UTN
Proyecto

Testing de Software - Calidad de productos de Software

Autor: Gabriela Muñoz

Índice

1 Fundamentos del Testing

- ✓ **La importancia económica del software**

El funcionamiento de maquinaria y equipamiento depende en gran medida del software.
No es posible imaginar grandes sistemas, en el ámbito de las finanzas ni el control del tráfico automotor, entre otros, funcionando sin software.

- ✓ **Calidad del Software**

Cada vez más, la calidad software se ha convertido en un factor determinante del éxito de sistemas y productos técnicos o comerciales.

- ✓ **Pruebas para la mejora de la calidad**

Las pruebas y revisiones aseguran la mejora de la calidad de productos de software así como de la calidad del proceso de desarrollo en sí.

Riesgo
No todo el software tiene el mismo nivel de riesgo y no todos los problemas tienen el mismo impacto cuando ocurren.

1.1 Calidad de Software

La calidad está constituida por:

- ✓ Atributos funcionales de calidad:

  Funcionalidad: correctitud y completitud de los requisitos del usuario.

- ✓ Atributos NO funcionales de calidad:

  Fiabilidad: el sistema mantendrá su capacidad y funcionalidad a lo largo de un período de tiempo.
  Usabilidad: fácil de usar, fácil de aprender, conforme a normas y uso intuitivo.
  Portabilidad: fácil de instalar y desinstalar, y configurar parámetros.

1.2 Calidad

Grado en el cual un componente, sistema o proceso satisface requisitos especificados y/o necesidades y expectativas del usuario/cliente.

Rol del Testing en el desarrollo, mantenimiento y operaciones de software (K2)
La prueba rigurosa de sistemas y la documentación pueden ayudar a reducir el riesgo de que ocurran problemas en un entorno operacional y a contribuir a la calidad del sistema

de software, si los defectos encontrados son corregidos antes que el sistema sea lanzado para uso operacional.
La prueba de software puede ser también requerida para cumplir con requisitos legales o contractuales o con estándares específicos de la industria.

1. Caso de Prueba (Test Case/Test Basis)
2. Código (source code)
3. Depuración (debugging)
4. Desarrollo de Software (Software development)
5. Requisitos (requirement)
6. Revisión (Review)

1. Caso de prueba (Según IEEE std 610)

- Precondiciones
- Conjunto de Valores de entrada
- Conjunto de resultados esperados
- Forma en la cual se debe ejecutar el caso de prueba y verificar los resultados
- PosCondiciones esperadas
  Conjunto de documentos que definen los requisitos de un componente o sistema.

2. Código (source code) Según IEE 610

Instrucciones de ordenador y definiciones de datos expresados en un lenguaje de programación o en una forma de salida generada por un ensamblador, compilador u traductor

3. Depuración (debugging)
Proceso de encontrar, analizar y eliminar las causas de los fallos en el software
Localización y corrección de defectos en el código fuente

4. Desarrollo de Software (Software development)
Proceso/secuencia de actividades cuyo objetivo es desarrollar un sistema basado en un computador

5. Requisito (requirement) Según IEEE 610
Condición o capacidad necesaria para un usuario con el objeto de solucionar un problema o lograr un objetivo que debe ser alcanzado o poseído por un sistema o componente de un sistema, para satisfacer un contrato, estándar, especificación, u otro documento impuesto formalmente

6. Revisión (Review) Según IEEE 1028
Evaluación de un producto o del estado de un proyecto para detectar discrepancias con los resultados planificados y para recomendar mejoras.

1.3 ¿Por qué es necesaria la prueba?

1.3.1 Términos

Bug, defecto, error, falla, avería, calidad, riesgo, software, prueba.

1.3.2 Contexto de los sistemas de software

Los sistemas de software son una parte creciente de la vida, desde las aplicaciones de negocio (ej. transacciones bancarias) hasta productos de consumo (ej. autos). La mayoría de la gente ha tenido una experiencia con software que no funcionó como se esperaba. El software que no funciona correctamente puede conducir a muchos problemas, incluyendo pérdida de dinero, tiempo o reputación del negocio y podría incluso causar lesión o muerte.

1.3.3 Causas de defectos de software

Un ser humano puede hacer un error, el cual produce un defecto (avería, bug) en el código, en el software o en un sistema, o en un documento. Si un defecto en el código es ejecutado, el sistema fallará en hacer lo que debía hacer (o hacer algo que no debía), causando una falla. Los defectos en el software, sistemas o documentos pueden resultar en fallas, mas no todos los defectos hacen eso.

Los defectos ocurren porque los seres humanos son falibles y porque hay presión de tiempo, código complejo, complejidad de infraestructura, tecnologías cambiadas y/o muchas interacciones del sistema.

Las fallas pueden ser causadas por condiciones ambientales asimismo: la radiación, magnetismo, campos electrónicos y polución pueden causar averías en el soporte lógico incorporado o influenciar la ejecución del software al cambiar las condiciones del hardware.

1.3.4 Rol de la prueba en el desarrollo, mantenimiento y operaciones de software

La prueba rigurosa de sistemas y la documentación pueden ayudar a reducir el riesgo de que ocurran problemas en un entorno operacional y a contribuir a la calidad del sistema de software, si los defectos encontrados son corregidos antes que el sistema sea lanzado para uso operacional.

La prueba de software puede ser también requerida para cumplir con requisitos legales o contractuales o con estándares específicos de la industria.

1.3.5 Prueba y calidad

Con la ayuda de la prueba, es posible medir la calidad del software en términos de defectos encontrados, para requisitos y características de software tanto funcionales como no funcionales

(ej. fiabilidad, usabilidad, eficiencia, mantenibilidad). Para mayor información sobre la prueba no funcional vea el capítulo 2; para mayor información sobre características de software vea "Ingeniería de Software – Calidad del Producto de Software" (ISO 9126).

La prueba puede dar confianza en la calidad del software si encuentra pocos o ningún defecto. Una prueba diseñada apropiadamente que pase, reduce el nivel total de riesgo de un sistema. Cuando la prueba encuentra defectos, la calidad del sistema de software se incrementa cuando defectos son arreglados.

Las lecciones deberían aprenderse de proyectos anteriores. Al entender las causas raíz de los defectos encontrados en otros proyectos, los procesos pueden ser mejorados, lo cual a su vez debería prevenir que aquellos defectos vuelvan a ocurrir y, como una consecuencia, mejorar la calidad de futuros sistemas.

La prueba debería estar integrada como una de las actividades de garantía de calidad (es decir, junto con estándares de desarrollo, entrenamiento y análisis de defectos).

1.3.6 ¿Cuánta prueba es suficiente?

El decidir cuánta prueba es suficiente debería tomar cuenta del nivel de riesgo, incluyendo productos técnicos y del negocio y riesgos del proyecto, y las restricciones del proyecto tales como tiempo y presupuesto.

La prueba debería proporcionar suficiente información a las partes interesadas para realizar decisiones informadas sobre el lanzamiento del software o sistema que está siendo probado, para el próximo paso de desarrollo o entrega a los clientes.

1.4 ¿Qué es prueba?

1.4.1 Términos

Código, depuración, desarrollo (de software), requisito, revisión, base de pruebas, caso de prueba, prueba, objetivos de prueba.

1.4.2 Introducción

Una percepción común de la prueba consiste en que solamente consiste de realizar pruebas, es decir ejecutar el software. Esto es parte de la prueba, más no todas las actividades de prueba.

Las actividades de la prueba existen antes y después de la ejecución de la prueba, las actividades tales como planificación y control, eligiendo condiciones de prueba, diseñando casos de la prueba y comprobando los resultados, evaluando criterios de finalización, informando sobre el proceso de prueba y el sistema bajo prueba, y finalizando o terminación (ej. después de que una fase de

prueba ha sido completada). La prueba también incluye la revisión de documentos (incluyendo el código fuente) y análisis estático.

Tanto como la prueba dinámica y la prueba estática pueden ser utilizadas como un medio para alcanzar objetivos similares y proporcionarán la información para mejorar tanto el sistema que se probará como los procesos de desarrollo y de prueba.

Puede haber diferentes objetivos de prueba:

- ✓ encontrar defectos.
- ✓ ganar confianza sobre el nivel de calidad y proporcionar información.
- ✓ prevenir defectos.

El proceso de pensamiento de diseñar pruebas temprano en el ciclo de vida (que verifica la base de prueba vía el diseño de prueba) puede ayudar a evitar que los defectos sean introducidos en el código. Las revisiones de documentos (ej. los requisitos) también ayudan a prevenir los defectos que aparecen en el código.

Los diversos puntos de vista en la prueba toman en cuenta diferentes objetivos. Por ejemplo, en las pruebas de desarrollo (ej. Pruebas de componente, de integración y de sistema), el objetivo principal puede ser causar cuantas fallas como sea posible para que los defectos en el software sean identificados y puedan ser arreglados. En las pruebas de aceptación, el objetivo principal puede ser confirmar que el sistema trabaja según lo esperado, para ganar confianza de que cumple los requisitos. En algunos casos el objetivo principal de la prueba puede ser evaluar la calidad del software (sin la intención de arreglar los defectos), para dar información a las partes interesadas del riesgo de lanzar el sistema en un momento dado. Las pruebas de mantenimiento a menudo incluyen pruebas de que ningún error nuevo ha sido introducido durante el desarrollo de los cambios. Durante las pruebas operacionales, el objetivo principal puede ser evaluar las características del sistema tales como fiabilidad o disponibilidad.

La depuración y la prueba son diferentes. La prueba puede mostrar las fallas que son causadas por los defectos. La depuración es la actividad de desarrollo que identifica la causa de un defecto, repara el código y comprueba que el defecto haya estado arreglado correctamente. Las pruebas de confirmación subsiguientes por parte de un probador garantizan que el arreglo resuelve en sí la falla. La responsabilidad para cada actividad es muy diferente, es decir, los probadores prueban y los desarrolladores depuran.

1.5 Principios generales de prueba

1.5.1 Términos

Prueba exhaustiva.

1.5.2 Principios

Un número de principios de prueba se han sugerido durante los últimos 40 años y ofrecen pautas generales comunes para toda prueba.

1.5.2.1 Principio 1 - La prueba muestra la presencia de defectos

La prueba puede mostrar que los defectos están presentes, pero no puede probar que no hay defectos. La prueba reduce la probabilidad de los defectos no descubiertos restantes en el software pero, incluso si no se encuentran defectos, no es una prueba de corrección.

1.5.2.2 Principio 2 - La prueba exhaustiva es imposible

Probar todo (todas las combinaciones de entradas y de precondiciones) no es factible a excepción de trivial casos. En vez de la prueba exhaustiva, usamos el riesgo y las prioridades para enfocar los esfuerzos de prueba.

1.5.2.3 Principio 3 - Prueba temprana

Las actividades de prueba deberían comenzar tan pronto como sea posible en el ciclo de vida del desarrollo del software o del sistema y deberían estar enfocados en los objetivos definidos.

1.5.2.4 Principio 4 - Agrupamiento de defectos

Un número pequeño de módulos contiene la mayoría de los defectos descubiertos durante la prueba de pre-lanzamiento o mostrar las fallas más operacionales.

1.5.2.5 Principio 5 - Paradoja del pesticida

Si las mismas pruebas se repiten una y otra vez, eventualmente el mismo conjunto de casos de prueba no encontrará más cualquier nuevo bug. Para superar esta "paradoja del pesticida", los casos de prueba necesitan ser repasadas y revisadas regularmente, y nuevas y diversas necesitan ser escritas al ejercicio diferente partes del software o del sistema potencialmente para encontrar más defectos.

1.5.2.6 Principio 6 - La prueba es dependiente del contexto

La prueba se hace diferentemente en diversos contextos. Por ejemplo, se prueba el software de seguridad crítica diferentemente de un sitio de e-comercio.

1.5.2.7 Principio 7 - Falacia de la ausencia de errores

Encontrar y arreglar defectos no ayuda si el sistema construido es inutilizable y no cumple las necesidades y expectativas de los usuarios.

1.6 Proceso fundamental de prueba

1.6.1 Términos

Pruebas de confirmación, criterios de salida, incidente, pruebas de regresión, base de prueba, condición de prueba, cobertura de prueba, datos de prueba, datos de prueba, ejecución de prueba, registro de prueba, estrategia de prueba, informe de resumen de la prueba, testware.

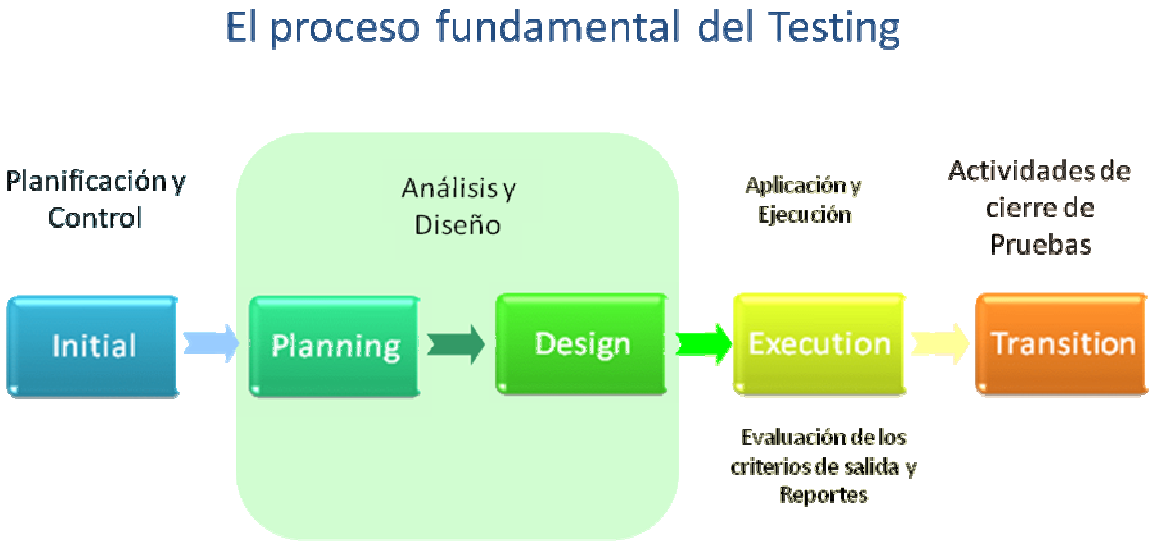
1.6.2 Introducción

La parte más visible de las pruebas es ejecutar las pruebas. Pero para ser eficaces y eficientes, los planes de prueba deben también incluir el tiempo que se tomará al planificar las pruebas, diseñar los casos de prueba, prepararse para la ejecución y evaluar el estado.

El proceso de prueba fundamental consiste de las siguientes actividades principales:

- ✓ planificación y control;
- ✓ análisis y diseño;
- ✓ implementación y ejecución;
- ✓ evaluar criterios salida e informar;
- ✓ actividades de cierre de prueba.

Aunque lógicamente secuenciales, las actividades en el proceso pueden superponerse u ocurrir concurrentemente.



La parte más visible de las pruebas es la ejecución. Pero para ser eficaz y eficiente, los Test Plans deben incluir también el tiempo para gastar en la planificación de las pruebas, el diseño de casos de prueba, la preparación para su ejecución y evaluar su estado.

1.6.3 Panificación y control de prueba

La planificación de la prueba es la actividad de verificar la misión de la prueba, definiendo los objetivos de la prueba y la especificación de las actividades de la prueba para satisfacer los objetivos y la misión.

El control de la prueba es la actividad en curso de comparar el progreso real contra el plan, y de informar el estado, incluyendo las desviaciones del plan. Implica tomar las acciones necesarias para cumplir la misión y objetivos del proyecto. Para controlar la prueba, ésta debe ser supervisada a través del proyecto. La planificación de la prueba considera la información del monitoreo y control de actividades.

Las tareas de planificación de prueba tienen las siguientes tareas principales:

- ✓ Determinar el alcance y los riesgos, e identificar los objetivos de la prueba.
- ✓ Determinar el acercamiento de prueba (técnicas, elementos de prueba, cobertura, identificar y realizar interfaz con los equipos involucrados en las pruebas, testware).
- ✓ Determinar los recursos de prueba requeridos (por ejemplo gente, entorno de prueba, PCs).
- ✓ Implementar la política de prueba y/o estrategia de prueba.
- ✓ Programar los análisis de prueba y las tareas de diseño.
- ✓ Programar la implementación, ejecución y evaluación de prueba.
- ✓ Determinar los criterios de salida.

Las tareas de control de prueba tienen las siguientes tareas principales:

- ✓ Medir y analizar los resultados.
- ✓ Monitorear y documentar el progreso, la cobertura de prueba y los criterios de salida.
- ✓ Iniciación de las acciones correctivas.
- ✓ Realizar decisiones.

1.6.4 Análisis y diseño de prueba

El análisis y diseño de prueba es la actividad donde los objetivos generales de prueba son transformados en condiciones de prueba y diseños de prueba tangibles.

El análisis y diseño de prueba tiene las siguientes tareas principales siguientes:

- ✓ Revisar la base de prueba (tales como requisitos, arquitectura, diseño, interfaces).
- ✓ Posibilidad de probar evaluación de los objetos de base de prueba y de prueba.
- ✓ Identificar las condiciones de prueba o los requisitos de prueba y los datos de prueba requeridos basado en el análisis de los elementos de prueba, la especificación, comportamiento y estructura.
- ✓ Diseñar las pruebas.
- ✓ Evaluar testabilidad de los requisitos y del sistema.
- ✓ Diseñar la configuración del entorno de prueba e identificar cualquier infraestructura y herramientas requeridas.

