CAPÍTULO 5
PRUEBAS DEL SOFTWARE

ACRÓNIMOS

SRET	Pruebas Orientadas a la Confiabilidad del Software

INTRODUCCIÓN

Hacer pruebas es una actividad que tiene el objetivo de evaluar y mejorar la calidad del producto, identificando defectos y problemas.

Las pruebas del software consisten en verificar el comportamiento de un programa *dinámicamente* a través de un grupo *finito* de casos de prueba, debidamente *seleccionados* del, típicamente, ámbito de ejecuciones infinito, en relación al comportamiento *esperado*. En la definición anterior las palabras en itálica se corresponden con aspectos esenciales en la identificación del "Área de Conocimiento de las Pruebas del Software". En particular:

- *Dinámicamente*: Este termino significa que hacer pruebas siempre supone ejecutar el programa con entrada de datos (valorados). Para precisar, es preciso afirmar que la entrada de valores no es siempre suficiente para definir una prueba, dado que un sistema complejo y no determinista podría tener diferentes comportamientos con las misma entrada de datos, dependiendo del estado en el que se encuentre. En cualquier caso, en este KA, mantendremos el término de "entrada de datos", asumiendo la convención de que el término incluye un estado del sistema específico, en los casos en que sea necesario. Existen otras técnicas complementarias a las pruebas, aunque diferentes, descritas en el KA sobre la Calidad del Software.
- *Finito*: Incluso en programas sencillos, teóricamente podría haber tantas pruebas que realizar, que hacer pruebas exhaustivas podría llevar meses o años. Esta es la razón por la que en la práctica el grupo completo de pruebas se podría considerar infinito. Hacer pruebas siempre supone un compromiso entre recursos y calendarios de trabajo limitados, por un lado, y necesidades inherentes de pruebas ilimitadas, por otro.
- *Seleccionados*: La diferencia esencial entre las distintas técnicas de pruebas propuestas se encuentra en cómo se escoge el conjunto de pruebas. Los ingenieros informáticos deben ser conscientes de que criterios de selección distintos pueden producir grados de efectividad muy diferentes. La forma de identificar el criterio de selección de pruebas más apropiado para un conjunto de condiciones particulares es un problema complejo; en la práctica se usa la experiencia en el diseño de pruebas y técnicas de análisis de riesgo.
- *Esperado*: Debería se posible, aunque a veces no sea fácil, decidir si el resultado observado de la ejecución de un programa es aceptable o no, porque si no el esfuerzo de realizar las pruebas sería inútil. El comportamiento observado se puede comprobar con los resultados esperados por el usuario (normalmente conocido como pruebas de validación), con las especificaciones (pruebas de verificación), o, finalmente, con el comportamiento anticipado de requerimientos implícitos o expectativas razonables. Vea más detalles en el KA de Requerimientos del Software, punto 6.4 *Pruebas de Aceptación*.

La apreciación de las pruebas del software ha evolucionado hacia una forma más constructiva. Ya no se asume que realizar pruebas es una tarea que empieza solamente cuando la fase de programación se ha completado, y que tiene el único propósito de detectar errores. Las pruebas del software se ven ahora como una actividad que debería estar presente durante todo el proceso de desarrollo y mantenimiento y es en sí misma una parte importante de la construcción del producto. Es más, la planificación de las pruebas debería empezar en las primeras etapas del proceso de requisitos, mientras que los planes y procedimientos de pruebas deberían desarrollare y posiblemente refinarse sistemáticamente según avanza el desarrollo. La planificación de las pruebas y las propias actividades de diseño constituyen una información muy útil que ayuda a los diseñadores de software a identificar debilidades potenciales (tales como elementos del diseño que han pasado desapercibidos, contradicciones de diseño, u omisiones o ambigüedades en la documentación).

En la actualidad se considera que la prevención es la actitud adecuada en lo que respecta a la calidad: obviamente es mejor evitar problemas que solucionarlos. Realizar pruebas debe verse como un medio para verificar, no sólo si la prevención ha sido efectiva, si no para identificar fallos en aquellos casos en los que, por alguna razón, no lo ha sido.

Aunque quizás sea obvio, vale la pena reconocer que, incluso después de una campaña de pruebas extensiva, el software aún podría contener errores. Las acciones de mantenimiento correctivas proporcionan la solución a errores en el software después de que éste ha sido entregado. El KA del Mantenimiento del Software aborda los temas relacionados con el mantenimiento del software.

