, con carácter proactivo en términos de los procesos y los procedimientos precisos para alcanzar las características y grados de calidad requerida por los sujetos implicados en el software. La gestión del riesgo también puede jugar un papel importante en la entrega de software de calidad. La incorporación de un análisis de riesgo disciplinado y técnicas de gestión en los procesos de ciclo de vida de software puede incrementar el potencial para producir un producto de calidad. Refiérase al KA Gestión en la Ingeniería del Software para el material relacionado sobre gestión de riesgos.

**2.1. Aseguramiento de la Calidad del Software.**

Los procesos SQA proporcionan la garantía de que los productos software y los procesos en el ciclo de vida de proyecto son conformes a los requerimientos especificados por medio de la planificación, emitiendo, y realizando un conjunto de actividades para generar la confianza adecuada en que se está construyendo calidad dentro del software. Ello significa asegurar que el problema está clara y suficientemente identificado y que los requerimientos de la solución están correctamente definidos y expresados. El SQA procura mantener la calidad a lo largo de todo el desarrollo y mantenimiento del producto mediante la ejecución de una variedad de actividades en cada etapa que pueden permitir identificación temprana de problemas, un rasgo casi inevitable de cualquier actividad compleja. El papel del SQA en lo que concierne al proceso es asegurar que procesos planificados son apropiados y posteriormente implementados de acuerdo con la planificación, y se proveen procesos de medición relevantes para una adecuada organización. El plan SQA define el medio que será usado para asegurar que el software desarrollado para un producto específico satisface las exigencias del usuario y es de la máxima calidad posible dentro de las restricciones del proyecto. Con el objetivo de llevar esto acabo, primero debe asegurarse que el objetivo de calidad es claramente definido y entendido. En ello deben considerarse los planes de gestión, desarrollo, y mantenimiento para el software. Ver el estándar (IEEE730-98) para detalles. Las actividades y tareas específicas de calidad se elaboran, con sus gastos y exigencias de recursos, sus objetivos totales de gestión, y su programa en relación con aquellos objetivos de gestión en la ingeniería, el desarrollo, o planes de mantenimiento. El plan SQA debería ser compatible con el plan de gestión de configuración de software (refiérase al KA Gestión de Configuración de Software). El plan SQA identifica documentos, normas, prácticas, y convenciones que guían el proyecto y de qué manera serán comprobados y supervisados para asegurar adecuación y conformidad. El plan SQA también identifica medidas, técnicas estadísticas, procedimientos para el reporte de problemas así como la correspondiente acción correctiva, recursos tales como herramientas, técnicas, y metodologías, seguridad para el medio físico, formación, además de reportes y documentación SQA. Por otro lado, el plan SQA considera las actividades de garantía de calidad de software como cualquier otro tipo de actividad descrita en los proyectos de software, tales como la consecución de proveedor de software para el proyecto o el software de instalación comercial disponible (COTS), así como el servicio tras la entrega del software. También puede incluir criterios de aceptación así como reportes y actividades de gestión críticas para la calidad de software.