1.6.5 Implementación y ejecución de prueba

La implementación y ejecución de prueba es la actividad donde las condiciones de prueba son transformadas en casos de prueba y testware y el entorno es configurado.

La implementación y ejecución de prueba tienen las siguientes tareas principales:

- ✓ Desarrollar y priorizar casos de prueba, crear datos de prueba, escribir procedimientos de prueba y, opcionalmente, preparar armaduras de pruebas y escribir scripts de prueba automatizados.
- ✓ Crear conjuntos de prueba de los casos de prueba para la ejecución de prueba eficiente.
- ✓ Verificar que el entorno de prueba haya sido configurado correctamente.
- ✓ Ejecutar los casos de prueba ya sea manualmente o mediante el uso de herramientas de ejecución de prueba, de acuerdo con la secuencia planeada.
- ✓ Registrar el resultado de la ejecución de prueba y grabar las identidades y versiones del software bajo prueba, herramientas de prueba y testware.
- ✓ Comparar los resultados reales con los resultados esperados.
- ✓ Informar discrepancias como incidentes y analizarlos para establecer su causa (por ejemplo, un defecto en el código, en datos de prueba especificados, en el documento de prueba o un error en la forma en que la prueba fue ejecutada).
- ✓ Repetir las actividades de prueba como resultado de la acción tomada para cada discrepancia. Por ejemplo, la re-ejecución de una prueba que falló previamente para confirmar un arreglo (pruebas de confirmación), ejecución de una prueba corregida y/o ejecución de pruebas para asegurarse que los defectos no han sido introducidos en áreas no cambiadas del software o que el arreglo del defecto no reveló otros defectos (pruebas de regresión).

1.6.6 Evaluación de criterios de salida y reporte

Evaluar los criterios de salida es la actividad donde la ejecución de prueba es evaluada contra los objetivos definidos. Esto debe hacerse para cada nivel de prueba.

Evaluar los criterios de prueba tiene las siguientes tareas principales:

- ✓ Comprobar los registros de prueba contra los criterios de salida especificados en la planificación de prueba.
- ✓ Evaluar si más pruebas son necesitadas o si los criterios de salida especificados deberían ser cambiados.
- ✓ Escribir un reporte de resumen de prueba para las partes interesadas.

1.6.7 Actividades de cierre de prueba

Las actividades de cierre de prueba recolectan datos de las actividades de prueba completadas para consolidar experiencia, testware, hechos y números. Por ejemplo, donde un sistema de software es lanzado, un proyecto de prueba es completado (o cancelado), un hito ha sido alcanzado o un lanzamiento de mantenimiento ha sido completado.

Las actividades de cierre de prueba incluyen las siguientes tareas principales:

- ✓ Comprobar cuales entregables planeados han sido entregados, el cierre de los informes de incidentes o el aumento de registros de cambio para cualquiera que permanezca abierto y la documentación de la aceptación del sistema.
- ✓ Finalizar y archivar el testware, el entorno de prueba y la infraestructura de prueba para reutilización posterior.
- ✓ Entrega del testware a la organización de mantenimiento.
- ✓ Analizar las lecciones aprendidas para futuros lanzamientos y proyectos y la mejora de la madurez de pruebas.

1.7 Cuanto Testing es necesario?

- ✓ El testing exhaustivo es imposible.
- ✓ Testear todas las combinaciones de entradas y precondiciones no es factible.
- ✓ Para enfocar el testing nos debemos basar en Riesgos y Prioridades.

1.8 La psicología de prueba

1.8.1 Términos

Prueba independiente.

1.8.2 Introducción

El modo de pensar a ser usado mientras se prueba y revisa es diferente al aquel usado mientras se analiza o desarrolla. Con el correcto modo de pensar los desarrolladores pueden probar su propio código, mas la separación de su responsabilidad hacia un probador es realizada

típicamente para ayudar a enfocar el esfuerzo y para proporcionar beneficios adicionales, tales como una visión independiente por recursos de prueba profesionales y entrenados. La prueba independiente pude ser llevada a cabo en cualquier nivel de prueba.

Un cierto grado de independencia (evitando la tendencia del autor) es a menudo más efectivo al encontrar defectos y fallas. La independencia no es, sin embargo, un reemplazo para la familiaridad y los desarrolladores pueden encontrar eficientemente muchos defectos en su propio código. Varios niveles de independencia pueden ser definidos:

- ✓ Pruebas diseñadas por la(s) persona(s) que escribió (escribieron) el software bajo prueba (bajo nivel de independencia).
- ✓ Pruebas diseñadas por otra(s) persona(s) (por ejemplo, del equipo de desarrollo).
- ✓ Pruebas diseñadas por una(s) persona(s) de un grupo organizacional diferente (por ejemplo, un grupo de prueba independiente.
- ✓ Pruebas diseñadas por una(s) persona(s) de una organización o compañía diferente (es decir subcontratación o certificación por un organismo externo).

La gente y los proyectos son conducidos por objetivos. La gente tiende a alinear sus planes con los objetivos establecidos por la gestión y otras partes interesadas, por ejemplo, para encontrar defectos o para confirmar que el software trabaja. Por lo tanto, es importa definir claramente los objetivos de la prueba.

Identificar fallas durante la prueba puede ser percibido como criticismo contra el producto y contra el autor. La prueba es, por lo tanto, a menudo vista como una actividad destructiva, aunque es muy constructiva en la gestión de riesgos del producto. Buscar fallas en un sistema requiere curiosidad, pesimismo profesional, un ojo crítico, atención al detalle, buena comunicación con los pares de desarrollo y experiencia en que basar la conjetura de error.

Si los errores, defectos o fallas son comunicados en una forma constructiva, malos sentimientos entre los probadores y los analistas, diseñadores y desarrolladores pueden ser evitados. Esto se aplica a la revisión así como en la prueba.

El probador y el líder de prueba necesitan buenas habilidades interpersonales para comunicar información factual sobre los defectos, el progreso y los riesgos, en una forma constructiva. Para el autor del software o del documento, la información del defecto puede ayudarlo a mejorar sus habilidades. Los defectos encontrados y arreglados durante la prueba ahorrarán tiempo y dinero más tarde y reducirán riesgos.

Los problemas de comunicación pueden ocurrir, particularmente si los probadores son vistos como mensajeros de noticias no deseadas sobre los defectos. Sin embargo, existen varias formas para mejorar la comunicación y las relaciones entre los probadores y los demás:

- ✓ Iniciar con la colaboración en lugar de batallas – recordar a todos de la meta común de mejores sistemas de calidad.

- ✓ Comunicar los descubrimientos sobre el producto en una forma neutral enfocada en hechos sin criticar a la persona que lo creo, por ejemplo, escribir informes de incidentes factuales y objetivos y revisar los descubrimientos.
- ✓ Intentar de comprender como se siente la otra persona y porque reacciona como lo hace.
- ✓ Confirmar que la otra persona ha entendido que es lo que usted ha dicho y viceversa.

Referencias

1.1.5 Black, 2001, Kaner, 2002
1.2 Beizer, 1990, Black, 2001, Myers, 1979
1.3 Beizer, 1990, Hetzel, 1998, Myers, 1979
1.4 Hetzel, 1998
1.4.5 Black, 2001, Craig, 2002
1.5 Black, 2001, Hetzel, 1998

2 Prueba a través del ciclo de vida del software

Objetivos de aprendizaje para probar a través del ciclo de vida del software

Los objetivos identifican lo que podrá hacer después de la finalización de cada módulo.

2.1 Modelos del desarrollo del software

- ✓ Entender la relación entre desarrollo, actividades de prueba y productos de trabajo en el ciclo de vida del desarrollo y dar ejemplos basados en las características de proyecto y producto y en el contexto.
- ✓ Reconocer el hecho que los modelos de desarrollo de software deben ser adaptados al contexto del proyecto y a las características del producto.
- ✓ Recordar las razones para diferentes niveles de prueba y características de buenas pruebas en cualquier modelo de ciclo de vida.

2.2 Niveles de prueba

- ✓ Comparar los diferentes niveles de prueba: objetivos principales, típicos objetos de prueba, típicas metas de prueba (por ejemplo, funcionales o estructurales) y productos de trabajo relacionados, gente que prueba, tipos de defectos y fallas a ser identificados.

2.3 Tipos de pruebas: los objetivos de la prueba

- ✓ Comparar cuatro tipos de prueba de software (funcionales, no funcionales, estructurales y relacionados al cambio) por ejemplo.
- ✓ Reconocer que pruebas funcionales y estructurales ocurren en cualquier nivel de prueba.
- ✓ Identificar y describir los tipos de prueba no funcionales basados en requisitos no funcionales.
- ✓ Identificar y describir los tipos de prueba basados en el análisis de una estructura o arquitectura del sistema de software.
- ✓ Describir el propósito de la prueba de confirmación y de la prueba de regresión.

2.4 Pruebas de mantenimiento

- ✓ Comprara las pruebas de mantenimiento (probando un sistema existente) para probar una nueva aplicación con respecto a los tipos de prueba, disparadores para pruebas y cantidad de pruebas.
- ✓ Identificar las razones para la prueba de mantenimiento (modificación, migración y retiro).
- ✓ Describir el rol de la prueba de regresión y del análisis de impacto en el mantenimiento.

2.5 Modelos de desarrollo de software

2.5.1 Términos

Comercial/listo para la venta (COTS), modelo de desarrollo incremental, nivel de prueba, validación, verificación, Modelo-V.

2.5.2 Introducción

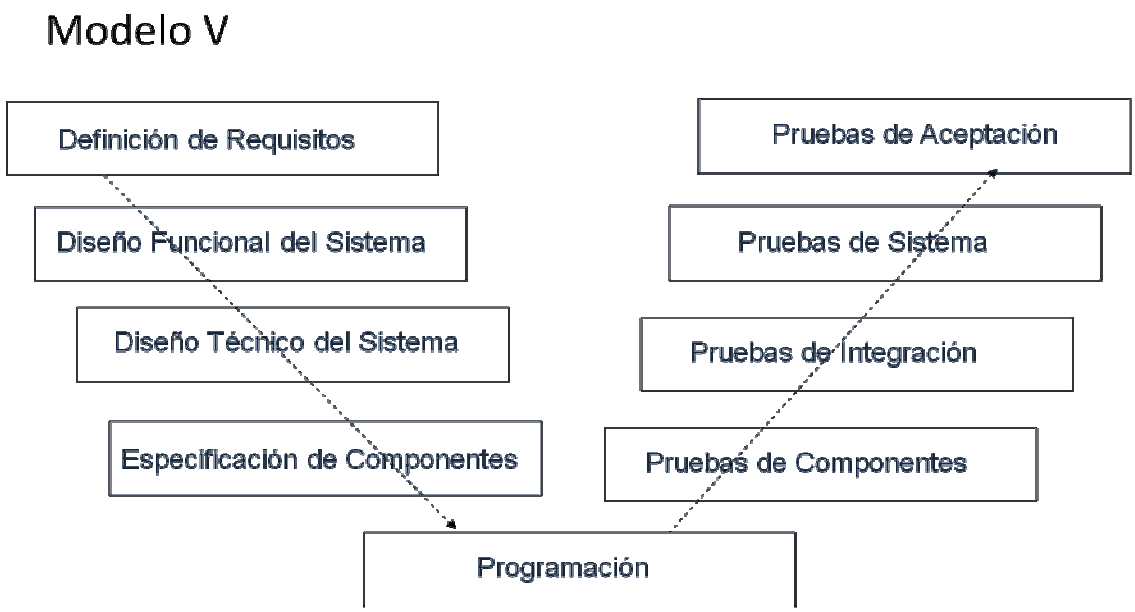
La prueba no existe en forma aislada; las actividades de prueba están relacionadas con las actividades de desarrollo de software. Los diferentes modelos de ciclo de vida de desarrollo necesitan diferentes enfoques hacia la prueba.

2.5.3 Modelo-V

Aunque existen variantes del Modelo-V, un tipo común de Modelo-V usa cuatro niveles de prueba, correspondientes a cuatro niveles de desarrollo.

Los cuatro niveles usados en este programa son:

- ✓ Prueba de componente (unidad).
- ✓ Prueba de integración.
- ✓ Prueba de sistema.
- ✓ Prueba de aceptación.



En la práctica, un Modelo-V puede tener más, pocos o diferentes niveles de desarrollo y prueba, dependiendo del proyecto y del producto de software. Por ejemplo, puede existir prueba de integración de componentes después de las pruebas de componente y prueba de integración del sistema después de la prueba del sistema.

Los productos de trabajo del software (tales como escenarios de negocio o casos de uso, especificaciones de requisito, documentos y código de diseño) producidos durante el desarrollo son a menudo la base de la prueba en uno o más niveles de prueba. Las referencias para los productos de trabajo genéricos incluyen el Modelo de Madurez de las Capacidades Integrado (CMMI) o "Procesos del ciclo de vida del software" (IEEE/IEC 12207). La verificación y validación (y diseño de prueba en forma temprana) pueden ser llevados a cabo durante el desarrollo de los productos de trabajo del software.

2.5.4 Modelos de desarrollo iterativo

El desarrollo iterativo es el proceso de establecer requisitos, diseñar, crear y probar un sistema, realizado como una serie de desarrollos más pequeños. Ejemplos son: la creación de prototipos, el desarrollo rápido de aplicaciones (RAD), el Proceso Unificado Racional (RUP) y los modelos de desarrollo ágiles. El incremento producido por una iteración puede ser probado en diversos niveles como parte de su desarrollo.
Un incremento, aumentado a otros desarrollados previamente, forma un sistema parcial creciente, el cual debería ser probado también. La prueba de regresión es progresivamente importante en todas las iteraciones después de la primera. La verificación y validación pueden ser llevadas a cabo en cada incremento.

2.5.5 Prueba dentro de un modelo de ciclo de vida

En cualquier modelo de ciclo de vida, existen varias características de buena prueba:

- ✓ Para cada actividad de desarrollo existe una actividad de prueba correspondiente.
- ✓ Cada nivel de prueba tiene objetivos de prueba específicos para ese nivel.
- ✓ El análisis y diseño de pruebas para un nivel de prueba dado deberían iniciar durante la actividad de desarrollo correspondiente.
- ✓ Los probadores deberían estar involucrados en la revisión de documentos tan pronto como los borradores estén disponibles en el ciclo de vida del desarrollo.

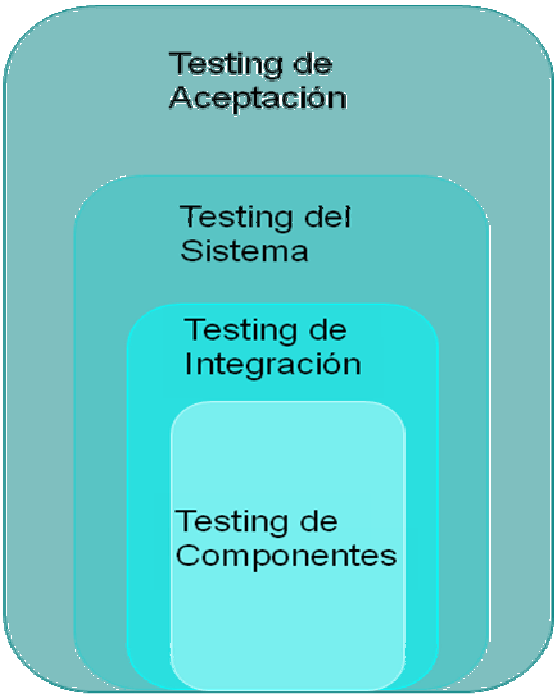
2.6 Niveles de prueba

2.6.1 Términos

Pruebas alfa, pruebas beta, pruebas de componentes (también conocidas como pruebas de unidad, módulo o de programa), pruebas de aceptación de contrato, controladores, pruebas de campo, requisitos funcionales, integración, pruebas de integración, requisitos no funcionales, pruebas operacionales (de aceptación), pruebas de aceptación de regulación, pruebas de robustez, agentes de sustitución, pruebas de sistema, desarrollo guiado por prueba, entorno de prueba, pruebas de aceptación de usuario.

2.6.2 Introducción

Para cada uno de los niveles de prueba, lo siguiente puede ser identificado: sus objetivos genéricos, el(los) producto(s) de trabajo siendo referenciados para derivar casos de prueba (es decir la base de prueba), el objeto de prueba (es decir lo que está siendo probado), defectos y fallas típicas a ser encontradas, requisitos de arnés de prueba y soporte de herramienta y los enfoques y responsabilidades específicos.



Los niveles de prueba pueden ser combinados o reorganizados dependiendo de la naturaleza del proyecto o de la arquitectura del sistema. Por ejemplo, para la integración de un producto de software comercial/listo para la venta (COTS) en un sistema, el comprador podrá realizar la prueba de integración en el nivel del sistema (por ejemplo, integración a la infraestructura o a otros sistemas o despliegue del sistema) y las pruebas de aceptación (funcional y/o no funcional y prueba operacional y/o del usuario).

2.6.3 Pruebas de componente

Las pruebas de componente buscan defectos en el, y verifican el funcionamiento del, software (por ejemplo, módulos, programas, objetos, clases, etc.) que son verificables separadamente. Pueden ser realizadas en forma aislada del resto del sistema, dependiendo del contexto del ciclo de vida de desarrollo y del sistema. Los agentes de sustitución, controladores y simuladores pueden ser usados.

Las pruebas de componente pueden incluir la prueba de funcionalidad y de las características no funcionales específicas, tales como comportamiento de recursos (por ejemplo, pérdidas de

memoria) o pruebas de robustez, así como pruebas estructurales (por ejemplo, cobertura de bifurcación). Los casos de prueba están derivados de los productos de trabajo tales como una especificación del componente, del diseño del software o del modelo de datos.