En el KA del Mantenimiento del Software (véase punto 3.3 *Técnicas de Gestión de Calidad del Software*), las técnicas de gestión de la calidad del software se dividen entre técnicas *estáticas* (sin ejecución de código) y técnicas *dinámicas* (con ejecución de código). Ambas categorías son útiles. Este KA se centra en técnicas dinámicas.

Las pruebas del software también están relacionadas con la construcción del software (véase la sección 3.4 *Construcción de Pruebas*). Las pruebas de unidad y de integración están íntimamente relacionadas con la construcción del software, si no son parte de la misma.

DIVISIÓN DE TEMAS

La descomposición de temas en el KA de las Pruebas del Software se muestra en la Figura 1.

La primera subárea describe los *Fundamentos de las Pruebas del Software*. Cubre las definiciones básicas del área de pruebas del software, la terminología básica y los términos clave, así como las relaciones con otras actividades.

La segunda subárea, *Niveles de Pruebas*, está formada por dos puntos ortogonales: el primero (2.1) enumera los niveles en que tradicionalmente se subdividen las pruebas para software grande, mientras que el segundo (2.2) considera las pruebas para situaciones o propiedades específicas y se conoce como *objetivos de las pruebas*. No todos los tipos de pruebas se pueden aplicar a todos los productos de software, tampoco se han enumerado todos los tipos posibles.

El objeto y los objetivos de las pruebas determinan la forma en que un grupo de pruebas se identifica, en lo que se refiere a su consistencia – *cuántas pruebas son suficientes para conseguir el objetivo especificado* – y su composición – *qué casos de prueba se deberían seleccionar para conseguir el objetivo especificado* (aunque normalmente la parte "para conseguir el objetivo especificado" es implícita y sólo se usa la primera parte de las dos frases anteriores). Los criterios para responder a la primera cuestión se denominan criterios de *idoneidad las pruebas*, mientras que los que se refieren a la segunda cuestión se denominan criterios de *selección de las pruebas*.

En las últimas décadas se han desarrollado varias *Técnicas de Pruebas* y aún se están proponiendo nuevas técnicas. El conjunto de pruebas comúnmente aceptadas están enumeradas en la subárea 3.
Las *Mediciones de Pruebas* se enumeran en la subárea 4. Finalmente, los aspectos relacionados con el *Proceso de las Pruebas* están enumeradas en la subárea 5

