**Capitulo 14 - Técnicas de Prueba de Software**

**Características:**

* Ingeniería destructiva
* Plan de ejecución
* Utilidad medida en errores
* Objetivo: Error (éxito)
  + Menor tiempo y esfuerzo
* Existencia de defectos en el software

**Principios**

* A todas las pruebas se les debería poder hacer un seguimiento hasta los requisitos del cliente
* Las pruebas deberían planificarse mucho antes de la generación de código.
* El principio de Pareto es aplicable a la prueba del software
  + 80% errores se encuentra en un 20% de los módulos del programa
* Las pruebas deberían empezar por «lo pequeño» y progresar hacia «lo grande»
* No son posibles las pruebas exhaustivas.
* Para ser más eficaces, las pruebas deberían ser realizadas por un equipo independiente

**Fundamentos de las pruebas del software**

**Facilidad de prueba:** Es la facilidad con la que se puede probar un programa de computadora. Características que propician la creación de que el software tenga facilidad de prueba:

***Operatividad:*** «Cuanto mejor funcione, más eficientemente se puede probar.»

***Observabilidad:*** «Lo que ves es lo que pruebas.»

***Controlabilidad:*** «Cuanto mejor podamos controlar el software, más se puede automatizar y optimizar.»

***Capacidad de descomposición:*** «Controlando el ámbito de las pruebas, podemos aislar más rápidamente los problemas y llevar a cabo mejores pruebas de regresión.»

***Simplicidad:*** «Cuanto menos haya que probar, más rápidamente podremos probarlo.»

***Estabilidad:*** «Cuanto menos cambios, menos interrupciones a las pruebas.»

***Facilidad de comprensión:*** «Cuanta más información tengamos, más inteligentes serán las pruebas.»

**Características de la prueba:** Atributos para una buena prueba:

1. Alta probabilidad de encontrar un error
2. No redundante
3. Heurística
4. Balance entre sencillez y complejidad

**Pruebas de Caja negra y Caja Blanca**

Cualquier producto de ingeniería puede probarse según

* Función específica para la que se diseño el producto. Examina algún aspecto funcional de un sistema **->** Prueba de caja negra
* Funcionamiento interno del producto según la especificación acertada de las funciones y la interacción de los componentes. Detalle procedimental **->** Prueba de caja blanca

**Prueba de Caja Blanca**

Utiliza la estructura de control descrita como parte del diseño al nivel de componentes para derivar los casos de prueba que:

* Ejecuten todos los *caminos* lógicos al menos una vez.
* Ejerciten ambas variantes (verdaderas como falsas) de una decisión.
* Ejecuten todos los bucles en sus límites.
* Ejerciten las estructuras internas de datos.

**¿Por qué utilizar esta técnica de prueba?**

* Los errores lógicos y las suposiciones incorrectas son inversamente proporcionales a la probabilidad de que se ejecute un camino del programa.
* El flujo de datos no siempre es intuitivo sobre el flujo de control.
* Los errores tipográficos tienen la misma probabilidad de encontrarse en un camino de “curso normal” que en casos lógicamente excepcionales.

**Prueba de la ruta básica**

Es una técnica de prueba de caja blanca que permite al diseñador de casos de prueba:

* Obtener la medida de complejidad lógica de un diseño procedimental.
* Asegurar la ejecución de cada sentencia al menos una vez.

**Complejidad ciclomática:** Determina la medición cuantitativa de complejidad lógica de un programa. Define el número de Caminos independientes en el conjunto básico del programa. Establece un límite máximo de pruebas para ejecutar todos los caminos lógicos.

Pasos para derivar el conjunto básico:

1. Utilizando el diseño o el código como base se dibuja la grafica de flujo correspondiente.
2. Determinar la complejidad ciclomatica de la grafica de flujo resultante
3. Determinar un conjunto básico de rutas linealmente independientes.
4. Preparar los casos de prueba que forzaran la ejecución de cada ruta en el conjunto básico

**Matrices de gráficos:** Procedimiento mecanizado para determinar el conjunto de caminos posibles**.**

Una matriz de grafica es una matriz cuadrada cuyo tamaño es igual a la cantidad de nodos, y sus entradas son las Conexiones entre nodos. Contiene un peso de enlace que proporciona información adicional acerca del flujo de control y contiene las siguientes propiedades:

* La probabilidad de que se ejecute un enlace.
* El tiempo de procesamiento gastado, la memoria y los recursos requeridos durante el recorrido de un enlace.