Típicamente, las pruebas de componente ocurren con acceso al código que está siendo probado y con el soporte del entorno de desarrollo, tales como un marco de trabajo de prueba de unidad o una herramienta de depuración, y, en la práctica, usualmente involucran al programador que escribió el código. Los defectos son típicamente arreglados tan pronto como son encontrados, sin registrar formalmente los incidentes.

Un enfoque en las pruebas de componente es preparar y automatizar los casos de prueba antes de la codificación. Esto se llama enfoque de primera prueba o desarrollo guiado por prueba. Este enfoque es altamente iterativo y está basado en ciclos de desarrollar casos de prueba, luego crear e integrar pequeñas piezas del código y ejecutar las pruebas de componente hasta que éstas pasen.

2.6.3.1 Definición

- ✓ Prueba de cada componente tras su realización/construcción.
- ✓ Los componentes son referidos como módulos, clases o unidades.
- ✓ También denominadas pruebas de Desarrollador (developer's test).

Dadas las convenciones de cada lenguaje de programación para la asignación de nombres a sus respectivos componentes, se podrá hacer referencia a una componente como

- ✓ Prueba de modulo: En C
- ✓ Prueba de Clase: En java o C++
- ✓ Pruebas de unidad: En pascal

2.6.3.2 Alcance

Solo se prueban componentes individuales.
Cada componente es probado de forma independiente.
Los casos de prueba tienen 3 fuentes.

- ✓ Un componente puede estar constituido por un conjunto de unidades más pequeñas.
- ✓ Los objetos de prueba no siempre pueden ser probados en solitario.
- ✓ Descubriendo fallos producidos por defectos internos.
- ✓ Los efectos cruzados entre componentes quedan fuera del alcance de estas pruebas

2.6.4 Pruebas de integración

Las pruebas de integración prueban interfaces entre componentes, interacciones a diferentes partes de un sistema, tales como el sistema operativo, sistema de archivos, hardware o interfaces entre sistemas.

Puede haber más de un nivel de pruebas de integración y éste puede ser llevado a cabo sobre objetos de prueba de diverso tamaño. Por ejemplo:

- ✓ Las pruebas de integración de componentes prueban las interacciones entre los componentes del software y son realizada después de las pruebas de componente.
- ✓ Las pruebas de integración de sistema prueban las interacciones entre los diferentes sistemas y pueden ser realizada después de las pruebas de sistema. En este caso, la organización desarrolladora puede controlar solamente un lado de la interfaz, así los cambios pueden estar desestabilizando. Los procesos de negocio implementados como flujos de trabajo pueden involucrar una serie de sistemas. Los problemas de multi-plataforma pueden ser importantes.

Cuan mayor es el alcance de integración, más difícil se vuelve el aislar las fallas a un componente o sistema específico, lo cual puede llevar a un riesgo incrementado.

Las estrategias de integración sistemáticas pueden estar basadas en la arquitectura del sistema (tales como arriba-abajo o abajo-arriba), tareas funcionales, secuencias de proceso de transacción o algún otro aspecto del sistema o componente. Para reducir el riesgo de descubrimiento de defecto tardío, la integración debería ser normalmente incremental en lugar del tipo “big bang”.

Las pruebas de características no funcionales específicas (por ejemplo, rendimiento) pueden ser incluidas en las pruebas de integración.

En cada fase de integración, los probadores se concentran solamente en la integración en sí. Por ejemplo, si están integrando el módulo A con el módulo B, están interesados en probar la comunicación entre los módulos, no en la funcionalidad de cualquier módulo. Ambos enfoques funcionales y estructurales pueden ser usados.

Idealmente, los probadores deben comprender la planificación de integración de influencia y arquitectura. Si las pruebas de integración son planeadas antes que los componentes o sistemas sean creados, ellos pueden ser creados en el orden requerido para una prueba más eficiente.

2.6.4.1 Definición

La integración es la actividad en la cual se combinan componentes software individual en subsistemas más amplios.
Cada componente ya ha sido probado en lo referente a su funcionalidad interna (prueba de componente).

- ✓ Comprueba la interacción entre elementos de software (componentes) tras la integración del sistema.
- ✓ La integración adicional/subsiguiente de subsistemas también es parte del proceso de integración del sistema.
- ✓ Comprueban las funciones externas.
- ✓ Pueden ser ejecutadas por desarrolladores, testers o ambos.

2.6.4.2 Estrategias

Tipos fundamentales de integración

- ✓ **Integración incremental**. Se combina el siguiente módulo que se debe probar con el conjunto de módulos que ya han sido probados Integración no incremental. Se prueba cada módulo por separado y luego se integran todos de una vez y se prueba el programa completo.

  - ✓ **Ascendente.** Se comienza por los módulos hoja.
    El proceso es el siguiente:

    - o Se combinan los módulos de bajo nivel en grupos que realicen una función o subfunción específica (o quizás si no es necesario, individualmente) -> de este modo reducimos el número de pasos de integración.
    - o Se escribe para cada grupo un módulo impulsor o conductor -> de este modo permitimos simular la llamada a los módulos, introducir datos de prueba y recoger resultados.
    - o Se prueba cada grupo mediante su impulsor.
    - o Se eliminan los módulos impulsores y se sustituyen por los módulos de nivel superior en la jerarquía.

  - ✓ **Descendente.** Se comienza por el módulo raíz.

  - ✓ Comienza por el módulo de control principal y va incorporando módulos subordinados progresivamente.
  - ✓ No hay un orden adecuado de integración, pero unos consejos son los siguientes:
    - o Si hay secciones críticas (especialmente complejas) se deben integrar lo antes posible.
    - o El orden de integración debe incorporar cuanto antes los módulos de entrada/salida para facilitar la ejecución de pruebas.

  - ✓ Existen dos formas básicas de hacer esta integración:

    - o Primero en profundidad: Se van completando ramas del árbol (A B E C F G D)
    - o Primero en anchura: Se van completando niveles de jerarquía (A B C D E F G)

ETAPAS FUNDAMENTALES DE LA INTEGRACION DESCENDENTE

- o El módulo raíz es el primero: Se escriben módulos ficticios que simulan los subordinados.
- o Una vez probado el módulo raíz (sin detectarse ya ningún defecto), se sustituye uno de los subordinados ficticios por el módulo correspondiente según el orden elegido.
- o Se ejecutan las correspondientes pruebas cada vez que se incorpora un módulo nuevo.
- o Al terminar cada prueba, se sustituye un ficticio por su correspondiente real.
- o Conviene repetir algunos casos de prueba de ejecuciones anteriores para asegurarse de que no se ha introducido ningún defecto nuevo.

- ✓ **Integración no incremental.** Se prueba cada módulo por separado y luego se integran todos de una vez y se prueba el programa completo.

COMPARACION DE LOS DISTINTOS TIPOS DE INTEGRACION PRUEBAS DEL SOFTWARE

- ✓ **Ventajas de la no incremental:**
  - o Requiere menos tiempo de máquina para las pruebas, ya que se prueba de una sola vez la combinación de los módulos.
  - o Existen más oportunidades de probar módulos en paralelo.

- ✓ **Ventajas de la incremental:**
  - o Requiere menos trabajo, ya que se escriben menos módulos impulsores y ficticios
  - o Los defectos y errores en las interfaces se detectan antes, ya que se empieza antes a probar las uniones entre los módulos.
  - o La depuración es mucho más fácil, ya que si se detectan los síntomas de un defecto en un paso de la integración hay que atribuirlo muy probablemente al último módulo incorporado.
  - o Se examina con mayor detalle el programa, al ir comprobando cada interfaz poco a poco.

VENTAJAS Y DESVENTAJAS DE LAS INTEGRACIONES ASCENDENTE Y DESCENDENTE

Ascendente	**Descendente**
Ventajas	
- Es un método ventajoso si aparecen Grandes fallos en la parte inferior del programa, ya que se prueba antes. - La entradas para las pruebas son más módulos inferiores tienen funciones. - Es más fácil observar los resultados de la prueba, ya que es en los módulos inferiores donde se elaboran los datos (los superiores suelen ser módulos de control).	- Es ventajosa si aparecen grandes defectos en los niveles superiores del programa, ya que se prueban antes. - Una vez incorporadas las funciones de entrada/salida, es fácil manejar los casos de prueba. - Permite ver antes una estructura previa del programa, lo que facilita el hacer demostraciones y ayuda a mantener la moral.
Desventajas	**Desventajas**
Se requieren módulos impulsores, que deben codificarse. • El programa, como entidad, sólo aparece cuando se agrega el último módulo.	- Se requieren módulos ficticios que suelen ser complejos de crear. - Antes de incorporar la entrada/salida resulta complicado el manejo de los casos de prueba. - Las entradas para las pruebas pueden ser difíciles o imposibles de crear, puesto que, a menudo, se carece de los módulos inferiores que proporcionan los detalles de operación. - Es más difícil observar la salida, ya

	que los resultados surgen de los módulos inferiores. • **Pueden inducir a diferir la terminación de la prueba de ciertos módulos.**

- ✓ **Ad-hoc:** Los componentes serán probados, si fuera posible, inmediatamente después de haber sido finalizada su construcción y se hayan finalizado las pruebas de componente.

2.6.4.3 Consideración al momento de la elección

La estrategia de integración determina la magnitud del esfuerzo requerido para las pruebas
La finalización de la construcción de componentes determina, para todos los tipos de estrategias, el intervalo temporal en el cual el componente estará disponible.
La estrategia de desarrollo influye en la estrategia de integración.

2.6.5 Pruebas de sistema

Las pruebas de sistema están relacionadas con el comportamiento de un sistema/producto completo como está definido por el alcance de un proyecto o programa de desarrollo.

En las pruebas de sistema, el entorno de prueba debe corresponder al objetivo o entorno de producción final tanto como sea posible para minimizar el riesgo de fallas de entorno específico que no están siendo encontradas en la prueba.

Las pruebas de sistema pueden incluir pruebas basadas en riesgos y/o en especificaciones de requisitos, procesos de negocio, casos de uso, u otras descripciones de alto nivel del comportamiento del sistema, interacciones con el sistema operativo y recursos del sistema.

Las pruebas de sistema deben investigar tanto los requisitos funcionales como los requisitos no funcionales del sistema. Los requisitos pueden existir como texto y/o modelos. Los probadores también necesitan tratar con requisitos incompletos o no documentados. Las pruebas de sistema de requisitos funcionales comienzan al usar las técnicas más apropiadas basadas en especificación (caja negra) para el aspecto del sistema a ser probado. Por ejemplo, una tabla de decisión puede ser creada para combinaciones o efectos descritos en reglas de negocio. Las técnicas basadas en estructura (caja blanca) pueden ser usadas para evaluar la minuciosidad de la prueba con respecto al elemento estructural, tales como estructura del menú o navegación de página web.

Un equipo de prueba independiente a menudo lleva a cabo pruebas de sistema.

2.6.5.1 Definición

Tienen que ver con el comportamiento de todo un sistema/producto, definida por el alcance de un proyecto de desarrollo o programa.

Los casos de prueba pueden ser obtenidos de:

- ✓ Evaluación de Riesgos
- ✓ Especificaciones funcionales
- ✓ Casos de Uso
- ✓ Procesos de Negocios

2.6.5.2 Enfoque para las pruebas funcionales

Basadas en Requisitos

Los casos de prueba se derivan de la especificación de requisitos.
El número de casos de prueba variará en función del tipo/profundidad de las pruebas basadas en la especificación de requisitos.

Basadas en procesos de Negocio

Cada proceso de negocio sirve como fuente para derivar/generar pruebas.
El orden de relevancia de los procesos de negocio puede ser aplicado para asignar prioridades a los casos de prueba.

Basadas en Casos de Uso

Los casos de prueba se derivan/generan a partir de las secuencias de usos esperados o razonables
Las secuencias utilizadas con mayor frecuencia reciben una prioridad más alta.

2.6.6 Pruebas de aceptación

Las pruebas de aceptación son a menudo la responsabilidad de los clientes o usuarios de un sistema; otras partes interesadas pueden estar involucradas también.

La meta en las pruebas de aceptación es establecer la confianza en el sistema, las partes del sistema o las características no funcionales específicas del sistema. Encontrar defectos no es el foco principal en las pruebas de aceptación. Las pruebas de aceptación pueden evaluar la predisposición del sistema para el despliegue y uso, aunque no es necesariamente el nivel final de la prueba. Por ejemplo, una prueba de integración del sistema a gran escala puede venir después de las pruebas de aceptación para un sistema.

Las pruebas de aceptación pueden ocurrir como más que apenas un simple nivel de prueba, por ejemplo:

- ✓ Un producto de software COTS puede ser probado para aceptación cuando es instalado o integrado.
- ✓ Las pruebas de aceptación de la usabilidad de un componente pueden ser realizadas durante las pruebas de componente.
- ✓ Las pruebas de aceptación de una nueva mejora funcional pueden venir antes de las pruebas de sistema.

Las formas típicas de las pruebas de aceptación incluyen lo siguiente:

2.6.6.1 Pruebas de aceptación del usuario

Verifica típicamente la aptitud para el uso del sistema por los usuarios del negocio.

2.6.6.2 Pruebas operacionales (de aceptación)

La aceptación del sistema por los administradores de sistema, incluyendo:

- ✓ Pruebas de respaldo/restauración;
- ✓ Recuperación de desastre;
- ✓ Gestión del usuario;
- ✓ Tareas de mantenimiento;
- ✓ Chequeos periódicos de las vulnerabilidades de seguridad.

2.6.6.3 Pruebas de aceptación de contrato y de regulación

Las pruebas de aceptación de contrato son realizadas contra unos criterios de aceptación del contrato para producir software desarrollado a la medida. Los criterios de aceptación deben ser definidos cuando el contrato es acordado. Las pruebas de aceptación de regulación son realizadas contra cualquier regulación que debe ser adherida, tales como regulaciones gubernamentales, legales o de seguridad.

2.6.6.4 Pruebas alfa y beta (o de campo)

Los desarrolladores de software de mercado, o de COTS, a menudo quieren obtener información de clientes existentes o potenciales en su mercado antes que el producto de software sea puesto a la venta comercialmente. La prueba alfa es realizada en el sitio de la organización desarrolladora. La prueba beta, o prueba de campo, es realizada por la gente en sus propios lugares. Ambas son realizadas por clientes potenciales, no por desarrolladores del producto.

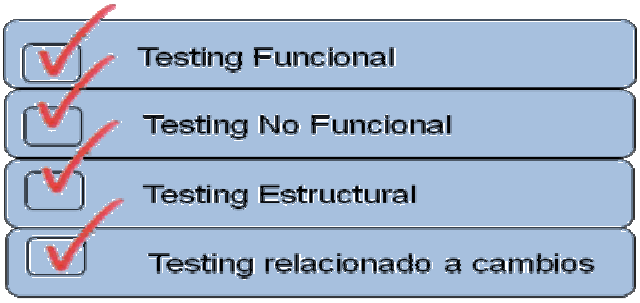
Las organizaciones pueden utilizar otros términos también, tales como pruebas de aceptación de fábrica y pruebas de aceptación del sitio para los sistemas que son probados antes y después de ser movidos al sitio del cliente.

2.7 Tipos de pruebas: los objetivos de la prueba

2.7.1 Términos

Automatización, pruebas de caja negra, cobertura de código, pruebas de confirmación, pruebas funcionales, pruebas de interoperabilidad, pruebas de carga, pruebas de mantenibilidad, pruebas de rendimiento, pruebas de portabilidad, pruebas de regresión, pruebas de fiabilidad, pruebas de

seguridad, pruebas basadas en especificación, pruebas de estrés, pruebas estructurales, conjunto de pruebas, pruebas de usabilidad, pruebas de caja blanca.



2.7.2 Introducción

Un grupo de actividades de prueba puede estar dirigido a la verificación del sistema del software (o a una parte de un sistema) basado en una razón u objetivo específico para la prueba.

Un tipo de la prueba se centra en un objetivo particular de prueba, el cual podría ser la prueba de una función a ser realizada por el software; una característica de calidad no funcional, tales como fiabilidad o usabilidad, la estructura o arquitectura del software o del sistema; o relacionado con los cambios, es decir confirmar que los defectos han sido arreglados (pruebas de confirmación) y buscar cambios involuntarios (pruebas de regresión).

Un modelo del software puede ser desarrollado y/o usado en pruebas estructurales y funcionales. Por ejemplo, en pruebas funcionales un modelo de flujo de proceso, un modelo de transición de estado o una especificación de lenguaje simple; y para pruebas estructurales un modelo de flujo de control o un modelo de estructura de menú.

2.7.3 Prueba de función (prueba funcional)

Las funciones, que un sistema, un subsistema o un componente realizará, pueden estar descritas en productos de trabajo tales como una especificación de requisitos, casos de uso o una especificación funcional, o podrían estar no documentados. Las funciones son “lo que” el sistema hace.

Las pruebas funcionales están basadas en estas funciones y rasgos (descritos en documentos o comprendidos por los probadores), y pueden ser realizadas en todos los niveles de prueba (por ejemplo, las pruebas para componentes pueden estar basadas en una especificación de componente).

Las técnicas basadas en especificación pueden ser usadas para derivar condiciones de prueba y casos de prueba de la funcionalidad del software o del sistema. Las pruebas funcionales consideran el comportamiento externo del software (pruebas de caja negra).

Un tipo de pruebas funcionales, pruebas de seguridad, investiga las funciones (por ejemplo, un firewall) relacionadas con la detección de amenazas, tales como virases, de desconocidos maliciosos.