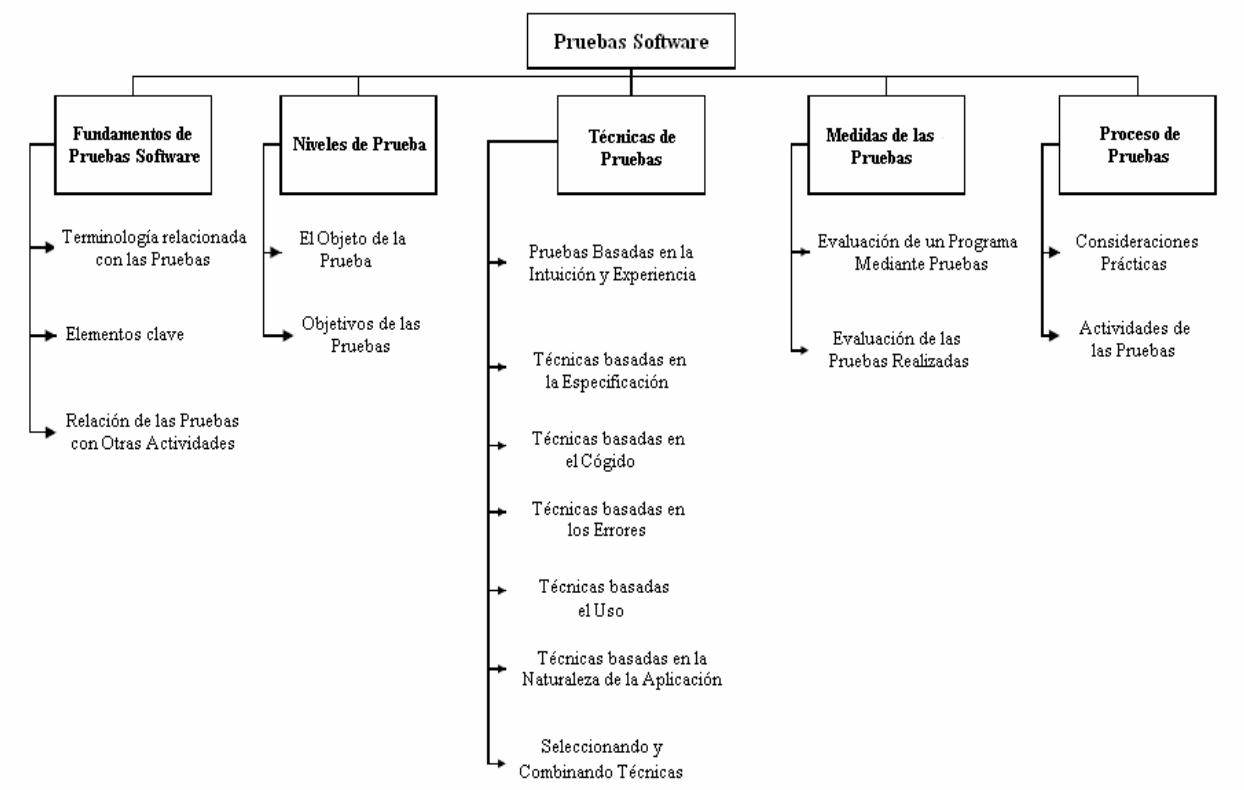

Figura 1 División de los temas para el KA de las Pruebas del Software

1. Fundamentos de las Pruebas del Software

1.1 Terminología relacionada con las pruebas

1.1.1 Definiciones de pruebas y terminología relacionada

[Bei90:c1; Jor02:c2; Lyu96:c2s2.2] (IEEE610.12-90)

Vea una introducción detallada del KA de las Pruebas del Software en las referencias recomendadas.

1.1.2 Errores Vs. Fallos [Jor02:c2; Lyu96:c2s2.2; Per95:c1; Pfl01:c8] (IEEE610.12-90; IEEE982.1-88)

En la bibliografía sobre Ingeniería del Software se usan diversos términos para describir un funcionamiento incorrecto, particularmente *falta*, *error*, *fallo* y otros términos. Esta terminología se define detalladamente en el estándar IEEE 610.12-1990, *Standard Glossary of Software Engineering Terminology* (IEEE610-90) y también se discute en el KA de la Calidad del Software. Es esencial distinguir claramente entre la causa de un funcionamiento incorrecto, en cuyo caso se usan términos como *error* y *defecto*, y los efectos no deseados observados en los servicios proporcionados por un sistema, que se llamarán *fallos*. Hacer pruebas puede descubrir fallos, pero es el error el que se puede, y se debe, eliminar.

En cualquier caso, debería aceptarse que no es siempre posible identificar unívocamente las causas de un fallo. No existe ningún criterio teórico que pueda usarse para determinar qué error produce el fallo observado. Podría decirse que hay que arreglar el error para eliminar el problema, pero otros cambios también podrían funcionar. Para evitar ambigüedades, algunos autores prefieren hablar de *entradas que causan fallos* (Fra98) en vez de errores – o lo que es lo mismo, aquellos grupos de entradas de datos que hacen que el fallo aparezca.

1.2 Cuestiones clave

1.2.1 Criterios de selección de pruebas/Criterios de idoneidad de pruebas (o finalización de pruebas) [Pfl01:c8s7.3; Zhu97:s1.1] (Wey83; Wey91; Zhu97)

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. Véase el apartado *Finalización* de la sección 5.1 *Consideraciones prácticas*.

1.2.2 Efectividad de las pruebas/Objetivos para las pruebas [Bei90:c1s1.4; Per95:c21] (Fra98)

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 [Bei90:c1; Kan99:c1]

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 [Bei90:c1] (Ber96, Wey83)

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 [Kan99:c2] (How76)

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 [Bei90:c3]

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

[Bei90:c3, c13] (Bac90; Ber96a; Voa95)

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, como se describe en (Bac90); por otro, se define como la probabilidad, posiblemente cualificada estadísticamente, de que los errores del software queden expuestos durante las pruebas, si es erróneo, tal y como se describe en (Voa95, Ber96a). 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. Véase también el KA de la Calidad del Software, punto 2. *Proceso de Gestión de la Calidad del Software*. [Bei90:c1; Per95:c17] (IEEE1008-87)
- Pruebas vs. Pruebas de Validez y Verificación Formal [Bei90:c1s5; Pfl01:c8].
- Pruebas vs. Depuración. Véase también el KA de la Construcción del Software, punto 3.4 *Pruebas de la construcción* [Bei90:c1s2.1] (IEEE1008-87).
- Pruebas vs. Programación. Véase también el KA de la Construcción del Software, punto 3.4 *Pruebas de la construcción* [Bei90:c1s2.1] (IEEE1008-87).
- Pruebas y Certificación (Wak99).