**2.2. Verificaciones y Validación.**

Con el propósito de ser breve, Verificación y Validación (V&V) son tratadas como un único asunto en esta Guía más que como dos asuntos separados tal y como se hace en el estándar (IEEE12207.0-96). La V&V del software es un acercamiento disciplinado a la evaluación de productos de software a lo largo de todo el ciclo de vida de producto. Un esfuerzo en V&V es esforzarse en asegurar que la calidad es construida dentro del software y que el software satisface exigencias de usuario " (IEEE1059-93). La V&V trata directamente la calidad de producto software y emplea técnicas de prueba que pueden localizar defectos de tal manera que estos puedan ser tratados. También evalúa los productos intermedios, como, y, en esta capacidad, los pasos intermedios de los procesos de ciclo de vida de software. El proceso V&V determina si productos de una actividad dada de desarrollo o mantenimiento se adecuan o no al correspondiente requisito de esa actividad, y si el producto final de software cumple o no cumple con su propósito fijado y converge o no con los requisitos del usuario. La Verificación es un intento para asegurar que el producto se construya correctamente, en el sentido que los productos resultantes de una actividad converjan con las especificaciones fijadas para los mismos en actividades previas. La validación es un intento por asegurar que se construye el producto correcto, es decir, que el producto satisface su propósito específico fijado. Tanto el proceso de comprobación como el proceso de validación empiezan temprano en la fase de desarrollo o mantenimiento. Proporcionan un examen de características claves del producto en relación con el producto inmediato predecesor y a las especificaciones con las que debe converger. El propósito de la planificación V&V es asegurar que cada recurso, papel y responsabilidad está claramente asignada. Los documentos del proyecto V&V resultantes describen varios recursos y sus papeles y actividades, así como técnicas y herramientas para ser usados. La comprensión de los objetivos diferentes de cada actividad V&V ayudará en la planificación cuidadosa de las técnicas y los recursos precisos para alcanzar sus respectivos objetivos. Estándares específicos (IEEE1012-98:s7 y IEEE1059-93: El apéndice A) que generalmente incluye un plan de V&V. El plan también considera la gestión, la comunicación, la política, y los procedimientos de las actividades V&V y su interacción, así como el reporte de defectos y exigencias de documentación.

**2.3. Revisiones y Auditorias**

Con el propósito de ser breve, revisiones y auditorías son tratadas como un solo tema en esta Guía, más que como dos temas separados tal y como se hace en (IEEE12207.0-96). La revisión y el proceso de auditoría son ampliamente definidos en (IEEE12207.0-96) y más detalladamente en (IEEE1028-97). Cinco tipos de revisiones o auditorias se presentan en el estándar IEEE1028-97:

* Revisiones de gestión.
* Revisiones técnicas.
* Inspecciones.
* Walk-throughs.
* Auditorias

**2.3.1. Revisiones de gestión.**

El objetivo de una revisión de gestión es supervisar el progreso, determinando el estado de planes y programas, requerimientos confirmados y su sistema de localización, o evaluar la efectividad de los enfoques de gestión empleados para lograr la idoneidad del objetivo. [IEEE1028-97]. Ello apoya decisiones sobre cambios y las acciones correctivas precisas durante un proyecto de software. Las revisiones de gestión determinan la idoneidad de los proyectos, programas, y requerimientos y supervisan su progreso o inconsistencias. Estas revisiones pueden ser realizadas sobre productos tales como informes de auditoría, informes de progreso, informes V&V, y proyectos de muchos tipos, incluyendo la gestión de riesgo, gestión del proyecto, gestión de configuración del software, seguridad del software, y la evaluación de riesgo, entre otros.

2.3.2. Revisiones técnicas

El propósito de una revisión técnica es evaluar el producto software para determinar si es idóneo para su correspondiente uso. El objetivo es identificar discrepancias con especificaciones aprobadas y estándares. El resultado debería proporcionar gestión con evidencias (o no) de que el producto converge con sus especificaciones y se adhiere a los estándares, y que los cambios están controlados. (IEEE1028- 97). En una revisión técnica deben estar establecidos los roles específicos: el que adopta las decisiones, un líder revisor, un registrador, y un personal técnico para apoyar las actividades de revisión. Una revisión técnica requiere que las entradas obligatorias estén en su lugar con el objeto de proceder a:

* Exposición de objetivos.
* Un producto software específico.
* El plan específico de gestión del proyecto.
* La lista de cuestiones claves asociadas al producto.
* El procedimiento de revisión técnica.

El equipo sigue el procedimiento de revisión. Un individuo técnicamente calificado presenta una descripción del producto, y el examen se lleva a cabo durante una o varias reuniones. La revisión técnica es completada una vez que todas las actividades catalogadas en el examen han sido completadas.