2.7.4 Prueba de características del producto de software (prueba no funcional)

Las pruebas no funcionales incluyen, mas no están limitadas a, pruebas de rendimiento, pruebas de carga, pruebas de estrés, pruebas de usabilidad, pruebas de interoperabilidad, pruebas de mantenibilidad, pruebas de fiabilidad y pruebas de portabilidad. Es la prueba de "como" funciona el sistema.

Las pruebas no funcionales pueden ser realizadas en todos los niveles de prueba. El término pruebas no funcionales describe las pruebas requeridas para medir las características de sistemas y software que pueden ser cuantificadas en una escala diversa, tales como tiempos de respuesta para pruebas de rendimiento. Estas pruebas pueden ser referenciadas a un modelo de calidad tales como aquel definido en "Ingeniería de Software – Calidad del Producto de Software" (ISO 9126).

2.7.5 Prueba de estructura/arquitectura de software (prueba estructural)

Las pruebas estructurales (de caja blanca) pueden ser realizadas en todos los niveles de prueba. Las técnicas estructurales son usadas mejor después de las técnicas basadas en especificación, para ayudar a medir la minuciosidad de las pruebas a través de la evaluación de cobertura de un tipo de estructura.

La cobertura está en la medida en que una estructura ha sido ejercida por un conjunto de pruebas, expresada como un porcentaje de elementos que están siendo cubiertos. Si la cobertura no es del 100%, entonces más pruebas podrán ser diseñadas para probar aquellos elementos que faltaron, y por lo tanto, incrementar la cobertura.

En todos los niveles de prueba, mas especialmente en las pruebas de componente y en las pruebas de integración de componente, herramientas pueden ser usadas para medir la cobertura de código de los elementos, tales como sentencias o decisiones. Las pruebas estructurales pueden estar basadas en la arquitectura del sistema, tales como una jerarquía de llamada.
Los enfoques de pruebas estructurales pueden ser también aplicados en niveles de sistema, de integración de sistema o de pruebas de aceptación (por ejemplo, a modelos de negocio o estructuras de menú).

2.7.6 Pruebas relacionadas con cambios (pruebas de confirmación y pruebas de regresión)

Cuando un defecto es detectado y arreglado, entonces el software debe ser probado de nuevo para confirmar que el defecto original ha sido retirado exitosamente. Esto se llama prueba de confirmación. La depuración (arreglo de defectos) es una actividad de desarrollo, no una actividad de pruebas.

Las pruebas de regresión son las pruebas repetidas de un programa ya probado, después de la modificación, para descubrir cualquier defecto introducido o no descubierto como un resultado de el(los) cambio(s). Estos defectos pueden estar ya sea en el software que está siendo probado o en cualquier componente del software relacionado o no relacionado. Son realizadas cuando el software, o su entorno, es cambiado.
La medida de las pruebas de regresión está basada en el riesgo de no encontrar defectos en el software que estaba funcionando previamente.

Las pruebas deben ser repetibles si serán usadas para las pruebas de confirmación y para asistir a las pruebas de regresión.

Las pruebas de regresión pueden ser realizadas en todos los niveles de prueba, y se aplican a las pruebas funcionales, no funcionales y estructurales. Los conjuntos de pruebas de regresión son corridos muchas veces y generalmente evolucionan lentamente, así las pruebas de regresión son un fuerte candidato para la automatización.

Pruebas típicas:

- Pruebas de confirmación: repetición de pruebas, después de la corrección de errores
- Pruebas de regresión: pruebas para descubrir defectos introducidos en funcionalidad previamente sin fallos

Criterio de selección:

- Casos de prueba de prioridad alta
- Probar solamente la funcionalidad estándar, saltarse casos y variaciones especiales
- Probar solamente la configuración utilizada con mayor frecuencia.
- Probar solamente subsistemas/Zonas seleccionadas del objeto de pruebas

Si durante fases tempranas del proyecto resulta evidente que ciertas pruebas son adecuadas para las pruebas de regresión, se deberá considerar la automatización de estas pruebas.

2.8 Pruebas de mantenimiento

2.8.1 Términos

Análisis de impacto, pruebas de mantenimiento, migración, modificaciones, retiro.

2.8.2 Introducción

Una vez desplegado, un sistema de software a menudo está en servicio durante años o décadas. Durante este tiempo el sistema y su entorno son a menudo corregidos, cambiados o extendidos. Las pruebas de mantenimiento son realizadas en un sistema operacional existente, y son activadas por modificaciones, migración o retiro del software o del sistema.

Las modificaciones incluyen cambios de mejora planeados (por ejemplo, basados en lanzamiento), cambios correctivos y de emergencia, y cambios de entorno, tales como actualizaciones planeadas del sistema operativo o de la base de datos, o parches a vulnerabilidades recientemente expuestas o descubiertas en el sistema operativo.

Las pruebas de mantenimiento para migración (por ejemplo, de una plataforma a otra) deben incluir pruebas operacionales del nuevo entorno, así como del software cambiado.

Las pruebas de mantenimiento para el retiro de un sistema pueden incluir las pruebas de migración de datos o de almacenamiento si periodos de largos de retención de datos son requeridos.

Además de probar que ha sido cambiado, las pruebas de mantenimiento incluyen pruebas de regresión extensivas a las partes del sistema que no han sido cambiadas. El alcance de las pruebas de mantenimiento está relacionado con el riesgo del cambio, del tamaño del sistema existente y con el tamaño del cambio. Dependiendo de los cambios, las pruebas de mantenimiento pueden ser realizadas en cualquiera o en todos los niveles de prueba y para cualquier o para todos los tipos de prueba.

Determinar como el sistema existente puede ser afectado por los cambios se llama análisis de impacto y es usado para ayudar a decidir cuantas pruebas de regresión se harán.

Referencias

2.1.3 CMMI, Craig, 2002, Hetzel, 1998, IEEE 12207
2.2 Hetzel, 1998
2.2.4 Copeland, 2004, Myers, 1979
2.3.1 Beizer, 1990, Black, 2001, Copeland, 2004
2.3.2 Black, 2001, ISO 9126
2.3.3 Beizer, 1990, Copeland, 2004, Hetzel, 1998
2.3.4 Hetzel, 1998, IEEE 829
2.4 Black, 2001, Craig, 2002, Hetzel, 1998, IEEE 829

3 Técnicas estáticas

Objetivos de aprendizaje para las técnicas estáticas

Los objetivos identifican lo que podrá hacer después de la finalización de cada módulo.

3.1 Revisiones y el proceso de prueba

- ✓ Reconocer los productos software que pueden ser examinados por las diferentes técnicas estáticas.
- ✓ Describir la importancia y el valor de considerar técnicas estáticas para la evaluación de los productos software.
- ✓ Explicar la diferencia entre técnicas estáticas y dinámicas.

3.2 Proceso de revisión

- ✓ Recordar las fases, roles y responsabilidades de una revisión formal típica.
- ✓ Explicar las diferencias entre los diferentes tipos de revisión: revisión informal, revisión técnica, revisión guiada e inspección.
- ✓ Explicar los factores para el funcionamiento exitoso de las revisiones.

3.3 Análisis estático por herramientas

- ✓ Describir el objetivo del análisis estático y comprarlo con las pruebas dinámicas.
- ✓ Recordar los defectos y errores típicos identificados por el análisis estático y compararlos con las revisiones y las pruebas dinámicas.
- ✓ Listar los beneficios típicos del análisis estático.
- ✓ Listar el código y los defectos de diseño típicos que pueden ser identificados por las herramientas de análisis estático.

3.4 Técnicas estáticas y el proceso de prueba

3.4.1 Términos

Pruebas dinámicas, revisiones, análisis estático.

3.4.2 Introducción

Las técnicas de pruebas estáticas no ejecutan el software que está siendo probado; son manuales (revisiones) o automatizados (análisis estático).

Las revisiones son una forma de probar productos software (incluyendo el código) y pueden ser realizadas bien antes de la ejecución de prueba dinámica. Los defectos detectados durante las revisiones temprano en el ciclo de vida son a menudo más baratos para retirar que aquellos detectados mientras se corren las pruebas (por ejemplo, defectos encontrados en los requisitos).

Una revisión podría ser realizada enteramente como una actividad manual, pero también existe el soporte de herramienta. La actividad manual principal es el examinar un producto de trabajo y realizar comentarios sobre éste. Cualquier producto software puede ser revisado, incluyendo las especificaciones de requisito, especificaciones de diseño, código, planos de prueba, especificaciones de prueba, casos de prueba, scripts de prueba, guías de usuario o páginas web.

Los beneficios de las revisiones incluyen corrección y detección temprana de defecto, mejoras de la productividad de desarrollo, escalas de tiempo de desarrollo reducidas, costo y tiempo reducidos de pruebas, reducciones del costo de vida útil, menos defectos y comunicación mejorada. Las revisiones pueden encontrar omisiones, por ejemplo, en los requisitos, las cuales son improbables de ser encontrados en las pruebas dinámicas.

Las revisiones, el análisis estático y las pruebas dinámicas tienen el mismo objetivo – identificar defectos. Son complementarios; las diferentes técnicas pueden encontrar diferentes tipos de defecto efectiva y eficientemente. En contraste con las pruebas dinámicas, las revisiones encuentran defectos en lugar de fallas.

Los defectos típicos que son más fáciles de encontrar en las revisiones que en las pruebas dinámicas son: desviaciones de los estándares, defectos de requisito, defectos de diseño, mantenibilidad insuficiente y especificaciones de interface incorrectos.

3.5 Proceso de revisión

3.5.1 Términos

Criterios de entrada, criterios de salida, revisión formal, revisión informal, inspección, inicio, métricas, líder de moderador/inspección, revisión de par, revisor, reunión de revisión, proceso de revisión, escriba, revisión técnica, revisión guiada.

3.5.2 Introducción

Las revisiones varían desde muy informal hasta muy formal (es decir bien estructurados y regulados). La formalidad de un proceso de revisión está relacionada con los factores tales como la madurez del proceso de desarrollo, cualquier requisito regulador o legal o la necesidad de un rastro de auditoría.

La forma en que una revisión es llevada a cabo depende del objetivo acordado de la revisión (por ejemplo, encontrar defectos, ganar entendimiento, o discusión y decisión por consenso).

3.5.3 Fases de una revisión formal

Una revisión formal típica tiene las siguientes fases principales:

- ✓ Planificación: seleccionar el personal, asignar roles; definir los criterios de entrada y salida para más tipos de revisión formal (por ejemplo, inspección); y seleccionar que partes de los documentos ser verán.
- ✓ Inicio: distribuir documentos; explicar los objetivos, procesos y documentos a los participantes; y comprobar los criterios de entrada (para más tipos de revisión formal).
- ✓ Preparación individual: el trabajo realizado por cada uno de los participantes por su cuenta antes de la reunión de revisión, notar los defectos potenciales, preguntas y comentarios.
- ✓ Reunión de revisión: discusión o registro, con resultados o minutas documentadas (para más tipos de revisión formal). Los participantes de la reunión pueden simplemente notar defectos, realizar recomendaciones para manejar los defectos o realizar decisiones sobre los defectos.
- ✓ Corrección del trabajo: arreglar los defectos encontrados, realizado típicamente por el autor.
- ✓ Seguimiento: comprobar que los defectos han sido tratados, reunir métricas y comprobar los criterios de salida (para más tipos de revisión formal).

3.5.4 Roles y responsabilidades

Una revisión formal típica incluirá los roles abajo:

- ✓ Gerente: decide sobre la ejecución de revisiones, asigna tiempo en los programas del proyecto y determina si se han cumplido los objetivos de la revisión.
- ✓ Moderador: la persona que lidera la revisión del documento o del conjunto de documentos, incluyendo planear la revisión, dirigir la reunión y hacer seguimiento después de la reunión. De ser necesario, el moderador puede mediar entre los varios puntos de vista y es a menudo la persona sobre quien recae el éxito de la revisión.
- ✓ Autor: el escritor o persona con principal responsabilidad del (de los) documento(s) a ser revisado(s).
- ✓ Revisores: individuos con un historial técnico o de negocio específico (también llamados chequeadores o inspectores) quienes, después de la preparación necesaria, identifican y describen los descubrimientos (ej. defectos) el producto bajo revisión. Los revisores deben ser elegidos para representar diferentes perspectivas y roles en el proceso de revisión, y ellos participan en cualquier reunión de revisión.
- ✓ Escriba (o redactor): documenta todos los asuntos, problemas y puntos abiertos que fueron identificados durante la reunión.

Mirar documentos desde diferentes perspectivas y usar listas de comprobación pueden hacer que las revisiones sean más eficaces y eficientes, por ejemplo, una lista de comprobación basada en perspectivas tales como usuario, mantenedor, probador u operaciones, o una lista de comprobación de los típicos problemas de los requisitos.

3.5.5 Tipos de revisión

Un simple documento puede ser el tema de más de una revisión. Si más de un tipo de revisión es usado, el orden puede variar. Por ejemplo, una revisión informal puede ser llevada a cabo antes de una revisión técnica, o una inspección puede ser llevada a cabo en una especificación de requisito antes de una revisión guiada con los clientes. Las principales características, opciones y propósitos de los tipos de revisión común son:

3.5.5.1 Revisión informal

Características clave:

- ✓ Ningún proceso formal;
- ✓ Puede haber programación de par o un líder técnico revisando los diseños y el código;
- ✓ Puede ser documentada opcionalmente;
- ✓ Puede variar en utilidad dependiendo del revisor;
- ✓ Propósito principal: manera barata de conseguir algo de beneficio.

3.5.5.2 Revisión guiada

Características clave:

- ✓ Reunión liderada por el autor;
- ✓ Escenarios, ensayos generales, grupo de par;
- ✓ Sesiones abiertas;
- ✓ Opcionalmente una preparación de pre-reunión de revisores, informe de revisión, lista de descubrimientos y escriba (quién no es el autor);

- ✓ Puede variar en la práctica desde absolutamente informal hasta muy formal;
- ✓ Propósitos principales: aprender, ganar entendimiento, encontrar defectos.

3.5.5.3 Revisión técnica

Características clave:

- ✓ Proceso de detección de defecto documentado y definido que incluye pares y expertos técnicos;
- ✓ Puede ser realizada como una revisión de par sin participación de la gestión;
- ✓ Idealmente conducida por un moderador entrenado (no el autor);
- ✓ Preparación de pre-reunión;
- ✓ Opcionalmente el uso de listas de comprobación, informe de revisión, lista de descubrimientos y participación de la gestión;
- ✓ Puede variar en la práctica desde absolutamente informal hasta muy formal;
- ✓ Propósitos principales: discutir, tomar decisiones, evaluar alternativas, encontrar defectos, solucionar problemas técnicos y comprobar la conformidad con las especificaciones y los estándares.

3.5.5.4 Inspección

Características clave:

- ✓ Liderada por un moderador entrenado (no el autor);
- ✓ Usualmente exanimación de par;
- ✓ Roles definidos;
- ✓ Incluye métricas;
- ✓ Proceso formal basado en reglas y listas de comprobación con criterios de entrada y de salida;
- ✓ Preparación de pre-reunión;
- ✓ Informe de la inspección, lista de descubrimientos;
- ✓ Proceso formal de seguimiento;
- ✓ Opcionalmente, mejora de proceso y lector;
- ✓ Propósito principal: encontrar defectos.

3.5.6 Factores de éxito para revisiones

Los factores de éxito para revisiones incluyen:

- ✓ Cada revisión tiene un objetivo predefinido claro.
- ✓ La gente adecuada para los objetivos de revisión está involucrada.
- ✓ Los defectos encontrados son bienvenidos y expresados objetivamente.
- ✓ Los asuntos de la gente y los aspectos sicológicos son tratados (por ejemplo, convertirlos en una experiencia positiva para el autor).

- ✓ Las técnicas de revisión aplicadas son convenientes para el tipo y nivel de los productos software y de los revisores.
- ✓ Las listas de comprobación o roles son usados si son apropiados para incrementar la efectividad de la identificación del defecto.
- ✓ El entrenamiento es dado en técnicas de revisión, especialmente las técnicas más formales, tales como la inspección.
- ✓ La gestión apoya un buen proceso de revisión (ej. incorporando el tiempo adecuado para las actividades de revisión en los programas del proyecto).
- ✓ Existe un énfasis en el aprendizaje y en la mejora del proceso.

3.6 Análisis estático por herramientas

3.6.1 Términos

Compilador, complejidad, flujo de control, flujo de datos, análisis estático.

3.6.2 Introducción

El objetivo del análisis estático es encontrar defectos en el código fuente y en los modelos del software. El análisis estático se realiza sin realmente ejecutar el software que está siendo examinado por la herramienta; las pruebas dinámicas si ejecutan el código del software. El análisis estático puede localizar los defectos que son difíciles de encontrar en las pruebas. Como con las revisiones, el análisis estático encuentra defectos en lugar de fallas. Las herramientas de análisis estático analizan el código del programa (ej. el flujo de control y el flujo de datos), así como la salida generada tales como HTML y XML.

El valor del análisis estático es:

- ✓ Detección temprana de defectos antes de la ejecución de la prueba.
- ✓ Detección temprana sobre los aspectos sospechosos del código o del diseño, mediante el cálculo de las métricas, tales como una alta medida de complejidad.
- ✓ Identificación de defectos difícilmente encontrados por las pruebas dinámicas.
- ✓ Detectar dependencias e inconsistencias en los modelos del software, tales como vínculos.
- ✓ Mantenibilidad mejorada del código y del diseño.
- ✓ Prevención de defectos, si las lecciones son aprendidas en el desarrollo.

Los defectos típicos descubiertos por las herramientas de análisis estático incluyen:

- ✓ Hacer referencia a una variable con un valor indefinido;
- ✓ Interface inconsistente de entre los módulos y los componentes;
- ✓ Variables de que nunca se utilizan;
- ✓ Código (muerto) inalcanzable;
- ✓ Violaciones de estándares de programación;
- ✓ Vulnerabilidades de seguridad;
- ✓ Violaciones de sintaxis del código y de los modelos del software.