2 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. [Bei90:c1; Jor02:c12;Pfl01:c8] 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
[Bei90:c1; Per95:c17; Pfl01:c8s7.3] (IEEE1008-87)

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
[Jor02:c13, 14; Pfl01:c8s7.4]

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
[Jor02:c15; Pfl01:c9]

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. Véase el KA de Requisitos del Software para más información acerca de requisitos funcionales y no funcionales.

2.2 *Objetivos de las pruebas*
[Per95:c8; Pfl01:c9s8.3]

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 [Per95:c10; Pfl01:c9s8.5] (IEEE12207.0-96:s5.3.9)

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 especifica, 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 [Per95:c9; Pfl01:c9s8.6]

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 [Kan99:c13]

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 [Kan99:c7; Per95:c8] (Wak99)

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 [Lyu96:c7; Pfl01:c9s.8.4] (Pos96)

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 (véase también el punto 4.1.4 Pruebas de ejecución, evaluación de la confiabilidad)

2.2.6 Pruebas de regresión [Kan99:c7; Per95:c11, c12; Pfl01:c9s8.1] (Rot96)

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 (Bei90) 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 [Per95:c17; Pfl01:c9s8.3] (Wak99)

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 [Per95:c17; Pfl01:c9s8.3]

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 [Per95:c17; Pfl01:c9s8.3]

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 [Kan99:c8; Pfl01:c9s8.3]

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 [Per95:c8; Pfl01:c9s8.3]

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 [Bec02]

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 [Kan99:c1]

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. [Kan01:c3]

3.2 Técnicas basadas en la especificación

3.2.1 Particiones de equivalencia [Jor02:c7; Kan99:c7]

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 [Jor02:c6; Kan99:c7]

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 ls entrada de datos, para comprobar la robustez del programa con entradas de datos erróneas e inesperadas.

3.2.3 Tablas de decisión
[Bei90:c10s3] (Jor02)

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 posible. Una técnica relacionada es el *gráfico causa-efecto*. [Pfl01:c9]

3.2.4 Basadas en máquinas de estado finito
[Bei90:c11; Jor02:c8]

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
[Zhu97:s2.2] (Ber91; Dic93; Hor95)

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. Existen métodos para derivar casos de prueba de especificaciones basadas en el modelo (Dic93, Hor95) o especificaciones algebraicas. (Ber91)

3.2.6 Pruebas aleatorias
[Bei90:c13; Kan99:c7]

En este caso las pruebas se generan de una manera completamente aleatoria, lo que no debe confundirse con las pruebas estadísticas basadas en el perfil operativo descritas en el punto 3.5.1 *Perfil Operativo*. 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
[Bei90:c3; Jor02:c10] (Zhu97)

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 datos
[Bei90:c5] (Jor02; Zhu97)

En las pruebas basadas en el flujo de datos, el gráfico de flujos de control tiene anotaciones con información acerca de como 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)
[Bei90:c3; Jor02:c5]

Aunque no es una técnica en si 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* (Mor90)

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
[Kan99:c7]

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
[Per95:c17; Zhu97:s3.2-s3.3]

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 si 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.

3.5.2 Pruebas Orientadas a la Confiabilidad del Software
[Lyu96:c6]

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 [Jor02:c17; Pfl01:c8s7.5] (Bin00)
- Pruebas basadas en componentes
- Pruebas para Internet
- Pruebas para GUI [Jor20]
- Pruebas para programas concurrentes (Car91)
- Pruebas de conformidad de protocolos (Pos96; Boc94)
- Pruebas para sistemas de tiempo real (Sch94)
- Pruebas para sistemas de seguridad crítica (IEEE1228-94)

3.7 Seleccionando y combinando técnicas

3.7.1 Funcional y estructuralmente
[Bei90:c1s.2.2; Jor02:c2, c9, c12; Per95:c17] (Pos96)

Las técnicas de pruebas basadas en las especificaciones y el código se contrastan frecuentemente como pruebas funcionales vs estructurales. Estas 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 a comprobado que remarcan diferentes tipos de problemas. Estas técnicas se pueden combinar, dependiendo del presupuesto para pruebas.

3.7.2 Deterministas vs aleatorias
(Ham92; Lyu96:p541-547)

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.