**2.3.3. Inspecciones.**

“*El propósito de una inspección es detectar e identificar anomalías en los productos software*”(IEEE1028-97).Existen dos importantes elementos diferenciadores entre inspección y revisión, son los siguientes:

1. Un individuo que mantiene una posición de dirección sobre cualquier miembro del equipo de inspección no participará en la inspección.

2.La inspección ha de ser llevada por un inspector con formación en inspecciones técnicas.

Las inspecciones de software siempre implican al autor de un producto intermedio o final, mientras otras revisiones puede que no. Las inspecciones también incluyen a un líder de inspección, un registrador, un lector, y un grupo (2 a 5) de inspectores. Los miembros de un equipo de inspección pueden poseer diferentes especializaciones, como especialización en el dominio, especialización en el método de diseño, o especialización en el lenguaje. Las inspecciones por lo general son llevadas a cabo sobre una relativamente pequeña sección del producto a la vez. Cada miembro del equipo debe examinar el producto software así como practicar otras revisiones de imputs con anterioridad a la reunión de revisión, quizás aplicando una técnica analítica (referida en la sección 3.3.3) en una pequeña porción del producto, o en todo el producto enfocando solo un aspecto, por ejemplo, interfaces. Cualquier anomalía encontrada se documenta y se envía al responsable de la inspección. Durante la inspección, el responsable de la inspección conduce la sesión y verifica que todos están preparados para la misma. Un herramienta comúnmente usada en las inspecciones es una lista de comprobación, con anomalías y preguntas pertinentes sobre las cuestiones de interés. La lista resultante a menudo clasifica las anomalías (refiérase a IEEE1044-93 para detalles) y se revisa por parte del equipo su exactitud e integridad. La decisión sobre el final de la inspección se corresponde con uno de los tres criterios siguientes:

1. No aceptar re-trabajo o como mucho un re-trabajo menor.

2. Aceptar con verificación del re-trabajo.

3. Re-inspección.

Típicamente las reuniones de inspección duran por lo general algunas horas mientras que las revisiones técnicas y auditorias son por lo general de mayor alcance y requieren más tiempo.

**2.3.4. Walk-throughs**

“*El objetivo de un Walk-throughs es evaluar un producto de software. Un Walk-throughs puede ser conducido para el objetivo de formar a una audiencia en cuanto a un producto de software.”* (IEEE1028-97) los objetivos principales son [IEEE1028-97]:

* Encontrar anomalías.
* Mejorar el producto software.
* Considerar implementaciones alternativa.
* Evaluar la conformidad con estándares y especificaciones.

El Walk-throughs es similar a una inspección, sin embargo, su desarrollo, por lo general, es menos formal. El Walk-throughs es organizado fundamentalmente por el ingeniero de software para dar a sus compañeros de equipo la oportunidad de repasar su trabajo, como una técnica de aseguramiento.

**2.3.5. Auditorias.**

“*El objetivo de una auditoría de software es proporcionar una evaluación independiente de la conformidad de productos software y procesos a regulaciones aplicables, estándares, directrices, planes, y procedimientos* " [IEEE1028-97]. La auditoría es una actividad formalmente organizada, con participantes que tienen papeles específicos, como el auditor jefe, otro auditor, un registrador, o un iniciador, e incluye a un representante de la organización auditada. La auditoría identificará los casos de no conformidad y producirá un informe requiriendo al equipo que adopte la acción correctiva correspondiente. Mientras puede haber muchos nombres formales para revisiones y auditorias como los identificados en el estándar (IEEE1028-97). La clave es qué revisiones y auditorias pueden practicarse sobre casi cualquier producto en cualquier etapa del proceso de mantenimiento o el desarrollo.