Las herramientas de análisis estático son utilizadas típicamente por los desarrolladores (que comprueban con reglas predefinidas o estándares de programación) antes y durante las pruebas de componente y de integración, y por los diseñadores durante la modelación del software. Las herramientas de análisis estático pueden producir un gran número de mensajes de advertencia, los cuales necesitan ser bien administrados para permitir el uso más efectivo de la herramienta.

Los compiladores pueden ofrecer algo de soporte para el análisis estático, incluyendo el cálculo de las métricas.

Referencias

3.2 IEEE 1028
3.2.2 Gilb, 1993, van Veenendaal, 2004
3.2.4 Gilb, 1993, IEEE 1028
3.3 Van Veenendaal, 2004

4 Técnicas de diseño de pruebas

Objetivos de aprendizaje para las técnicas de diseño de pruebas

4.1 Identificando las condiciones de prueba y diseñando los casos de prueba

- ✓ Diferenciar entre una especificación del diseño de prueba, una especificación de casos de prueba y una especificación de procedimientos de prueba.
- ✓ Comparar los términos de condición de prueba, caso de prueba y procedimiento de prueba.
- ✓ Escribir casos de prueba:
  - o Mostrando una clara trazabilidad a los requisitos;
  - o Conteniendo un resultado esperado.

- ✓ Traducir casos de prueba hacia una especificación de procedimientos de prueba bien estructurada en un nivel de detalle relevante al conocimiento de los probadores.
- ✓ Escribir un programa de ejecución de prueba para un conjunto dado de casos de prueba, considerando priorización, y dependencias lógicas y técnicas.

4.2 Categorías de técnicas de diseño de prueba

- ✓ Recordar las razones que tanto los enfoques basados en especificación (caja negra) y los basados en estructura (caja blanca) para el diseño de casos de prueba son útiles, y listar las técnicas comunes para cada uno.
- ✓ Explicar las características y las diferencias entre las pruebas basadas en especificación, las pruebas basadas en estructura y entre las pruebas basadas en experiencia.

4.3 Técnicas basadas en especificación o de caja negra

- ✓ Escribir los casos de prueba de modelos de software dados usando las siguientes técnicas de diseño de prueba:
  - o Particiones de equivalencia;
  - o Análisis del valor límite;
  - o Tablas de decisión;
  - o Diagramas de transición de estado.
- ✓ Comprender el propósito principal de cada una de las cuatro técnicas, que nivel y que tipo de pruebas podrían usar la técnica y como la cobertura puede ser medida.
- ✓ Comprender el concepto de pruebas de caso de uso y sus beneficios.

4.4 Técnicas basadas en estructura o de caja blanca

- ✓ Describir el concepto y la importancia de la cobertura de código.
- ✓ Explicar los conceptos de cobertura de sentencias y de decisión, y comprender que estos conceptos también pueden ser usados en otros niveles de prueba que en las pruebas de componente (por ejemplo, procedimientos de negocio a nivel de sistema).
- ✓ Escribir casos de prueba de flujos de control dados usando las siguientes técnicas de diseño de prueba:
  - o Pruebas de sentencias;
  - o Pruebas de decisión.
- ✓ Evaluar la cobertura de sentencias y de decisión para compleción.

4.5 Técnicas basadas en experiencia

- ✓ Recordar las razones para escribir casos de prueba basados la intuición, experiencia y conocimiento sobre defectos comunes.
- ✓ Comparar las técnicas basadas en experiencia con las técnicas de pruebas basadas en especificación.

4.6 Eligiendo las técnicas de prueba

- ✓ Listar los factores que influyen la selección de una técnica apropiada de diseño de prueba para una clase de problema en particular, tales como un tipo de sistema, riesgo, requisitos del cliente, modelos para modelación de caso de uso, modelos de requisito o conocimiento del probador.

4.7 Identificando las condiciones de prueba y diseñando los casos de prueba

4.7.1 Términos

Casos de prueba, especificación de casos de prueba, condición de prueba, datos de prueba, especificación de procedimientos de prueba, script de prueba, trazabilidad.

4.7.2 Introducción

El proceso de identificar las condiciones de prueba y de diseñar pruebas consiste en un número de pasos:

- ✓ Diseñar las pruebas identificando las condiciones de prueba.
- ✓ Especificar los casos de prueba.
- ✓ Especificar los procedimientos de prueba.

El proceso puede ser realizado en diferentes maneras, desde muy informal con poca o nada de documentación, hasta muy formal (como está descrito en esta sección). El nivel de formalidad depende del contexto de la prueba, incluyendo la organización, la madurez de la prueba y los procesos de desarrollo, restricciones de tiempo y de la gente involucrada.

Durante el diseño de prueba, la documentación base de pruebas es analizada para determinar que probar, es decir, para identificar las condiciones de prueba. Una condición de prueba está definida como un elemento o evento que podría ser verificado por uno o más casos de prueba (por ejemplo, una función, transacción, característica de calidad o un elemento estructural).

Establecer la trazabilidad desde las condiciones de prueba de vuelta hasta las especificaciones y requisitos permite tanto el análisis de impacto, cuando los requisitos cambian, como la cobertura de requisitos a ser determinada por un conjunto de pruebas. Durante el diseño de prueba el enfoque de prueba detallado es implementado basado en, entre otras consideraciones, los riesgos identificados (ver capítulo 5 para más información sobre análisis de riesgo).

Durante la especificación de casos de prueba, los casos de prueba y los datos de prueba son desarrollados y descritos en detalle al usar las técnicas de diseño de prueba. Un caso de prueba consiste en un conjunto de valores de entrada, precondiciones de ejecución, resultados esperados y post-condiciones de ejecución, desarrollados para cubrir cierta(s) condición(es) de prueba. El "Estándar para la Documentación de Prueba de Software" (IEEE 829) describe el contenido de las especificaciones del diseño de prueba.

Los resultados esperados deben ser producidos como parte de la especificación de un caso de prueba y deben incluir entradas, cambios a los datos y estados y cualquier otra consecuencia de la prueba. Si los resultados esperados no han sido definidos entonces un resultado plausible, pero erróneo, puede ser interpretado como el correcto. Los resultados esperados deben ser idealmente definidos previos a la ejecución de prueba.

Los casos de prueba son colocados en un orden ejecutable; esto es la especificación de procedimientos de prueba. El procedimiento de prueba (o script de prueba manual) especifica la secuencia de acción para la ejecución de una prueba. Si las pruebas son ejecutadas usando una herramienta de ejecución de prueba, la secuencia de acciones está especificada en un script de prueba (el cual es un procedimiento de prueba automatizado).

Los diversos procedimientos de prueba y los scripts de prueba automatizados son formados subsecuentemente en una agenda de ejecución de pruebas que define el orden en el que los diversos procedimientos de prueba, y posiblemente los scripts de prueba automatizados, son ejecutados, cuando serán llevados a cabo y por quien. La agenda de ejecución de pruebas tomará en cuenta tales factores como pruebas de regresión, priorización y dependencias técnicas y lógicas.

4.8 Categorías de técnicas de diseño de prueba

4.8.1 Términos

Técnicas de caja negra, técnicas basadas en experiencia, técnicas basadas en especificación, técnicas basadas en estructura, técnicas de caja blanca.

4.8.2 Introducción

El propósito de una técnica de diseño de prueba es identificar las condiciones de prueba y los casos de prueba.

Es una distinción clásica el denotar técnicas de prueba como caja negra o caja blanca. Las técnicas de caja negra (también llamadas técnicas basadas en especificación) son una forma de derivar y seleccionar las condiciones de prueba o los casos de prueba basadas en un análisis de la documentación base de pruebas, ya sea funcional o no funcional, para un componente o sistema sin referencia a su estructura interna. Las técnicas de caja blanca (también llamadas técnicas estructurales o basadas en estructura) están basadas en un análisis de la estructura interna del componente o sistema.

Algunas técnicas caen claramente en una simple categoría, otras tienen elementos de más de una categoría. Este programa se refiere a los enfoques basados en especificación o basados en experiencia como técnicas de caja negra y a los basados en estructura como técnicas de caja blanca.

Los rasgos comunes de las técnicas basadas en especificación:

- ✓ Modelos, ya sean formales o informales, son usados para la especificación del problema a ser resuelto, el software o sus componente.
- ✓ De estos modelos los casos de prueba pueden ser derivados sistemáticamente.

Los rasgos comunes de las técnicas basadas en estructura:

- ✓ Información sobre como el software está construido y usado para derivar los casos de prueba, por ejemplo, el código y el diseño.
- ✓ La medida de cobertura del software puede ser medida por los casos de prueba existentes y posteriores casos de prueba pueden ser derivados sistemáticamente para incrementar la cobertura.

Los rasgos comunes de las técnicas basadas en experiencia:

- ✓ El conocimiento y la experiencia de la gente son usados para derivar los casos de prueba.
  - o El conocimiento de los probadores, desarrolladores, usuarios y otras partes interesadas sobre el software, su utilización y su entorno;
  - o El conocimiento sobre defectos probables y su distribución.

4.9 Técnicas basadas en especificación o de caja negra

4.9.1 Términos

Análisis del valor límite, pruebas de tabla de decisión, particiones de equivalencia, pruebas de transición de estado, pruebas de caso de uso.

4.9.2 Particiones de Equivalencia

Las entradas al software o al sistema están dividas en grupos que son esperados de mostrar comportamiento similar, así que son probables de ser procesados en la misma manera. Las particiones (o clases) de equivalencia pueden ser encontradas para tanto datos válidos como datos inválidos, es decir, valores que deben ser rechazados. Las particiones también pueden ser identificadas por salidas, valores internos, valores relacionados con el tiempo (por ejemplo antes o después de un evento) y por parámetros de interface (por ejemplo durante las pruebas de integración). Las pruebas pueden estar diseñadas para cubrir particiones. Las particiones de equivalencia (EP) son aplicables en todos los niveles de prueba.

Las particiones de equivalencia como una técnica pueden ser usadas para alcanzar cobertura de entrada y de salidas. Pueden ser aplicadas a la entrada humana, a interfaces vía entrada a un sistema, o a parámetros de interface en pruebas de integración.

4.9.3 Análisis del valor límite

El comportamiento al borde de cada partición de equivalencia es más probable de ser incorrecto, así los límites son un área donde la prueba es probable de obtener defectos. Los valores máximos y mínimos de una partición son sus valores límite. Un valor límite para una partición válida es un valor límite válido; el límite de una partición inválida es un valor límite inválido. Las pruebas pueden estar diseñadas para cubrir tantos valores límites válidos como inválidos. Al diseñar casos de prueba, un valor sobre cada límite es escogido.

El análisis del valor límite puede ser aplicado a todos los niveles de prueba. Es relativamente fácil de aplicar y su capacidad para encontrar defectos es alta; las especificaciones detalladas son útiles.

Esta técnica es a menudo considerada una extensión de las particiones de equivalencia y puede ser usada sobre entradas por humanos así como, por ejemplo, sobre límites de tabla o sincronización. Los valores límite también pueden ser usados para la selección de datos de prueba.

4.9.4 Pruebas de tabla de decisión

Las tablas de decisión son una buena forma para capturar requisitos de sistema que contienen condiciones lógicas, y para documentar el diseño interno del sistema. Ellas pueden ser usadas

para registras las complejas reglas de negocio que un sistema debe implementar. La especificación es analizada, y las condiciones y acciones del sistema son identificadas. Las condiciones y acciones de entrada son más a menudo establecidas en tal forma que o son verdaderas o son falsas (Booleano). La tabla de decisión contiene las condiciones de activación, a menudo combinaciones de verdadero y falso para todas las condiciones de entrada, y las acciones resultantes para cada combinación de condiciones. Cada columna de la tabla corresponde a una regla del negocio que define una combinación única de condiciones que resultan en la ejecución de las acciones asociadas con esa regla. La cobertura estándar comúnmente usada con las pruebas de tabla de decisión es tener al menos una prueba por columna, la cual involucra típicamente cubrir todas las condiciones de las condiciones de activación.

La fuerza de las pruebas de tabla de decisión es que crea combinaciones de condiciones que no podrían ser ejercidas durante la prueba. Se puede aplicar a todas las situaciones cuando la acción del software depende de varias decisiones lógicas.

4.9.5 Pruebas de transición de estado

Un sistema puede exhibir una respuesta diferente dependiendo de las condiciones actuales o de la historia previa (su estado). En este caso, ese aspecto del sistema puede ser mostrado como un diagrama de transición de estado. Permite que el probador vea el software en términos de sus estados, transiciones entre estados, las entradas o eventos que activan cambios de estado (transiciones) y las acciones que pueden resultar de esas transiciones. Los estados del sistema o del objeto bajo prueba son separados, identificables y finitos en número. Una tabla de estado muestra la relación entre los estados y las entradas, y puede resaltar posibles transiciones que son inválidas. Las pruebas pueden estar diseñadas para cubrir una secuencia típica de estados, para cubrir cada estado, para ejercer cada transición, para ejercer secuencias específicas de transiciones o para probar transiciones inválidas. Las pruebas de transición de estado son más usadas dentro de la integrada industria del software y de la automatización técnica en general. Sin embargo, la técnica es también adecuada para modelar un objeto de negocio que tenga estados específicos o que pruebe flujos de capción de diálogo (por ejemplo para aplicaciones de Internet o escenarios de negocio).

4.9.6 Pruebas de caso de uso

Las pruebas pueden ser especificadas de los casos de uso o escenarios de negocio. Un caso de uso describe interacciones entre actores, incluyendo usuarios y el sistema, los cuales producen un resultado de valor a un usuario de sistema. Cada caso de uso tiene precondiciones, las cuales necesitan ser cumplidas para que un caso de uso trabaje exitosamente. Cada caso de uso finaliza con post-condiciones, las cuales son los resultados observables y el estado final del sistema después que el caso de uso ha sido completado. Un caso de uso usualmente tiene un escenario predominante (es decir, muy probablemente), y a veces bifurcaciones alternativas.

Los casos de uso describen los “flujos de proceso” a través de un sistema basados en su uso probable real, así los casos de prueba derivados de casos de uso son más útiles al descubrir

defectos en los flujos de proceso durante el uso real del sistema. Los casos de uso, a menudo referidos como escenarios, son muy útiles para diseñar pruebas de aceptación con participación cliente/usuario. También ayudan a descubrir defectos de integración causados por la interacción e interferencia de diferentes componentes, lo cual las pruebas de componente individuales no verían.

4.10 Técnicas basadas en estructura o de caja blanca

4.10.1 Términos

Cobertura de código, cobertura de decisión, cobertura de sentencias, pruebas estructurales, pruebas basadas en estructura, pruebas de caja blanca.

4.10.2 Introducción

Las pruebas basadas en estructura/pruebas de caja blanca están basadas en una estructura identificada del software o del sistema, como se ve en los siguientes ejemplos:

- ✓ Nivel de componente: la estructura es la del código en sí, es decir, sentencias, decisiones o bifurcaciones.
- ✓ Nivel de integración: la estructura puede ser un árbol de llamada (un diagrama en el cual los módulos llaman a otros módulos).
- ✓ Nivel de sistema: la estructura puede ser una estructura de menú, una estructura de proceso de negocio o de página web.

En esta sección, dos técnicas estructurales relacionadas con el código para la cobertura de código, basadas en sentencias y decisiones, son discutidas. Para las pruebas de decisión, un diagrama de control de flujo puede ser usado para visualizar las alternativas para cada decisión.

4.10.3 Pruebas y cobertura de sentencias

En las pruebas de componente, la cobertura de sentencias es la evaluación del porcentaje de sentencias ejecutables que han sido ejercidas por un conjunto de prueba. Las pruebas de sentencias derivan casos de prueba para ejecutar sentencias específicas, normalmente para incrementar la cobertura de sentencias.

4.10.4 Pruebas y cobertura de decisión

La cobertura de decisión, relacionada con las pruebas de bifurcación, es la evaluación del porcentaje de salidas de decisión (por ejemplo, opciones Verdadero y Falso de una sentencia SI) que han sido ejercidas por un conjunto de prueba. Las pruebas de decisión derivan casos de

prueba para ejecutar salidas de decisión específicas, normalmente para incrementar la cobertura de decisión.

Las pruebas de decisión son una forma de pruebas de flujo de control conforme genera un flujo específico de control a través de los puntos de decisión. La cobertura de decisión es más fuerte que la cobertura de sentencias: el 100% de la cobertura de decisión garantiza el 100% de la cobertura de sentencias, más no viceversa.

4.10.5 Otras técnicas basadas en estructura

Existen niveles más fuertes de cobertura estructural más allá de la cobertura de decisión, por ejemplo, cobertura de condición y cobertura de condición múltiple.

El concepto de cobertura puede ser también aplicado a otros niveles de prueba (por ejemplo a nivel de integración) donde el porcentaje de módulos, componentes o clases que han sido ejercidos por un conjunto de prueba podría ser expresado como cobertura de módulo, de componente o de clase.

El soporte de herramienta es útil para las pruebas estructurales de código.

4.11 Técnicas basadas en experiencia

4.11.1 Términos

Conjetura de error, pruebas exploratorias.