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 si 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. Se proporciona más información acerca de medidas para programas en el KA de la Gestión del la Ingeniería del Software, punto 6, *Medidas en la ingeniería del software*. Se puede encontrar más información en el KA de la Gestión del la Ingeniería del Software, punto 4, *Proceso y medidas del producto*.

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. Las medidas relacionadas con el proceso de gestión de pruebas se abordan en el punto 5.1 *Consideraciones prácticas*.

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 [Bei90:c7s4.2; Jor02:c9] (Ber96; IEEE982.1-88)

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 [Bei90:c2; Jor02:c2; Pfl01:c8] (Bei90; IEEE1044-93; Kan99; Lyu96)

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. Se puede encontrar más información en el KA de la Calidad del Software, punto 3.2 *Caracterización de defectos*. Existe un estándar del IEEE acerca de como clasificar “anomalías del software” (IEEE1044-93).

4.1.3 Densidad de fallos [Per95:c20] (IEEE982.1-88; Lyu96:c9)

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 [Pfl01:c9] (Pos96:p146-154)

Una estimación estadística de al 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 [Lyu96:c7; Pfl01:c9] (Lyu96:c3, c4)

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 [Jor02:c9; Pfl01:c8] (IEEE982.1-88)

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 [Pfl01:c8] (Zhu97:s3.1)

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 cual de los errores artificiales aparezca y cuantos 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 [Zhu97:s3.2-s3.3]

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 [Jor02:c9, c12; Per95:c17; Zhu97:s5]

(Fra93; Fra98; Pos96: p64-72)

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 [Bei90:c13s3.2; Pfl01:c8]

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 [Kan01]

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 [Bec02: III; Per95:c1-c4; Pfl01:c9] (IEEE1074-97; IEEE12207.0-96:s5.3.9, s5.4.2, s6.4, s6.5)

Las actividades de pruebas realizadas a diferentes niveles (véase punto 2, *Niveles de pruebas*) 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 [Bei90:c13s5; Kan99:c12; Per95:c19; Pfl01:c9s8.8] (IEEE829-98)

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 [Bei90:c13s2.2-c13s2.3; Kan99:c15; Per95:c4; Pfl01:c9]

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 [Per95:c4, c21] (Per95: Appendix B; Pos96:p139-145; IEEE982.1-88)

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
[Bei90:c2s2.4; Per95:c2]

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 si 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. Véase también el punto 1.2.1 *Criterios de selección de pruebas/Criterios de idoneidad de pruebas*.

5.1.8 Reutilización de pruebas y patrones de pruebas [Bei90:c13s5]

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
[Kan99:c12; Per95:c19; Pfl01:c8s7.6] (IEEE829-98:s4; IEEE1008-87:s1-s3)

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
[Kan99:c7] (Pos96:c2; IEEE1008-87:s4, s5)

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
[Kan99:c11]

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
[Bei90:c13; Kan99:c11] (IEEE1008-87:s6, s7)

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
[Per95:c20,c21] (Pos96:p18-20, p131-138)

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 [Kan99:c5; Per95:c20] (IEEE829-98:s9-s10)

Las actividades de las pruebas se pueden añadir a un diario de pruebas para identificar cuando una prueba se a 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 (véase el KA de la Gestión de la Configuración del Software, punto 3, *Control de la configuración del software*)

5.2.7 Seguimiento de defectos [Kan99:c6]

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 que aspectos de la ingeniería del software necesitan mejorase y la efectividad de análisis y pruebas anteriores.