**3. Consideraciones Prácticas.**

**3.1. Requerimientos de calidad del software.**

**3.1.1. Factores de influencia.**

Varios factores influyen en la planificación, gestión, y selección de actividades de SQM y técnicas, incluyendo:

* El dominio del sistema en el cual el software residirá (seguridad crítica, misión crítica, negocio crítico).
* Requerimientos del Sistema y del software.
* Los componentes comerciales (externos) o estándar (internos) que serán usados por el sistema.
* Los estándares específicos de ingeniería del software específico aplicables.
* Los métodos y herramientas de software para ser usados en el desarrollo y el mantenimiento así como para la evaluación de calidad y mejora.
* El presupuesto, el personal, planes de organización, proyectos, y la planificación de todos los procesos.
* Los usuarios implicados y el empleo del sistema.
* El nivel de integridad del sistema La Información sobre estos factores de influencia como los procesos SQM son organizados y documentados, son seleccionadas como actividades SQM específicas, que recursos son necesarios, y que impondrá límites a los esfuerzos.

**3.1.2. Confiabilidad.**

En casos en los que el fracaso del sistema puede tener consecuencias sumamente severas, la confiabilidad total (en hardware, el software, y humanos) son la exigencia de calidad principal además de la funcionalidad básica. La confiabilidad del software incluye características tales como tolerancia al defecto, fiabilidad, seguridad, y usabilidad. La fiabilidad es también un criterio que puede ser definido en términos de confiabilidad (ISO9126). El cuerpo de conocimiento para sistemas debe ser sumamente confiable (" alta confianza " o “alta integridad en sistemas"). La terminología para los sistemas tradicionales mecánicos y eléctricos que no incluyen software ha sido importada para tratar amenazas o peligros, riesgos, integridad del sistema, y conceptos relacionados, y puede ser encontrada en las referencias citadas para esta sección.

**3.1.3. Niveles de integridad del software.**

El nivel de integridad se determina en base a las consecuencias posibles del fracaso del software y a la probabilidad de fracaso. Para el software en el que la seguridad o la fiabilidad son importantes, técnicas como el análisis de riesgo para la seguridad o el análisis de amenazas para la fiabilidad pueden ser usadas para desarrollar una actividad de planificación que se identificaría en donde se encuentren conflictos potenciales. Históricos de fracaso de software similar también puede ayudar en la identificación de qué técnicas serán las más útiles en el descubrimiento de defectos y evaluación de calidad. Se proponen niveles de integridad (por ejemplo, la gradación de integridad) en (IEEE1012-98).

**3.2. Caracterización de defectos.**

Los procesos SQM encuentran defectos. Caracterizar estos defectos conduce a un entendimiento del producto, facilita correcciones al proceso o al producto, e informa al gestor del proyecto o al cliente del estado del proceso o el producto. Existes muchas taxonomías defectuosas, y, mientras se ha intentado obtener un consenso en una taxonomía de defectos o fallos, la literatura indica que hay bastantes en uso (IEEE1044-93). La caracterización del Defecto (la anomalía) también es usada en auditorias y revisiones, el responsable de revisión a menudo presenta una lista de anomalías proporcionadas por miembros de equipo para su consideración en una reunión de revisión. Nuevos métodos de diseño y lenguajes se desarrollan, junto con avances en todas las tecnologías de software, las nuevas clases de defectos aparecen, y requieren mucho esfuerzo para interpretar clases previamente definidas. Rastreando defectos, el ingeniero de software está interesado tanto en el número como en el tipo de defectos. La Información sola, sin ninguna clasificación, no es realmente de ninguna utilidad en la identificación de las causas subyacentes de los defectos, pues los tipos específicos de problemas tienen que ser agrupados juntos para determinar cómo se gestionan. El asunto es establecer una taxonomía de defectos que sea significativa para la organización y los ingenieros de software. SQM descubre la información en todas las etapas de desarrollo de software y mantenimiento. Típicamente donde se usa la palabra "defecto" se refiere "a un fallo" tal y como se detalla más abajo. Sin embargo, diferentes culturas y normas pueden usar significados algo diferentes para estos términos, lo cual ha llevado a tentativas para definirlos. Definiciones parciales tomadas del estándar (IEEE610.12-90) son:

* Error: “*Una diferencia entre un resultado calculado y un resultado concreto*”.
* Faultt: “*Un paso, proceso, o definición de datos incorrecto en programa de computadora*”.
* Failure: “*El resultado [incorrecto] de un fault*”.
* Mistake: “*Un error humano que produce un resultado incorrecto*”

Los fracasos encontrados en pruebas como consecuencia de fallos de software están incluidos como defectos en esta sección.