4.11.2 Introducción

Quizás la más amplia técnica practicada es la conjetura de error. Las pruebas son derivadas de la habilidad e intuición del probador y de su experiencia con aplicaciones y tecnologías similares. Cuando es usada para aumentar las técnicas sistemáticas, las pruebas intuitivas pueden ser útiles para identificar pruebas especiales que no son capturadas fácilmente por técnicas formales, especialmente cuando son aplicadas después de más enfoques formales. Sin embargo, esta técnica puede obtener grados de efectividad ampliamente diferentes, dependiendo de la experiencia del probador. Un enfoque estructurado a la técnica de conjetura de error es enumerar una lista de posibles errores y diseñar pruebas para atacar estos errores. Estas listas de defectos y fallas pueden ser creadas basadas en la experiencia, datos de defectos y fallas disponibles, y del conocimiento común sobre porque falla el software.

Las pruebas exploratorias son diseño de prueba concurrente, ejecución de prueba, registro y aprendizaje de prueba, basadas en una carta de navegación de pruebas que contiene los objetivos de prueba, y llevadas a cabo dentro de relojes de control. Es un enfoque que es muy útil donde existen pocas o inadecuadas especificaciones y severa presión del tiempo, o para aumentar o complementar otras pruebas más formales. Pueden servir como un chequeo sobre el proceso de prueba, para ayudar a garantizar que los defectos más serios sean encontrados.

4.12 Eligiendo las técnicas de prueba

4.12.1 Términos

No hay términos específicos.

4.12.2 Introducción

La elección de cuales técnicas de prueba usar depende de un número de factores, incluyendo el tipo de sistema, estándares reguladores, requisitos contractuales o del cliente, nivel de riesgo, tipo de riesgo, objetivo de prueba, documentación disponible, conocimiento de los probadores, tiempo y presupuesto, ciclo de vida de desarrollo, modelos de caso de uso y experiencia previa de los tipos de defectos encontrados.

Algunas técnicas son más aplicables para ciertas situaciones y niveles de prueba; otras son aplicables a todos los niveles de prueba.

Referencias

4.1 Craig, 2002, Hetzel, 1998, IEEE 829
4.2 Beizer, 1990, Copeland, 2004
4.3.1 Copeland, 2004, Myers, 1979
4.3.2 Copeland, 2004, Myers, 1979
4.3.3 Beizer, 1990, Copeland, 2004
4.3.4 Beizer, 1990, Copeland, 2004
4.3.5 Copeland, 2004
4.4.3 Beizer, 1990, Copeland, 2004
4.5 Kaner, 2002
4.6 Beizer, 1990, Copeland, 2004

5 Gestión de pruebas

Objetivos de aprendizaje para la gestión de pruebas

Los objetivos identifican lo que podrá hacer después de la finalización de cada módulo.

5.1 Organización de pruebas

- ✓ Reconocer la importancia de las pruebas independientes.
- ✓ Listar los beneficios y desventajas de las pruebas independientes dentro de una organización.
- ✓ Reconocer los diferentes miembros del equipo a ser considerados para la creación de un equipo de prueba.
- ✓ Recordar todas las tareas del típico líder de prueba y del probador.

5.2 Planificación y estimación de prueba

- ✓ Reconocer los diferentes niveles y objetivos de la planificación de prueba.
- ✓ Resumir el propósito y el contenido del plan de prueba, la especificación del diseño de prueba y de los documentos de procedimiento de prueba de acuerdo con el "Estándar para la Documentación de Prueba de Software" (IEEE 829).
- ✓ Recordar los típicos factores que influencian el esfuerzo relacionado con la prueba.
- ✓ Diferenciar entre dos enfoques de estimación conceptualmente diferentes: el enfoque basado en las métricas y el enfoque basado en el experto.
- ✓ Diferenciar entre el tema de la planificación de prueba para un proyecto, para niveles de prueba individuales (por ejemplo, prueba del sistema) o para objetivos de prueba específicos (por ejemplo prueba de usabilidad, y para ejecución de prueba).
- ✓ Listar las tareas de preparación y ejecución de prueba que necesitan planificación.
- ✓ Reconocer/justificar los criterios de salida adecuados para los específicos niveles de prueba y grupos de casos de prueba (por ejemplo para pruebas de integración, pruebas de aceptación o casos de prueba para pruebas de usabilidad).

5.3 Monitoreo y control de progreso de prueba

- ✓ Recordar las métricas comunes usadas para la ejecución y preparación de la prueba de monitoreo.
- ✓ Comprender e interpretar las métricas de prueba para el informe de prueba y el control de prueba (por ejemplo, defectos encontrados y arreglados, y las pruebas pasadas y falladas).

- ✓ Resumir el propósito y contenido del documento de informe del resumen de prueba de acuerdo con el "Estándar para la Documentación de Prueba de Software" (IEEE 829).

5.4 Gestión de configuración

- ✓ Resumir como la gerencia de configuración apoya la prueba.

5.5 Riesgo y prueba

- ✓ Describir un riesgo como un posible problema que amenazaría el logro de uno o más objetivos del proyecto de las partes interesadas.
- ✓ Recordar que riesgos son determinados por la probabilidad (de suceder) y el impacto (daño resultante si éste sucede).
- ✓ Distinguir entre los riesgos del proyecto y del producto.
- ✓ Reconocer los típicos riesgos del producto y del proyecto.
- ✓ Describir, usando ejemplos, como el análisis de riesgo y la gestión de riesgo pueden ser usados para la planificación de prueba.

5.6 Gestión de incidentes

- ✓ Reconocer el contenido del informe de incidentes del "Estándar para la Documentación de Prueba de Software" (IEEE 829).
- ✓ Escribir un informe de incidentes cubriendo la observación de una falla durante las pruebas.

5.7 Organización de pruebas

5.7.1 Términos

Probador, líder de prueba, gerente de prueba.

5.7.2 Independencia y organización de prueba

La efectividad de descubrir defectos mediante pruebas y revisiones puede ser mejorada al usar probadores independientes. Las opciones para independencia son:

- ✓ Probadores independientes dentro de los equipos de desarrollo.
- ✓ Equipo o grupo de prueba independiente dentro de la organización, informando a la gestión de proyecto o a la gestión ejecutiva.
- ✓ Probadores independientes para la organización empresarial, comunidad de usuario e IT.

- ✓ Especialistas de prueba independientes para objetivos de prueba específicos tales como probadores de usabilidad, probadores de seguridad o probadores de certificación (quienes certifican un producto de software contra los estándares y regulaciones).
- ✓ Probadores independientes subcontratados o externos a la organización.

Para proyectos críticos de seguridad, complejos o largos, usualmente es mejor tener múltiples niveles de prueba, con algunos o todos los niveles realizados por probadores independientes. El personal de desarrollo puede participar en las pruebas, especialmente a niveles más bajos, pero su falta de objetividad a menudo limita su efectividad. Los probadores independientes pueden tener la autoridad para solicitar y definir los procesos y reglas de prueba, pero los probadores deben asumir dichos roles relacionados con el proceso solamente en la presencia de un mandato de gestión claro para así hacerlo.

Los beneficios de la independencia incluyen:

- ✓ Los probadores independientes ven diferentes defectos y otros, y son imparciales.
- ✓ Un probador independiente puede verificar suposiciones que la gente hizo durante la especificación e implementación del sistema.

Las desventajas incluyen:

- ✓ Aislamiento del equipo de desarrollo (si es tratado como totalmente independiente).
- ✓ Los probadores independientes pueden ser el obstáculo como el último punto de control.
- ✓ Los desarrolladores pierden el sentido de responsabilidad por la calidad.

Las tareas de prueba pueden ser realizadas por gente en un rol de prueba específico, o pueden ser realizadas por alguien en otro rol, tales como un gestor de proyecto, gestor de calidad, desarrollador, experto de negocio y dominio, operaciones de infraestructura o de IT.

5.7.3 Tareas del líder de prueba y del probador

En este programa dos posiciones de prueba están cubiertas, líder de prueba y probador. Las actividades y tareas realizadas por la gente en estos dos roles dependen del contexto del proyecto y del producto, de la gente en los roles y de la organización.

Algunas veces el líder de prueba es llamado gestor de prueba o coordinador de prueba. El rol del líder de prueba puede ser realizado por un gestor de proyecto, un gestor de desarrollo, un gestor de garantía de calidad o el gestor de un grupo de prueba. En proyectos más grandes dos posiciones pueden existir: líder de prueba y gestor de prueba. Típicamente el líder de prueba planea, monitorea y controla las actividades y tareas de prueba como está definido en la sección 1.4.

Típicamente las tareas del líder de prueba pueden incluir:

- ✓ Coordinar la estrategia y el plan de prueba con los gestores de proyecto y otros.
- ✓ Escribir o revisar una estrategia de prueba para el proyecto y probar la política para la organización.
- ✓ Contribuir la perspectiva de prueba a las otras actividades del proyecto, tales como la planificación de integración.
- ✓ Planear las pruebas – considerando el contexto y comprendiendo los riesgos – incluyendo seleccionar los enfoques de prueba, estimar el tiempo, esfuerzo y el costo de las pruebas, adquirir recursos, definir los niveles, ciclos, enfoque y objetivos de prueba, y planificar la gestión de incidentes;
- ✓ Iniciar la especificación, preparación, implementación y ejecución de pruebas, y monitorear y controlar la ejecución.
- ✓ Adaptar la planificación basada en el progreso y los resultados de prueba (a veces documentados en los reportes de estado) y tomar cualquier acción necesaria para compensar por problemas.
- ✓ Establecer la gestión de configuración adecuada del testware para trazabilidad.
- ✓ Introducir las métricas apropiadas para medir el progreso de la prueba y evaluar la calidad de las pruebas y del producto.
- ✓ Decidir que debe ser automatizado, hasta qué grado y como.
- ✓ Elegir las herramientas para apoyar las pruebas y organizar cualquier entrenamiento en el uso de herramientas para los probadores.
- ✓ Decidir sobre la implementación del entorno de prueba.
- ✓ Programar las pruebas.
- ✓ Escribir informes de resumen de prueba basados en la información reunida durante las pruebas.

Las típicas tareas del probador pueden incluir:

- ✓ Revisar y contribuir a los planes de prueba.
- ✓ Analizar, revisar y evaluar los requisitos de usuario, especificaciones y modelos para testabilidad.
- ✓ Crear especificaciones de prueba.
- ✓ Configurar el entorno de prueba (a menudos coordinando con la administración del sistema y la gestión de red).
- ✓ Preparar y adquirir datos de prueba.
- ✓ Implementar pruebas en todos los niveles de prueba, ejecutar y registrar las pruebas, evaluar los resultados y documentar las desviaciones de los resultados esperados.
- ✓ Usar la administración de prueba o las herramientas de gestión y probar las herramientas de prueba como sea requerido.
- ✓ Automatizar las pruebas (puede ser apoyado por un desarrollador o un experto en automatización de prueba).
- ✓ Medir el rendimiento de los componentes y sistemas (de ser aplicable).
- ✓ Revisar las pruebas desarrolladas por otros.

Las personas que trabajan en análisis de prueba, diseño de prueba, tipos de prueba específicos o en automatización de prueba pueden ser especialistas en estos roles. Dependiendo del nivel de prueba y de los riesgos relacionados con el producto y el proyecto, diferentes personas pueden encargarse del rol de probador, manteniendo algún grado de independencia. Típicamente los probadores a nivel de componente e integración serían desarrolladores, probadores al nivel de prueba de aceptación serían usuarios y expertos del negocio, y los probadores para pruebas de aceptación operacionales serían operadores.

5.7.4 Planificación y estimación de prueba

5.7.5 Términos

Criterios de entrada, criterios de salida, pruebas exploratorias, enfoque de prueba, nivel de prueba, plan de prueba, procedimiento de prueba, estrategia de prueba.

5.7.6 Planificación de prueba

Esta sección cubre el propósito de la planificación de prueba dentro de los proyectos de desarrollo y de implementación, y para las actividades de mantenimiento. La planificación puede estar documentada en un proyecto o en un plan de prueba maestro, y en planes de prueba separados para niveles de prueba, tales como pruebas de sistema y pruebas de aceptación. Esbozos de los documentos de planificación de prueba están cubiertos por el "Estándar para la Documentación de Prueba de Software" (IEEE 829).

La planificación está influenciada por la política de prueba de la organización, el alcance de las pruebas, objetivos, riesgos, restricciones, criticidad, testabilidad y disponibilidad de los recursos. Entre más progrese la planificación de prueba y del proyecto, más información estará disponible y más detalles pueden ser incluidos en el plan.

La planificación de prueba es una actividad continua y es realizada en todos los procesos de ciclo de vida y actividades. La información de las actividades de prueba es usada para reconocer riesgos cambiantes para que la planificación pueda ser ajustada.

5.7.7 Actividades de planificación de prueba

Las actividades de planificación de prueba pueden incluir:

- ✓ Definir el enfoque global de la prueba (la estrategia de prueba), incluyendo la definición de los niveles de prueba y los criterios de entrada y salida.
- ✓ Integrar y coordinar las actividades de prueba a las actividades de ciclo de vida del software: adquisición, provisión, desarrollo, operación y mantenimiento.
- ✓ Realizar decisión sobre que probar, que roles realizarán las actividades de prueba, cuando y como deben ser realizadas las actividades de prueba, como serán evaluados los resultados de prueba y cuando detener las pruebas (criterios de salida).

- ✓ Asignar recursos para las diferentes tareas definidas.
- ✓ Definir el monto, nivel de detalle, estructura y plantillas para la documentación de prueba.
- ✓ Seleccionar las métricas para monitorear y controlar la preparación y ejecución de prueba, resolución de defecto y los problemas de riesgo.
- ✓ Establecer el nivel de detalle para los procedimientos de prueba para proporcionar suficiente información para apoyar a la preparación y ejecución de prueba reproducibles.

5.7.8 Criterios de salida

El propósito de los criterios de salida es definir cuando detener las pruebas, tales como al final de un nivel de prueba o cuando un conjunto de pruebas tiene una meta específica.

Típicamente los criterios de salida pueden consistir de:

- ✓ Medidas de minuciosidad, tales como cobertura de código, funcionalidad o riesgo.
- ✓ Estimaciones de la densidad del defecto o de las medidas de fiabilidad.
- ✓ Costo.
- ✓ Riesgos residuales, tales como defectos no arreglados o falta de cobertura de prueba en ciertas áreas.
- ✓ Agendas tales como aquellas basadas a tiempo para el mercado.

5.7.9 Estimación de prueba

Dos enfoques para la estimación del esfuerzo de prueba están cubiertos en este programa:

- ✓ Estimar el esfuerzo de prueba basado en las métricas de proyectos anteriores o similares, o basado en valores típicos.
- ✓ Estimar las tareas por el dueño de estas tareas o por expertos.

Una vez que el esfuerzo de prueba es estimado, los recursos pueden ser identificados y una agenda puede ser formulada.

El esfuerzo de prueba puede depender de un número de factores, incluyendo:

- ✓ Las características del producto: la calidad de la especificación y de otra información usada para los modelos de prueba (es decir, la base de prueba), el tamaño del producto, la complejidad del dominio del problema, los requisitos para fiabilidad y seguridad, y los requisitos para la documentación.
- ✓ Las características del proceso de desarrollo: la estabilidad de la organización, las herramientas usadas, el proceso de prueba, las habilidades de la gente involucrada y la presión del tiempo.
- ✓ El resultado de las pruebas: el número de defectos y el monto del rediseño requerido.

5.7.10 Enfoques de prueba (estrategias de prueba)

Una manera de clasificar las estrategias o enfoques de prueba está basada en el punto en el tiempo en el cual el volumen del trabajo de diseño de prueba comienza:

- ✓ Enfoques preventivos, donde las pruebas son diseñadas tan pronto como sea posible.
- ✓ Enfoques reactivos, donde el diseño de prueba viene después que el software o el sistema ha sido producido.

Los enfoques o estrategias típicas incluyen:

- ✓ Enfoques analíticos, tales como pruebas basadas en riesgos donde la prueba está dirigida hacia áreas de mayor riesgo.
- ✓ Enfoques basados en modelos, tales como pruebas fortuitas usando información estadística sobre frecuencia de fallas (tales como modelos de crecimiento de fiabilidad) o uso (tales como perfiles operacionales).
- ✓ Enfoques metódicos, tales como basados en fallas (incluyendo conjetura de error y ataques de falla), basados en listas de comprobación, y basados en características de calidad.
- ✓ Enfoques compatibles con el estándar o el proceso, tales como aquellos especificados por los estándares específicos de la industria o las variadas metodologías ágiles.
- ✓ Enfoques dinámicos y heurísticos, tales como pruebas exploratorias donde la prueba es más reactiva a los eventos que a lo pre-planeado, y donde la ejecución y evaluación son tareas concurrentes.
- ✓ Enfoques consultivos, tales como aquellos donde la cobertura de prueba es conducida principalmente por el consejo y guía de expertos en tecnología y/o en dominio del negocio fuera del equipo de prueba.
- ✓ Enfoques de regresión adversa, tales como aquellos que incluyen la reutilización del material de prueba existente, automatización extensiva de pruebas de regresión funcionales y conjuntos de prueba estándar.

Los diferentes enfoques pueden ser combinados, por ejemplo, un enfoque dinámico basado en riesgos.

La selección de un enfoque de prueba debe considerar el contexto, incluyendo:

- ✓ Riesgo de fracaso del proyecto, peligros hacia el producto y riesgos de falla del producto hacia los humanos, el entorno y la compañía.
- ✓ Habilidades y experiencia de la gente en las técnicas, herramientas y métodos propuestos.
- ✓ El objetivo de la iniciativa de prueba y la misión del equipo de prueba.
- ✓ Los aspectos reguladores, tales como regulaciones internas y externas para el proceso de desarrollo.
- ✓ La naturaleza del producto y del negocio.

5.8 Monitoreo y control de progreso de prueba

5.8.1 Términos

Densidad de defectos, frecuencia de fallas, control de prueba, cobertura de prueba, monitoreo de prueba, informe de prueba.