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	[Bec02]	[Bei09]	[Jor02]	[Kan99]	[Kan01]	[Lyu96]	[Per95]	[Pfl01]	[Zhu97]
1. Fundamentos de las Pruebas Software									
1.1Terminología relacionada									
Definiciones de pruebas y terminología		c1	c2			c2s2.2			
Errores Vs. Fallos			c2			c2s2.2	c1	c8	
1.2 Cuestiones Clave									
Criterios de selección de Pruebas/ Criterios de idoneidad de Pruebas								c8s7.3	s1.1
Efectividad de las Pruebas/ Objetivos de las Pruebas		c1s1.4					c21		
Realizar pruebas para la identificación de defectos		c1		c1					
El problema del oráculo		c1							
Limitaciones teóricas y prácticas de las pruebas				c2					
El problema de los caminos no alcanzables		c3							
Posibilidad de hacer pruebas		c3, c13							
1.3 Relación de las pruebas con otras actividades									
2. Niveles de Pruebas									
2.1El objeto de la prueba		c1	c13					c8	
Pruebas de Unidad		c1					c17	c8s7.3	
Pruebas de Integración			c13, c14					c8s7.4	
Pruebas del sistema			c15					c9	
2.2 Objetivos de las Pruebas							c8	c9s8.3	
Pruebas de aceptación/ calificación							c10	c9s8.5	
Pruebas de Instalación							c9	c9s8.6	
Pruebas Alfa y Beta				c13					
Pruebas de conformidad/Pruebas funcionales/Pruebas de corrección				c7			c8		
Materialización de la confiabilidad y evaluación						c7		c9s8.4	
Pruebas de Regresión				c7			c11, c12	c9s8.1	
Pruebas de Rndimiento							c17	c9s8.3	
Pruebas de Desgaste							c17	c9s8.3	
Pruebas de Continuidad									
Pruebas de Recuperación							c17	c9s8.3	
Pruebas de Configuración								c9s8.3	
Pruebas de Facilidad de Uso				c8			c8	c9s8.3	
Desarrollo dirigido por Pruebas	III								

	[Bec02]	[Bei09]	[Jor02]	[Kan99]	[Kan01]	[Lyu96]	[Per95]	[Pfl01]	[Zhu97]
3. Técnicas de Pruebas									
3.1Pruebas Basadas en la Intuición y Experiencia									
Pruebas ad hoc				c1					
Pruebas por Exploración					c3				
3.2 Basadas en la Especificación									
Particiones de Equivalencia			c7	c7					
Análisis de los Valores Límite			c6	c7					
Tablas de Decisión		c10s3						c9	
Basadas en Máquinas de Estado Finito		c11	c8						
Basadas en las especificaciones formales									s2.2
Pruebas aleatorias		c13		c7					
3.3 Basadas en el código									
Criterio basado en el Flujo de Control		c3	c10					c8	
Criterio basado en el Flujo de Datos		c5							
Modelos de referencia para pruebas basadas en código		c3	c5						
3.4 Basadas en errores									
Conjeturar errores				c7					
Pruebas por mutación							c17		s3.2, s3.3
3.5 Basadas en el uso									
Perfil operativo			c15			c5		c9	
Pruebas Orientadas a la Confiabilidad del Sofware						c6			
3.6 Basadas en la Naturaleza de la Aplicación									
Pruebas Orientadas a Objetos									
Basadas en Componentes			c17				c8s7.5		
Pruebas para Internet									
Pruebas para GUI			c20						
Pruebas para Programas Concurrentes									
Pruebas de conformidad de Protocolos									
Pruebas para Sistemas de Tiempo Real									
Pruebas para Sistemas de Seguridad Crítica									
3.7 Seleccionando y combinando técnicas						c17			
Funcional y Estructuralmente		c1s2.2	c1, c11s11.3						
Deterministas Vs. Aleatorias									

	[Bec02]	[Bei09]	[Jor02]	[Kan99]	[Kan01]	[Lyu96]	[Per95]	[Pfl01]	[Zhu97]
4. Medidas de las Pruebas									
4.1 Evaluación de un programa									
Medidas para ayudar en la planificación y diseño de pruebas de programas		c7s4.2	c9						
Tipos de errores, clasificación y Estadísticas		c2	c1					c8	
Densidad de fallos							c20		
Vida de las pruebas								c9	
Modelos de crecimiento de la Confiabilidad						c7		c9	
4.2 Evaluación de las pruebas realizadas									
Medidas de la cobertura/completitud			c9					c8	
Introducción de errores								c8	
Puntuación de la mutación									s3.2, s3.3
Comparación y efectividad relativa de las técnicas			c8, c11				c17		s5
5. El proceso de las pruebas									
5.1 Consideraciones prácticas									
Actitudes y programación egoless		c13s3.2						c8	
Guías para las pruebas	III				c5				
Gestión del proceso de pruebas							c1-c4	c9	
Documentación y productos de las pruebas		c13s5		c12			c19	c9s8.8	
Equipo de Pruebas Interno vs. Equipo Independiente		c13s2.2, c1s2.3		c15			c4	c9	
Estimación Coste/Esfuerzo y otras Medidas del Proceso							c4, c21		
Finalización		c2s2.4					c2		
Reutilización de pruebas y patrones de pruebas		c13s5							
5.2 Actividades de Pruebas									
Planificación				c12			c19	c87s7.6	
Generación de casos de Prueba				c7					
Desarrollo del entorno de Pruebas				c11					
Ejecución		c13		c11					
Evaluación de los resultados							c20, c21		
Notificación de Problemas/ Diario de pruebas				c5			c20		
Seguimiento de los defectos				c6					

REFERENCIAS RECOMENDADAS PARA LA GESTIÓN DEL SOFTWARE

[Bec02] K. Beck, *Test-Driven Development by Example*, Addison-Wesley, 2002.