Los modelos de fiabilidad son construidos en base a los fallos recogidos durante pruebas de software o procedentes de software en servicio, y de esta manera pueden ser usados para predecir futuros fracasos y asistir en decisiones sobre cuando detener las pruebas.

Una probable acción resultante de las conclusiones SQM es eliminar los defectos del producto durante su inspección. Otras acciones permiten alcanzar el valor completo de las conclusiones SQM. Estas acciones incluyen el análisis y el resumen de las conclusiones, y técnicas de medida de utilización para mejorar el producto y el proceso así como para rastrear los defectos y su eliminación. La mejora de proceso principalmente es tratada en el KA Proceso de Ingeniería de Software, junto con el proceso SQM como una fuente de información.

Los datos sobre las insuficiencias y defectos encontrados durante la implementación de técnicas SQM pueden ser perdidos a no ser que sean registrados. Para algunas técnicas (por ejemplo, revisiones técnicas, auditorias, inspecciones), los registradores están presentes para poner por escrito tal información, incluyendo cuestiones y decisiones. Cuando se utilizan herramientas automatizadas, las salidas de la herramienta pueden proporcionar la información sobre defectos. Los datos sobre defectos pueden ser recogidos y registrados sobre un formulario SCR (software change request) y posteriormente puede ser trasladado a algún tipo de base de datos, a mano o automáticamente, desde una herramienta de análisis. Proporcionan informes sobre defectos a la dirección de la organización.

**3.3. Técnicas de Gestión de Calidad del Software.**

Las técnicas SQM pueden categorizarse de diferentes formas: estáticas, Personal-intensivas, analíticas, dinámicas.

**3.3.1. Técnicas Estáticas.**

Las técnicas estáticas implican el examen de la documentación del proyecto y el software, y otra información sobre los productos de software, sin ejecutarlos. Estas técnicas pueden incluir actividades intensivas de personal (como se define en 3.3.2) o actividades analíticas (como se define en 3.3.3) conducidas por individuos, con o sin la ayuda de herramientas automatizadas.

**3.3.2. Técnicas intensivas de personal.**

La puesta en marcha de técnicas intensivas de personal, incluyendo revisiones y auditorias, puede variar de una reunión formal a una reunión informal o una situación de comprobación de escritorio, pero (por lo general, al menos) dos o más personas están implicadas. Puede resultar necesaria una preparación anticipada. Tanto los recursos como los ítems bajo examen pueden incluir listas de comprobación y son resultado de técnicas analíticas y pruebas. Estas actividades son tratadas en (IEEE1028-97) sobre revisiones y auditorias.

**3.3.3. Técnicas Analíticas.**

Un ingeniero de software generalmente aplica técnicas analíticas. A veces varios ingenieros de software usan la misma técnica, pero cada uno lo aplica a partes diferentes del producto. Algunas técnicas son llevadas a cabo por herramientas; las otras son manuales. Algunas pueden encontrar defectos directamente, pero generalmente son usadas para apoyar otras técnicas. Algunas técnicas también incluyen varias evaluaciones como la parte de análisis de calidad total. Ejemplos de tales técnicas incluyen el análisis de complejidad, controlan el análisis de flujo, y el análisis algorítmico.

Cada tipo de análisis tiene un objetivo específico, y no se aplican todos ellos a cada proyecto. Un ejemplo de una técnica de apoyo es el análisis de complejidad, que es útil para determinar si realmente el diseño o la implementación resultan demasiado complejos para desarrollar correctamente, realizar las pruebas, o el mantenimiento. Los resultados de un análisis de complejidad también pueden ser usados en test de desarrollo. Las técnicas para encontrar defectos como el análisis de flujo de control, también pueden utilizarse para apoyar otra actividad.