5.8.2 Monitoreo de control de prueba

El propósito del monitoreo de prueba es dar información y visibilidad sobre las actividades de prueba. La información a ser monitoreada puede ser recolectada manual o automáticamente y puede ser usada para medir criterios de salida, tales como cobertura. Las métricas pueden ser también usadas para evaluar el progreso contra la agenda planeada y el presupuesto.

Las métricas de prueba comunes incluyen:

- ✓ Porcentaje del trabajo realizado en la preparación del caso de prueba (o porcentaje de los casos de prueba planeados y preparados).
- ✓ Porcentaje del trabajo realizado en la preparación del entorno de prueba.
- ✓ Ejecución del caso de prueba (por ejemplo, número de casos de prueba corridos/no corridos, y los casos de prueba pasados/fallados).
- ✓ Información de defectos (por ejemplo, densidad de defectos, los defectos encontrados y arreglados, frecuencia de fallas y resultados de re-prueba).
- ✓ Cobertura de prueba de los requisitos, riesgos o del código.
- ✓ Confianza subjetiva de los probadores en el producto.
- ✓ Fechas de los hitos de prueba.
- ✓ Costos de prueba, incluyendo el costo comparado con el beneficio de encontrar el siguiente defecto o de correr la siguiente prueba.

5.8.3 Reporte de prueba

El reporte de prueba se preocupa de resumir la información sobre la iniciativa de prueba, incluyendo:

- ✓ Lo que pasó durante un periodo de prueba, tales como fechas cuando se cumplieron los criterios de salida.
- ✓ Información y métricas analizadas para apoyar recomendaciones y decisiones sobre futuras acciones, tales como una evaluación de los defectos restantes, el beneficio económico de pruebas constantes, riesgos sobresalientes y el nivel de confianza en el software probado.

El esbozo de un informe de resumen de prueba es dado en el "Estándar para la Documentación de Prueba de Software" (IEEE 829).

Las métricas deben ser recolectadas durante y al final de un nivel de prueba para evaluar:

- ✓ La conveniencia de los objetivos de prueba para ese nivel de prueba.
- ✓ La conveniencia de los enfoques de prueba tomados.
- ✓ La efectividad de la prueba con respecto a sus objetivos.

5.8.4 Control de prueba

EL control de prueba describe cualquier acción correctiva o de guía tomada como un resultado de la información y de las métricas tomadas e informadas. Las acciones pueden cubrir cualquier actividad de prueba y pueden afectar cualquier otra actividad o tarea del ciclo de vida del software.

Son ejemplos de las acciones de control de prueba:

- ✓ Re-priorizar las pruebas cuando ocurre un riesgo no identificado (por ejemplo, software entregado tarde).
- ✓ Cambiar la agenda de prueba debido a disponibilidad de un entorno de prueba.
- ✓ Establecer un criterio de entrada que requiere arreglos para haber sido re-probados por un desarrollador antes de aceptarlos en una construcción.

5.9 Gestión de configuración

5.9.1 Términos

Gestión de configuración, control de versión.

5.9.2 Introducción

El propósito de la gestión de configuración es establecer y mantener la integridad de los productos (componentes, datos y documentación) del software o del sistema a través del ciclo de vida del proyecto y del producto.

Para las pruebas, la gestión de configuración puede involucrar el garantizar que:

- ✓ Todos los elementos del testware están identificados, la versión está controlada, se han rastreado los cambios, relacionados el uno con el otro y relacionados con los elementos de desarrollo (objetos de prueba) para que la trazabilidad pueda ser mantenida a lo largo del proceso de prueba.
- ✓ Todos los documentos identificados y los elementos del software están referenciados sin ambigüedades en la documentación de prueba.

Para el probador, la gestión de configuración ayuda a identificar unívocamente (y a reproducir) el elemento probado, los documentos de prueba, las pruebas y la armadura de pruebas.

Durante la planificación de prueba, los procedimientos de la gestión de configuración y la infraestructura (herramientas) deben ser escogidos, documentados e implementados.

5.10 Riesgo y prueba

5.10.1 Términos

Riesgo de producto, riesgo de proyecto, riesgo, pruebas basadas en riesgos.

5.10.2 Introducción

Riesgo puede ser definido como la oportunidad de que ocurra un evento, peligro, amenaza o situación y sus consecuencias no deseables, un problema potencial. El nivel de riesgo estará determinado por la probabilidad de que suceda un caso adverso y por el impacto (el daño resultante de ese evento).

5.10.2.1 Riesgos de proyecto

Los riesgos de proyecto son riesgos que rodean la capacidad del proyecto de entregar sus objetivos, tales como:

- ✓ Asuntos del proveedor:
  - o Fracaso de un tercero;
  - o Asuntos contractuales.
- ✓ Factores organizacionales:
  - o Falta de habilidad y personal;
  - o Asuntos personales y de entrenamiento;
  - o Asuntos políticos, tales como:
- ✓ Problemas con los probadores al comunicar sus necesidades y resultados de prueba;
- ✓ Falla al seguir la información encontrada en la prueba y en las revisiones (por ejemplo, al no mejorar las prácticas de desarrollo y de prueba).
  - o Actitud inapropiada hacia la o expectativas de la prueba (por ejemplo, al no apreciar el valor de encontrar defectos durante la prueba).
- ✓ Asuntos técnicos:
  - o Problemas al definir los requisitos correctos;
  - o La medida en que los requisitos pueden ser cumplidos dadas las restricciones existentes; o La calidad del diseño, código y de las pruebas.

Al analizar, gestar y mitigar estos riesgos, el gestor de prueba está siguiendo principios de gestión de proyecto bien establecidos. El esbozo del "Estándar para la Documentación de Prueba de Software" (IEEE 829) para planes de prueba requiere que los riesgos y contingencias sean manifestados.

5.10.2.2 Riesgos de producto

Las potenciales áreas de falla (eventos o peligros futuros y adversos) en el software o sistema son conocidos como riesgos de producto, ya que son un riesgo a la calidad del producto, tales como:

- ✓ Software propenso a errores entregado.
- ✓ El potencial de que el software/hardware pudiese causar daño a un individuo o compañía.
- ✓ Pobres características del software (por ejemplo, funcionalidad, seguridad, fiabilidad, usabilidad y rendimiento).
- ✓ Software que no realiza sus funciones pretendidas.

Los riesgos son usados para decidir donde comenzar la prueba y donde probar más; la prueba es usada para reducir el riesgo de que ocurra un efecto adverso o para reducir el impacto de un efecto adverso.

Los riesgos de producto son un tipo especial de riesgo al éxito de un proyecto. La prueba como una actividad para el control de riesgos proporciona información sobre el riesgo residual al medir la efectividad del retiro de defectos críticos y de los planes de contingencia.

Un enfoque basado en riesgos hacia la prueba proporciona oportunidades proactivas para reducir los niveles del riesgo de producto, empezando en las fases iniciales de un proyecto. Involucra la identificación de los riesgos de producto y su uso al guiar la planificación de prueba, especificación, preparación y ejecución de pruebas. En un enfoque basado en riesgos, los riesgos identificados pueden ser usados para:

- ✓ Determinar las técnicas de prueba a ser empleadas.
- ✓ Determinar la medida de la prueba a ser llevada a cabo.
- ✓ Priorizar la prueba en un intento de encontrar los defectos críticos tan pronto como sea posible.
- ✓ Determinar si cualquier actividad no relacionada con pruebas podría ser empleada para reducir el riesgo (por ejemplo, proporcionar entrenamiento a diseñadores inexpertos).

La prueba basada en riesgos se basa en el conocimiento colectivo y en la perspicacia de las partes interesadas del proyecto para determinar los riesgos y los niveles de prueba requeridos para tratar esos riesgos.

Para garantizar que la oportunidad de una falla de producto sea minimizada, las actividades de la gestión de riesgo proporcionan un enfoque disciplinado para:

- ✓ Evaluar (y re-evaluar regularmente) que puede salir mal (riesgos).
- ✓ Determinar que riesgos son importantes para ser tratados.
- ✓ Implementar acciones para tratar con esos riesgos.

Además, la prueba puede apoyar la identificación de nuevos riesgos, puede ayudar a determinar que riesgos deben ser reducidos y puede bajar la incertidumbre sobre los riesgos.

5.11 Gestión de incidentes

5.11.1 Términos

Registro de incidentes.

5.11.2 Introducción

Dado que uno de los objetivos de la prueba es encontrar defectos, las discrepancias entre los resultados reales y los esperados necesitan ser registrados como incidentes. Los incidentes deben ser rastreados desde el descubrimiento y la clasificación hasta la corrección y confirmación de la solución. Para gestar todos los incidentes hasta la finalización, una organización debe establecer un proceso y reglas para la clasificación.

Los incidentes pueden surgir durante el desarrollo, revisión, prueba o uso de un producto de software. Pueden surgir de problemas en el código o en el sistema en el que se trabaja, o en cualquier tipo de documentación incluso en documentos de desarrollo, documentos de prueba o información del usuario tales como “Ayuda” o en guías de instalación.

Los informes de incidentes tienen los siguientes objetivos:

- ✓ Proporcionar a los desarrolladores y a las otras partes de información sobre el problema para permitir la identificación, aislamiento y corrección como sea necesario.
- ✓ Proporcionar a los líderes de prueba un medio para rastrear la calidad del sistema bajo prueba y el progreso de la prueba.
- ✓ Proporcionar ideas para la mejora del proceso de prueba.

Un probador o revisor típicamente registran la siguiente información, de ser conocida, con respecto a un incidente:

- ✓ Fecha del problema, organización emisora, autor, aprobaciones y estado.
- ✓ Alcance, severidad y prioridad del incidente.
- ✓ Referencias, incluso la identidad de la especificación del caso de prueba que reveló el problema.

Los detalles del informe de incidentes pueden incluir:

- ✓ Resultados esperados y reales.
- ✓ Fecha en que el incidente fue descubierto.
- ✓ Elemento de identificación o configuración del software o sistema.
- ✓ Proceso del ciclo de vida del sistema o del software en el cual se observó el incidente.
- ✓ Descripción de la anomalía para permitir la solución.
- ✓ Grado de impacto en los intereses de la(s) parte(s) interesada(s).
- ✓ Severidad del impacto en el sistema.
- ✓ Urgencia/prioridad para ser arreglado.

- ✓ Estado del incidente (por ejemplo, abierto, deferido, duplicado, esperando a ser arreglado, arreglado esperando prueba de confirmación o cerrado).
- ✓ Conclusiones y recomendaciones.
- ✓ Problemas globales, tales como otras áreas que puedan ser afectadas por un cambio resultante del incidente.
- ✓ Historia de cambio, tales como la secuencia de acciones tomadas por los miembros de equipo del proyecto con respecto al incidente para aislarlo, repararlo y confirmarlo como arreglado.

La estructura de un informe de incidentes está cubierta en el "Estándar para la Documentación de Prueba de Software" (IEEE 829) y es llamado un informe de anomalías.

Referencias

5.1.1 Black, 2001, Hetzel, 1998
5.1.2 Black, 2001, Hetzel, 1998
5.2.5 Black, 2001, Craig, 2002, IEEE 829, Kaner 2002
5.3.3 Black, 2001, Craig, 2002, Hetzel, 1998, IEEE 829

6 Soporte de herramientas para prueba

Objetivos de aprendizaje para el soporte de herramientas para prueba

Los objetivos identifican lo que podrá hacer después de la finalización de cada módulo.

6.1 Tipos de herramienta de prueba

- ✓ Clasificar los diferentes tipos de herramientas de prueba de acuerdo con las actividades del proceso de prueba.
- ✓ Reconocer las herramientas que pueden ayudar a los desarrolladores en su prueba.

6.2 Uso efectivo de herramientas: beneficios y riesgos potenciales

- ✓ Resumir los beneficios y riesgos potenciales de la automatización de prueba y del soporte de herramienta para la prueba.
- ✓ Reconocer que herramientas de ejecución de prueba puede tener diferentes técnicas de scripting, incluyendo guiadas por datos y guiadas por palabra clave.

6.3 Introduciendo una herramienta a una organización

- ✓ Manifestar los principios principales de introducir una herramienta a una organización.
- ✓ Manifestar las metas de una fase de muestra/prueba de concepto para la evaluación de la herramienta.
- ✓ Reconocer que factores aparte de simplemente adquirir una herramienta son requeridos para un buen soporte de herramienta.

6.4 Tipos de herramienta de prueba

6.4.1 Términos

Herramienta de gestión de configuración, herramienta de medida de cobertura, herramienta de depuración, controlador, herramienta de análisis dinámico, herramienta de gestión de incidentes, herramienta de pruebas de carga, herramienta de modelación, herramienta de monitoreo, herramienta de pruebas de rendimiento, efecto sonda, herramienta de gestión de requisitos, herramienta de soporte del proceso de revisión, herramienta de seguridad, herramienta de análisis estático, herramienta de pruebas de estrés, agente de sustitución, comparador de pruebas, herramienta de preparación de datos de prueba, herramienta de diseño de pruebas, armadura de pruebas, herramienta de ejecución de pruebas, herramienta de gestión de pruebas, herramienta de marco de trabajo para pruebas unitarias.

6.4.2 Clasificación de herramienta de trabajo

Existen un número de herramientas que soportan diferentes aspectos de las pruebas. Las herramientas están clasificadas en este programa de acuerdo a las actividades de prueba que estas soporten.

Algunas herramientas claramente soportan una actividad; otras pueden soportar más de una actividad, pero están clasificadas bajo la actividad con la que están más cercanamente asociadas. Algunas herramientas comerciales ofrecen soporte para solamente un tipo de actividad; otros distribuidores de herramientas comerciales ofrecen juegos o familias de herramientas que proporcionan soporte a muchas de o a todas estas actividades.

Las herramientas de prueba pueden mejorar la eficiencia de las actividades de prueba al automatizar tareas repetitivas. Las herramientas de prueba también pueden mejorar la fiabilidad de la prueba al, por ejemplo, automatizar comparaciones de datos o al simular comportamiento.

Algunos tipos de herramienta de prueba pueden ser intrusos en que la herramienta en sí puede afectar el resultado real de la prueba. Por ejemplo, la medición del tiempo real puede ser diferente dependiendo de cómo se mida con diferentes herramientas de rendimiento, o se puede obtener una medida diferente de la cobertura de código dependiendo en que herramienta de cobertura se usa. La consecuencia de las herramientas intrusas se llámale efecto sonda.

Algunas herramientas ofrecen soporte más apropiado para desarrolladores (por ejemplo, durante las pruebas de componente y de integración de componente). Tales herramientas están marcadas con la letra “(D)” en las clasificaciones abajo.

6.4.3 Soporte de herramienta para gestión de pruebas

Las herramientas de gestión se aplican a todas las actividades de prueba sobre el entero ciclo de vida del software.

6.4.4 Herramientas de gestión de prueba

Las características de las herramientas de gestión de prueba incluyen:

- ✓ Soporte para la gestión de pruebas y para las actividades de prueba llevadas a cabo.
- ✓ Interfaces a las herramientas de ejecución de prueba, herramientas de rastreo de defectos y a herramientas de gestión de requisitos.
- ✓ Control de versión independiente o interfaz con una herramienta externa de gestión de configuración.
- ✓ Soporte para la trazabilidad de pruebas, resultados de prueba e incidentes a los documentos de fuente, tales como especificaciones de requisito.
- ✓ Registro de los resultados de prueba y generación de informes de progreso.
- ✓ Análisis cuantitativo (métricas) relacionado con las pruebas (por ejemplo, pruebas corridas y pruebas pasadas) y con el objeto de prueba (por ejemplo, incidentes surgidos), para dar información sobre el objeto de prueba y para controlar y mejorar el proceso de prueba.

6.4.5 Herramientas de gestión de requisitos

Las herramientas de gestión de requisitos guardan las sentencias de requisito, verifican la consistencia y los requisitos no definidos (faltantes), permiten que los requisitos sean priorizados y permiten que las pruebas individuales sean rastreables a los requisitos, funciones y/o características. La trazabilidad puede ser reportada en los informes de progreso de la gestión de prueba. La cobertura de requisitos, funciones y/o características por un conjunto de probadores también puede ser informada.

6.4.6 Herramientas de gestión de incidentes

Las herramientas de gestión de incidentes guardan y administran los informes de incidentes, es decir, defectos, fallas o problemas percibidos y anomalías, y soportan la gestión de informes de incidente en formas que incluyen:

- ✓ Facilitar la priorización.
- ✓ Asignación de acciones a las personas (por ejemplo, prueba de confirmación o de arreglo).
- ✓ Atribución de estado (por ejemplo, rechazado, listo para ser probado o diferido al siguiente lanzamiento).

Estas herramientas permiten que el progreso de incidentes sea monitoreado con el paso del tiempo, a menudo proporcionan soporte para el análisis estadístico y proporcionan informes sobre incidentes. Son también conocidas como herramientas de rastreo de defectos.

6.4.7 Herramientas de gestión de configuración

Las herramientas de gestión de configuración (CM) no son estrictamente herramientas de prueba, pero son típicamente necesarias para seguir la pista de diferentes versiones y construcciones del software y de las pruebas.

Las herramientas de gestión de configuración:

- ✓ Guardan información sobre versiones y construcciones de software y testware.
- ✓ Permiten la trazabilidad entre el testware y los productos software y las variantes del producto.
- ✓ Son particularmente útiles al desarrollar en más de una configuración del entorno de hardware/software (por ejemplo, para diferentes versiones del sistema operativo, diferentes librerías y compiladores, diferentes navegadores o diferentes computadoras).