[Bei90] B. Beizer, *Software Testing Techniques*, International Thomson Press, 1990, Chap. 1-3, 5, 7s4, 10s3, 11, 13.

[Jor02] P. C. Jorgensen, *Software Testing: A Craftsman's Approach*, second edition, CRC Press, 2004, Chap. 2, 5-10, 12-15, 17, 20.

[Kan99] C. Kaner, J. Falk, and H.Q. Nguyen, Testing Computer Software, second ed., John Wiley & Sons, 1999, Chaps. 1, 2, 5-8, 11-13, 15.

[Kan01] C. Kaner, J. Bach, and B. Pettichord, *Lessons Learned in Software Testing*, Wiley Computer Publishing, 2001.

[Lyu96] M.R. Lyu, *Handbook of Software Reliability Engineering*, Mc-Graw-Hill/IEEE, 1996, Chap. 2s2.2, 5-7.

[Per95] W. Perry, *Effective Methods for Software Testing*, John Wiley & Sons, 1995, Chap. 1-4, 9, 10-12, 17, 19-21.

[Pfl01] S. L. Pfleeger, *Software Engineering: Theory and Practice*, second ed., Prentice Hall, 2001, Chap. 8, 9.

[Zhu97] H. Zhu, P.A.V. Hall and J.H.R. May, "Software Unit Test Coverage and Adequacy," *ACM Computing Surveys*, vol. 29, iss. 4 (Sections 1, 2.2, 3.2, 3.3), Dec. 1997, pp. 366-427.

APÉNDICE A. LISTA DE LECTURAS ADICIONALES

(Bac90) R. Bache and M. Müllerburg, "Measures of Testability as a Basis for Quality Assurance," *Software Engineering Journal*, vol. 5, March 1990, pp. 86-92.

(Bei90) B. Beizer, *Software Testing Techniques*, International Thomson Press, second ed., 1990.

(Ber91) G. Bernot, M.C. Gaudel and B. Marre, "Software Testing Based On Formal Specifications: a Theory and a Tool," *Software Engineering Journal*, Nov. 1991, pp.387-405.

(Ber96) A. Bertolino and M. Marrè, "How Many Paths Are Needed for Branch Testing?" *Journal of Systems and Software*, vol. 35, iss. 2, 1996, pp. 95-106.

(Ber96a) A. Bertolino and L. Strigini, "On the Use of Testability Measures for Dependability Assessment," *IEEE Transactions on Software Engineering*, vol. 22, iss.2, Feb. 1996, pp. 97-108.

(Bin00) R.V. Binder, *Testing Object-Oriented Systems Models, Patterns, and Tools*, Addison-Wesley, 2000.

(Boc94) G.V. Bochmann and A. Petrenko, "Protocol Testing: Review of Methods and Relevance for Software Testing," presented at *ACM Proc. Int'l Symp. on Software Testing and Analysis* (ISSTA '94), Seattle, Wash., 1994.

(Car91) R.H. Carver and K.C. Tai, "Replay and Testing for Concurrent Programs," *IEEE Software*, March 1991, pp. 66-74.

(Dic93) J. Dick and A. Faivre, "Automating the Generation and Sequencing of Test Cases from Model-Based Specifications," presented at *FME '93: Industrial-Strength Formal Methods*, LNCS 670, Springer-Verlag, 1993.

(Fran93) P. Frankl and E. Weyuker, "A Formal Análisis of the Fault Detecting Ability of Testing Methods," *IEEE Transactions on Software Engineering*, vol. 19, iss. 3, March 1993, p. 202.

(Fran98) P. Frankl, D. Hamlet, B. Littlewood, and L. Strigini, "Evaluating Testing Methods by Delivered Reliability," *IEEE Transactions on Software Engineering*, vol. 24, iss. 8, August 1998, pp. 586-601.

(Ham92) D. Hamlet, "Are We Testing for True Reliability?" *IEEE Software*, July 1992, pp. 21-27.