Para software con muchos algoritmos, el análisis algorítmico es importante, especialmente cuando un algoritmo incorrecto podría causar un resultado catastrófico. Hay demasiadas técnicas analíticas para listarlas todas aquí. La lista y referencias proporcionadas pueden ofrecer ideas en la selección de una técnica, así como sugerencias para posteriores lecturas.

Otros tipos, más formales, de técnicas analíticas se conocen como métodos formales. Son usados para verificar requerimiento de software y diseños. Las pruebas de corrección corresponden a las partes críticas de software. Han sido usadas, sobre todo, en la verificación de las partes cruciales de sistemas críticos, tales como la seguridad y exigencias de fiabilidad.

**3.3.4. Técnicas Dinámicas.**

Las diferentes clases de técnicas dinámicas son ejecutadas durante todo el desarrollo y el mantenimiento de software.

Generalmente, son técnicas de testeo, pero técnicas tales como la simulación, la comprobación de modelo, y la ejecución simbólica pueden ser consideradas dinámicas. La lectura de código es considerada una técnica estática, pero ingenieros de software experimentados puede ejecutar el código tal y como lo leen. En este sentido, la lectura de código también puede considerarse una técnica dinámica. Esta discrepancia en la categoría indica que personal con papeles diferentes en la organización puede considerar y aplicar estas técnicas de manera diferente.

Algunas pruebas, así mismo, pueden ejecutarse en el proceso de desarrollo, el proceso SQA, o el proceso de V&V, de nuevo dependen de la organización de proyecto. Debido a que la planificación SQM incluye testeo, esta sección incluye algunos comentarios sobre pruebas.

El KA Pruebas de Software proporciona el tratamiento y referencias técnicas a la teoría, técnicas para pruebas, y automatización.

**3.3.5. Pruebas.**

Los procesos de aseguramiento descritos en SQA y V&V examinan todos los output de la especificación de requerimientos del software con el objeto de asegurar su trazabilidad, consistencia, completitud, corrección y ejecución. Esta comprobación también incluye los output de los procesos de desarrollo y mantenimiento, recopilando, analizando y midiendo los resultados. SQA asegura la planificación, desarrollo e implementación de determinados tipos de pruebas, y V&V desarrolla planes de prueba, estrategias, casos y procedimientos.

Las pruebas son tratadas detalladamente en el KA. testeo de Software. Dos tipos de pruebas pueden incluirse en el ámbito de SQA y V\*V, por razón de su responsabilidad por la calidad de los materiales usados en el proyecto:

* Evaluación y prueba de herramientas que serán usadas en el proyecto (IEEE1462-98).
* Prueba de Conformidad (o revisión de prueba de conformidad) de componentes y productos COTS que se usaran en el producto; allí ahora existe un estándar para paquetes de software (IEEE1465-98).

En ocasiones una organización independiente V&V puede ser requerida para monitorear el proceso de pruebas y a veces para atestiguar la ejecución actual con el objeto de asegurar que este es llevado a cabo conforme a los procedimientos especificados. También se puede apelar a V&V para evaluar las pruebas en sí mismas: adecuación a los proyectos y procedimientos, y suficiencia y exactitud de resultados.

Otro tipo de pruebas que puede incluirse en el ámbito de la organización V&V es el de pruebas de terceros.

Este tercero no es el desarrollador, tampoco está relacionado en modo alguno con el desarrollo del producto. En cambio, el tercero es un “*Facility*” independiente, por lo general acreditado por alguna autoridad. Su objetivo es el de probar un producto respecto de su conformidad con un conjunto específico de exigencias.

**3.4. Medición de calidad del software**

Los modelos de calidad del software a menudo incluyen métricas para determinar el grado de calidad de cada característica alcanzada por el producto.