6.4.8 Herramienta de soporte para prueba estática

Herramientas de soporte para procesos de revisión

Las herramientas de soporte para procesos de revisión guardan información sobre procesos de revisión, guardan y comunican los comentarios de revisión, informan sobre defectos y esfuerzos, administran referencias a las reglas de revisión y/o listas de comprobación y siguen la trazabilidad entre documentos y código fuente. También pueden proporcionar ayuda para revisiones en línea, lo cual es útil si el equipo está dispersado geográficamente.

6.4.8.1 Herramientas de análisis estático (D)

Las herramientas de análisis estático soportan desarrolladores, probadores y personal de garantía de calidad al encontrar defectos antes de las pruebas dinámicas. Sus principales propósitos incluyen:

- ✓ La ejecución de los estándares de codificación.
- ✓ El análisis de estructuras y dependencias (por ejemplo, páginas web vinculadas).
- ✓ Ayudar a la comprensión del código.
- ✓ Las herramientas de análisis estático pueden calcular métricas del código (por ejemplo, complejidad), las cuales pueden dar
- ✓ Información valiosa, por ejemplo, para planificación o análisis de riesgo.

6.4.8.2 Herramientas de modelación (D)

Las herramientas de modelación pueden validar modelos del software. Por ejemplo, un chequeador de modelo de base de datos puede encontrar defectos e inconsistencias en el modelo de datos; otras herramientas de modelación pueden encontrar defectos en un modelo de estado o en un modelo de objeto. Estas herramientas pueden a menudo ayudar en la generación de algunos casos de prueba basadas en el modelo (ver también herramientas de diseño de prueba abajo).

El principal beneficio de las herramientas de análisis estático y de las herramientas de modelación es la efectividad de costo de encontrar más defectos en un momento más temprano en el proceso de desarrollo. Como un resultado, el proceso de desarrollo puede acelerarse y mejorarse al tener menos corrección del trabajo.

6.4.8.3 Soporte de herramienta para especificación de prueba

Herramientas de diseño de pruebas

Las herramientas de diseño de pruebas generan entradas de prueba o las pruebas reales de los requisitos, de una interfaz gráfica de usuario, de modelos de diseño (estado, datos u objeto) o del código. Este tipo de herramienta puede generar resultados esperados también (es decir, puede usar un oráculo de prueba). Las pruebas generadas de un modelo de estado u objeto son útiles para verificar la implementación del modelo en el software, pero son rara vez suficientes para verificar todos los aspectos del software o del sistema. Pueden ahorrar tiempo valioso y proporcionar minuciosidad incrementada de las pruebas debido al grado de compleción de las pruebas que la herramienta generar.

Otras herramientas en esta categoría pueden ayudar a soportar la generación de pruebas al proporcionar plantillas estructuradas, a veces llamadas un marco de prueba, que generan pruebas y agentes de sustitución de prueba, y por lo tanto acelerar el proceso del diseño de prueba.

6.4.8.4 Herramientas para preparación de datos de prueba (D)

Las herramientas para preparación de datos de prueba manipulan las bases de datos, archivos o las transmisiones de datos para configurar los datos de prueba a ser usados durante la ejecución de pruebas. Un beneficio de estas herramientas es garantizar que los datos vivos transferidos a un entorno de prueba se vuelvan anónimos, para la protección de datos.

6.4.9 Soporte de herramienta para ejecución de prueba y registro

Herramientas de ejecución de prueba

Las herramientas de ejecución de prueba permiten que las pruebas sean ejecutadas automáticamente, o semi-automáticamente, usando entradas guardadas y resultados esperados, a través del uso de un lenguaje de scripting. El lenguaje de scripting hace posible la manipulación de las pruebas con esfuerzo limitado, por ejemplo, para repetir la prueba con datos diferentes o para probar una parte diferente del sistema con pasos similares. Generalmente estas herramientas incluyen características de comparación dinámica y proporcionan un registro de prueba por cada corrida de prueba.

Las herramientas de ejecución de prueba también pueden ser usadas para registrar pruebas, cuando puedan ser referidas como herramientas de reproducción de captura. Capturar entradas de prueba durante las pruebas exploratorias o pruebas sin script puede ser útil para reproducir y/o documentar una prueba, por ejemplo, si ocurre una falla.

6.4.9.1 Herramientas de armadura de pruebas/marco de trabajo para pruebas unitarias (D)

Una armadura de pruebas puede facilitar las pruebas de componentes o parte de un sistema al similar el entorno en el cual el objeto de prueba correrá. Esto puede ser realizado ya sea porque otros componentes del entorno todavía no están disponibles y están reemplazados por agentes de sustitución y/o controladores, o simplemente para proporcionar un entorno predecible y controlable en el cual cualquier falla puede ser localizada al objeto bajo prueba.

El marco de trabajo puede ser creado donde parte del código, objeto, método o función, unidad o componente pueda ser ejecutada, al llamar al objeto a ser probado y/o al dar información a ese objeto. Puede hacer esto al proporcionar medios artificiales de suministrar entrada al objeto de prueba, y/o al suministrar agentes de sustitución para tomar salida del objeto, en lugar de los objetivos reales de salida.

Las herramientas de armadura de pruebas también pueden ser usadas para proporcionar un marco de trabajo de ejecución en middleware, donde los lenguajes, sistemas operativos o el hardware deben ser probados juntos.

Pueden ser llamadas herramientas de marco de trabajo para pruebas unitarias cuando tengan un foco particular en el nivel de prueba del componente. Este tipo de herramienta ayuda a ejecutar las pruebas de componente en paralelo con la creación del código.

6.4.9.2 Comparadores de prueba

Los comparadores de prueba determinan las diferencias entre los archivos, las bases de datos o los resultados de prueba. Las herramientas de ejecución de prueba típicamente incluyen comparadores dinámicos, pero la comparación post-ejecución puede ser realizada por una herramienta de comparación por separado. Un comparador de prueba puede usar un oráculo de prueba, especialmente si es automatizado.

6.4.9.3 Herramientas de medición de cobertura (D)

Las herramientas de medición de cobertura pueden ser intrusas o no intrusas dependiendo de las técnicas de medición usadas, de lo que es medido y del leguaje de codificación. Las herramientas de cobertura de código miden el porcentaje de los tipos específicos de la estructura de código que han sido ejercidos (por ejemplo, sentencias, bifurcaciones o decisiones, y llamadas de módulo o función). Estas herramientas muestran cuan minuciosamente ha sido ejercido el tipo de estructura medido por un conjunto de pruebas.

6.4.9.4 Herramientas de seguridad

Las herramientas de seguridad verifican los virus de computadora y el rechazo de los ataques de servicio. Un firewall, por ejemplo, no es estrictamente una herramienta de prueba, pero puede ser usado en las pruebas de seguridad. Otras herramientas de seguridad presionan al sistema al buscar vulnerabilidades específicas del sistema.

6.4.10 Soporte de herramienta para rendimiento y monitoreo

6.4.10.1 Herramientas de análisis dinámico (D)

Las herramientas de análisis dinámico encuentran defectos que son evidentes solamente cuando el software se está ejecutando, tales como dependencias de tiempo o pérdidas de memoria. Son usadas típicamente en pruebas de componente y de integración de componente, y al probar el middleware.

6.4.10.2 Herramientas de pruebas de estrés/pruebas de carga/pruebas de rendimiento

Las herramientas de pruebas de rendimiento monitorean e informan sobre como un sistema se comporta bajo una variedad de condiciones de uso simuladas. Simulan la carga de una aplicación, una base de datos o un entorno de sistema, tales como una red o un servidor. Las herramientas son a menudo nombradas por el aspecto del rendimiento que miden, tales como carga o estrés, así que también son conocidas como herramientas de pruebas de carga o herramientas de

pruebas de estrés. Están basadas a menudo en la ejecución repetitiva y automatizada de pruebas, controladas por parámetros.

6.4.10.3 Herramientas de monitoreo

Las herramientas de monitoreo no son estrictamente herramientas de prueba mas proporcionan información que puede ser usada para los propósitos de prueba y la cual no está disponible por otros medios.

Las herramientas de monitoreo continuamente analizan, verifican e informan sobre la utilización de recursos específicos de sistema y dan advertencias de posibles problemas del servicio. Guardan información sobre la versión y construcción del software y testware, y permiten la trazabilidad.

6.4.11 Soporte de herramienta para áreas específicas de aplicación

Los ejemplos individuales de los tipos de herramientas clasificados arriba pueden estar especializados para el uso en un tipo particular de aplicación. Por ejemplo, existen herramientas de pruebas de rendimiento específicamente para aplicaciones basadas en la web, herramientas de análisis estático para plataformas específicas de desarrollo y herramientas de análisis dinámico específicamente para probar los aspectos de seguridad.

Los conjuntos de herramientas comerciales pueden ir dirigidas a áreas específicas de aplicación (por ejemplo, sistemas integrados).

6.4.12 Soporte de herramienta usando otras herramientas

Las herramientas de prueba listada aquí no son los únicos tipos de herramientas usadas por los probadores – también pueden usar hojas de cálculo, SQL, herramientas de recurso o depuración (D), por ejemplo.

6.5 Uso efectivo de herramientas: beneficios y riesgos potenciales

6.5.1 Términos

Guiado por datos (prueba), guiado por palabra clave (prueba), lenguaje de scripting.

6.5.2 Beneficios y riesgos potenciales del soporte de herramienta para prueba (para toda herramienta)

El simplemente comprar o alquilar una herramienta no garantiza el éxito con esa herramienta. Cada tipo de herramienta puede requerir esfuerzo adicional para alcanzar los beneficios reales y duraderos. Existen potenciales oportunidades de beneficio con el uso de herramientas en la prueba, pero también existen riesgos.

Los beneficios potenciales de usar herramientas incluyen:

- ✓ El trabajo repetitivo se reduce (por ejemplo, al correr pruebas de regresión, al re-ingresar los mismos datos de prueba y al cotejar con los estándares de codificación).
- ✓ Mayor consistencia y repetición (por ejemplo, pruebas ejecutadas por una herramienta y pruebas derivadas de requisitos).
- ✓ Evaluación objetiva (por ejemplo, medidas estáticas, comportamiento del sistema y de la cobertura).
- ✓ Facilidad de acceso a la información sobre las pruebas (por ejemplo, estadísticas y gráficas sobre el progreso de la prueba, índices de falla y rendimiento).

Los riesgos de usar herramientas incluyen:

- ✓ Expectativas poco realistas para la herramienta (incluyendo funcionalidad y facilidad de uso).
- ✓ Subestimar el tiempo, costo y esfuerzo para la introducción inicial de una herramienta (incluyendo entrenamiento y pericia externa).
- ✓ Subestimar el tiempo y el esfuerzo necesitado para alcanzar los beneficios significativos y continuos de la herramienta (incluyendo la necesidad de cambios en el proceso de pruebas y la mejora continua de la forma en que la herramienta es usada).
- ✓ Subestimar el esfuerzo requerido para mantener los recursos de prueba generados por la herramienta.
- ✓ Sobre-dependencia en la herramienta (reemplazo para el diseño de prueba o donde las pruebas manuales serían mejores).

6.5.3 Consideraciones especiales para algunos tipos de herramientas

6.5.3.1 Herramientas de ejecución de prueba

Las herramientas de ejecución de prueba repiten los scripts diseñados para implementar las pruebas que están almacenadas electrónicamente.
Este tipo de herramienta a menudo requiere esfuerzo significativo para alcanzar los beneficios significativos.

Las pruebas de captura al grabar las acciones de un probador manual parecen atractivas, pero este enfoque no se adapta a grandes números de pruebas automatizadas. Un script capturado es una representación lineal con datos específicos y acciones como parte de cada script. Este tipo de script puede ser inestable cuando ocurren eventos no esperados.

Un enfoque guiado por datos filtra las entradas de prueba (los datos), usualmente en una hoja de cálculo, y usa un script más genérico que puede leer los datos de prueba y realiza la misma prueba con datos diferentes. Los probadores que no están familiarizados con el lenguaje de scripting pueden ingresa los datos de prueba para estos scripts predefinidos. En un enfoque guiado por palabras clave, la hoja de cálculo contiene palabras clave que describen las acciones a ser tomadas (también llamadas palabras de acción), y los datos de prueba. Los probadores (incluso si no están familiarizados con el lenguaje de scripting) pueden entonces definir las pruebas usando las palabras clave, las cuales pueden ser adecuadas a la aplicación que está siendo probada.

La pericia técnica en el lenguaje de scripting es necesitada para todos los enfoques (ya sea por probadores o por especialistas en automatización de pruebas).

Independientemente de la técnica de scripting usada, los resultados esperados para cada prueba necesitan ser guardados para posterior comparación.

6.5.3.2 Herramientas de pruebas de rendimiento

Las herramientas de pruebas de rendimiento necesitan a alguien con pericia en las pruebas de rendimiento para que ayude a diseñar las pruebas e intérprete los resultados.

6.5.3.3 Herramientas de análisis estático

Las herramientas de análisis estático aplicadas al código fuente pueden imponer los estándares de codificación, pero si son aplicadas al código existente pueden generar muchos mensajes. Los mensajes de advertencia no paran que el código sea traducido a un programa ejecutable, pero deben ser idealmente tratados para que el mantenimiento del código sea más fácil en el futuro. Una implementación gradual con filtros iniciales para excluir algunos mensajes sería un enfoque efectivo.

6.5.3.4 Herramientas de gestión de prueba

Las herramientas de gestión de prueba necesitan hacer interfaz con otras herramientas u hojas de cálculo para producir información en el mejor formato para las necesidades actuales de la organización. Los informes necesitan ser diseñados y monitoreados para que proporcionen beneficio.

6.6 Introduciendo una herramienta a una organización

6.6.1 Introducción

Los principios principales de introducir una herramienta a una organización incluyen:

- ✓ Evaluación de la madurez organizacional, ventajas y desventajas, e identificación de las oportunidades para un proceso mejorado de prueba soportado por herramientas.
- ✓ Evaluación contra los requisitos claros y los criterios objetivos.
- ✓ Una prueba de concepto para probar la funcionalidad requerida y determinar si los productos cumplen sus objetivos.
- ✓ Evaluación del proveedor (incluyendo entrenamiento, soporte y aspectos comerciales).
- ✓ Identificación de los requisitos internos para la preparación y tutoría en el uso de la herramienta.

La prueba de concepto podría ser realizada en un proyecto piloto de pequeña escala, haciendo posible minimizar los impactos si se encontraron mayores impedimentos y si el piloto no fue exitoso.

Un proyecto piloto tiene los siguientes objetivos:

- ✓ Aprender más detalles sobre la herramienta.
- ✓ Ver como la herramienta se adecuaría a las prácticas y procesos existentes, y como ellas necesitarían cambiar.
- ✓ Decidir sobre las formas estándar de usar, gestar, almacenar y mantener la herramienta y los recursos de prueba (por ejemplo, decidir sobre nombrar convenciones para archivos y pruebas, crear librearías y definir la modularidad de los conjuntos de prueba).
- ✓ Evaluar si los beneficios serán alcanzados a un costo razonable.

Los factores de éxito para el despliegue de la herramienta dentro de una organización incluyen:

- ✓ Presentar la herramienta al resto de la organización en forma progresiva.
- ✓ Adaptar y mejorar los procesos para adecuarse con el uso de la herramienta.
- ✓ Proporcionar entrenamiento y preparación/tutoría para usuarios nuevos.
- ✓ Definir las guías de utilización.
- ✓ Implementar una forma de aprender lecciones del uso de la herramienta.
- ✓ Monitorear el uso y los beneficios de la herramienta.

Referencias

6.2.2 Buwalda, 2001, Fewster, 1999

6.3 Fewster, 1999

7 Referencias

7.1 Estándares

[CMMI] Chrissis, M.B., Konrad, M. and Shrum, S. (2004) CMMI, Guidelines for Process Integration and Product Improvement, Addison
Wesley: Reading, MA

[IEEE 829] IEEE Std 829™ (1998/2005) IEEE Standard for Software Test Documentation (actualmente bajo revisión)

[IEEE 1028] IEEE Std 1028™ (1997) IEEE Standard for Software Reviews

[IEEE 12207] IEEE 12207/ISO/IEC 12207-1996, Software life cycle processes

[ISO 9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality

7.2 Libros

[Beizer, 1990] Beizer, B. (1990) Software Testing Techniques (2da edición), Van Nostrand Reinhold: Boston

[Black, 2001] Black, R. (2001) Managing the Testing Process (2da edición), John Wiley & Sons: New York

[Buwalda, 2001] Buwalda, H. et al. (2001) Integrated Test Design and Automation, Addison Wesley: Reading, MA

[Copeland, 2004] Copeland, L. (2004) A Practitioner's Guide to Software Test Design, Artech House: Norwood, MA

[Craig, 2002] Craig, Rick D. and Jaskiel, Stefan P. (2002) Systematic Software Testing, Artech House: Norwood, MA

[Fewster, 1999] Fewster, M. and Graham, D. (1999) Software Test Automation, Addison Wesley: Reading, MA

[Gilb, 1993]: Gilb, Tom and Graham, Dorothy (1993) Software Inspection, Addison Wesley: Reading, MA

[Hetzel, 1988] Hetzel, W. (1988) Complete Guide to Software Testing, QED: Wellesley, MA

[Kaner, 2002] Kaner, C., Bach, J. and Pettticord, B. (2002) Lessons Learned in Software Testing, John Wiley & Sons:

[Myers 1979] Myers, Glenford J. (1979) The Art of Software Testing, John Wiley & Sons:

[van Veenendaal, 2004] van Veenendaal, E. (ed.) (2004) The Testing Practitioner (Capítulos 6, 8, 10), UTN Publishers: The Netherlands