(Hor95) H. Horcher and J. Peleska, "Using Formal Specifications to Support Software Testing," *Software Quality Journal*, vol. 4, 1995, pp. 309-327.

(How76) W. E. Howden, "Reliability of the Path Analysis Testing Strategy," *IEEE Transactions on Software Engineering*, vol. 2, iss. 3, Sept. 1976, pp. 208-215.

(Jor02) P.C. Jorgensen, *Software Testing: A Craftsman's Approach*, second ed., CRC Press, 2004.

(Kan99) C. Kaner, J. Falk, and H.Q. Nguyen, "Testing Computer Software," second ed., John Wiley & Sons, 1999.

(Lyu96) M.R. Lyu, *Handbook of Software Reliability Engineering*, Mc-Graw-Hill/IEEE, 1996.

(Mor90) L.J. Morell, "A Theory of Fault-Based Testing," *IEEE Transactions on Software Engineering*, vol. 16, iss. 8, August 1990, pp. 844-857.

(Ost88) T.J. Ostrand and M.J. Balcer, "The Category-Partition Method for Specifying and Generating Functional Tests," *Communications of the ACM*, vol. 31, iss. 3, June 1988, pp. 676-686.

(Ost98) T. Ostrand, A. Anodide, H. Foster, and T. Goradia, "A Visual Test Development Environment for GUI Systems," presented at *ACM Proc. Int'l Symp. On Software Testing and Analysis* (ISSTA '98), Clearwater Beach, Florida, 1998.

(Per95) W. Perry, *Effective Methods for Software Testing*, John Wiley & Sons, 1995.

(Pfl01) S.L. Pfleeger, *Software Engineering: Theory and Practice*, second ed., Prentice-Hall, 2001, Chap. 8, 9.

(Pos96) R.M. Poston, *Automating Specification-Based Software Testing*, IEEE, 1996.

(Rot96) G. Rothermel and M.J. Harrold, "Analyzing Regression Test Selection Techniques," *IEEE Transactions on Software Engineering*, vol. 22, iss. 8, Aug. 1996, p. 529.

(Sch94) W. Schütz, "Fundamental Issues in Testing Distributed Real-Time Systems," *Real-Time Systems Journal*, vol. 7, iss. 2, Sept. 1994, pp. 129-157.

(Voa95) J.M. Voas and K.W. Miller, "Software Testability: The New Verification," *IEEE Software*, May 1995, pp. 17-28.

(Wak99) S. Wakid, D.R. Kuhn, and D.R. Wallace, "Toward Credible IT Testing and Certification," *IEEE Software*, July-Aug. 1999, pp. 39-47.

(Wey82) E.J. Weyuker, "On Testing Non-testable Programs," *The Computer Journal*, vol. 25, iss. 4, 1982, pp. 465-470.

(Wey83) E.J. Weyuker, "Assessing Test Data Adequacy through Program Inference," *ACM Trans. On Programming Languages and Systems*, vol. 5, iss. 4, October 1983, pp. 641-655.

(Wey91) E.J. Weyuker, S.N. Weiss, and D. Hamlet, "Comparison of Program Test Strategies," presented at *Proc. Symp. on Testing, Analysis and Verification (TAV 4)*, Victoria, British Columbia, 1991.

(Zhu97) H. Zhu, P.A.V. Hall, and J.H.R. May, "Software Unit Test Coverage and Adequacy," *ACM Computing Surveys*, vol. 29, iss. 4, Dec. 1997, pp. 366-427.

APÉNDICE B. LISTA DE ESTÁNDARES

(IEEE610.12-90) IEEE Std 610.12-1990 (R2002), *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, 1990.

(IEEE829-98) IEEE Std 829-1998, *Standard for Software Test Documentation*, IEEE, 1998.

(IEEE982.1-88) IEEE Std 982.1-1988, *IEEE Standard Dictionary of Measures to Produce Reliable Software*, IEEE, 1988.

(IEEE1008-87) IEEE Std 1008-1987 (R2003), *IEEE Standard for Software Unit Testing*, IEEE, 1987.

(IEEE1044-93) IEEE Std 1044-1993 (R2002), *IEEE Standard for the Classification of Software Anomalies*, IEEE, 1993.

(IEEE1228-94) IEEE Std 1228-1994, *Standard for Software Safety Plans*, IEEE, 1994.

(IEEE12207.0-96) IEEE/EIA 12207.0-1996 // ISO/IEC12207:1995, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, IEEE, 1996.