Si estos modelos se seleccionan correctamente, las métricas pueden apoyar la calidad del software de muchas maneras (entre otros aspectos de los procesos de ciclo de vida de software).

Pueden ayudar en procesos de toma de decisiones de gestión. Pueden encontrar áreas problemáticas y cuellos de botella en procesos de software y pueden ayudar a los ingenieros de software a evaluar la calidad de su trabajo respecto de objetivos SQA y respecto de procesos a largo plazo de mejora de calidad.

Con el incremento de la complejidad del software las cuestiones sobre calidad van más allá de si el software es capaz de alcanzar objetivos de calidad mensurables. Existen algunas áreas en las que las métricas soportan SQM directamente. Esto incluye asistencias en la decisión de cuando detener las pruebas. Para ello los modelos de fiabilidad y pruebas, resultan útiles usando datos de fallos y fracasos.

El coste de los procesos SQM es una cuestión que casi siempre se suscita cuando se adopta la decisión de cómo debería organizarse un proyecto. A menudo, los modelos genéricos de coste usados, están basados en cuando se encuentra y cuanto esfuerzo se precisa para fijar el defecto en relación con el hallazgo del defecto temprano en el proceso de desarrollo. Los datos de proyecto pueden ofrecer una mejor idea del coste. La información relacionada puede ser encontrada en los KAs el Proceso de Ingeniería de Software y Gestión en Ingeniería del Software.

Finalmente los informes SQM en sí mismos proveen información valiosa no sólo sobre estos procesos, sino también sobre cómo pueden perfeccionarse todos los procesos de ciclo de vida.

Mientras las métricas para características de calidad y rasgos de producto pueden resultar útiles en sí mismas (por ejemplo, el número de requerimientos defectuosos o la proporción de requerimientos defectuosos), pueden aplicarse técnicas matemáticas y gráficas para ayudar en la interpretación de esa métricas.

* Basadas estadísticamente basado (por ejemplo, Pareto analysis, run charts, scatter plots, normal distribution.
* Pruebas Estadísticas (por ejemplo, binomial test, chisquared test ).
* El análisis de Tendencia.
* Predicción (por ejemplo, modelos de fiabilidad).

Las técnicas basadas en estadísticas y las pruebas a menudo proporcionan una imagen de las áreas más problemáticas del producto software examinado. Los cuadros y gráficos resultantes resultan ayudas de visualización que los gestores con poder de decisión pueden utilizar para concentrar recursos donde resulten más necesarios. Los resultados del análisis de tendencia pueden indicar que una programación no ha sido respetada, tales como las pruebas, o que ciertas clases de fallos se intensificarán a no ser que se adopte alguna acción correctiva en el desarrollo. Las técnicas de predicción asisten en la planificación del periodo de prueba y en la predicción del fracaso. Más información sobre medición en general pude encontrarse en los Kas Proceso de Ingeniería de Software y Gestión en Ingeniería del Software. Información más específica sobre medición en pruebas se presenta en el Ka Testeo de Software. También ayudan a comprender las tendencias en cómo están trabajando las técnicas de detección, y cómo están progresando los procesos de desarrollo y mantenimiento. La medición de la cobertura de la prueba ayuda a estimar cuanto esfuerzo en términos de prueba queda por hacer, y predecir los posibles defectos restantes. De estos métodos de medición, pueden desarrollarse los perfiles del defecto para un dominio específico de la aplicación.

Así, para el siguiente sistema de software dentro de esa organización, los perfiles podrán utilizarse para guiar los procesos SQM, es decir, para focalizar esfuerzo allí donde sea más probable que puedan surgir problemas. Semejantemente, los puntos de referencia, o los defectos típicos de ese dominio, pueden servir como ayuda en la determinación de cuándo estará el producto listo para su entrega.

El estudio de cómo usar datos de SQM para mejorar procesos de desarrollo y mantenimiento puede encontrarse en los Kas Gestión en Ingeniería del Software y Procesos en Ingeniería del Software.