**Prueba de la Estructura de Control**

Si bien la ruta básica es simple y efectiva, no es suficiente. Hay diferentes variaciones sobre la prueba de estructuras de control.

**Prueba de Condición**

Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en un modulo del programa. Se focaliza en la prueba de cada condición del programa para asegurar que no contiene errores.

**Prueba del flujo de datos**

Selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa (cadenas definición-uso). La estrategia de prueba determina que al menos una vez se cubra cada cadena definición-uso.

**Prueba de bucles**

Es una técnica de prueba de caja blanca que se concentra en la validez en la construcción de bucles.

**Bucles simples:** Se asegura encontrar errores tanto en la ausencia de su ejecución como en el número máximo posible de ejecuciones.

**Bucles anidados:** Se sugiere un enfoque que reduce el número de pruebas: iniciando desde el bucle mas interno asignándole valores mínimos a los demás bucles. Agregar otras pruebas para los valores fuera de rango o excluidos. Además, se debe trabajar hacia afuera manteniendo los bucles externos en valores mínimos y otros bucles anidados en valores “típicos”. Se finaliza al probar todos los bucles del ciclo.

**Bucles concatenados:** Se prueban empleando los enfoques de bucles simples y anidados.

**Bucle no estructurado:** Se debería buscar la manera de rediseñar estos bucles para ajustarlos al método de construcción estructurado.

**Prueba de Caja Negra**

Se concentra en los requisitos funcionales del software. Se aplican durante las últimas etapas de la prueba. Tratan de encontrar errores en las siguientes categorías:

* Funciones incorrectas
* Errores de interfaz
* Errores de estructuras de datos o accesos a bases de datos externas
* Errores de rendimiento
* Errores de curso de ejecución (inicialización y finalización)

**Métodos gráficos de prueba**

Su objetivo es verificar que todos los objetos tienen entre ellos las relaciones esperadas.

**PARA MAS INFORMACION VER LA PAGINA 435.**

**Partición equivalente**

Es un método de prueba de caja negra que Realiza una división de un campo de entrada de un programa en clases de datos de las cuales pueden derivarse casos de prueba.

Se esfuerza para definir un caso de prueba que descubra ciertas clases de errores, reduciendo el número de caos de prueba que deben desarrollarse.

Clases de equivalencia: Representa un conjunto de estados validos y no validos para las condiciones de entrada, éstas pueden ser: valores numéricos, Rango de valores, condición lógica, entre otras.

Directrices:

* Si una condición de entrada especifica un rango o un valor especifico, se definen una clase de equivalencia valida y dos no validas.
* Si una condición de entrada especifica un miembro de un conjunto o es booleana, se definen una clase de equivalencia valida y otra no valida.

**Análisis de Valores Límites**

Los errores tienden a darse con mayor frecuencia en los límites de campo de entrada que en valores “promedio” por eso se ha desarrollado el AVL como técnica de prueba. Este análisis complementa al método de partición equivalente. Contempla casos de uso tanto para los campos de entrada como para los campos de salida.

EL AVL extiende la partición equivalente al concentrase en los datos de las “aristas” (flujos de control) de una clase de equivalencia. Directrices similares a las de partición equivalente.

**Prueba de Tabla Octogonal**

Se aplica a problemas en que el dominio de entrada es relativamente pequeño pero demasiado grande para posibilitar prueba exhaustiva. Es útil para encontrar errores asociados con las fallas de región (errores asociados a defectos de la lógica en un componente de software).

Detecta y aísla los fallos de:

* Modalidad simple: Errores en valores de 1 solo parámetro
* Modalidad doble: Error en 2 parámetros que intervienen conjuntamente por inconsistencia o la imposibilidad de interacción de los mismos
* Multimodales

**Métodos de pruebas orientadas a objetos**

**Pruebas basada en fallas:** se diseñan pruebas que tengan una alta probabilidad de descubrir posibles fallas. Por lo general comienza en el modelo de análisis. Las pruebas de integración buscan fallas en llamadas a operación o en conexiones entre mensajes. Se encuentran 3 tipos de fallas en este contexto: resultado inesperado, operación incorrecta/mensaje empleado, invocación incorrecta.

**Casos de prueba y jerarquía de clase:** Aunque se haya probado por completo una clase de base, aun se tendrá que probar todas las clases derivadas de ella.

**Prueba basada en escenarios:** Se concentra en lo que hace el usuario, no al producto. Descubrirá errores que ocurren cuando cualquier actor interactúa con el software.

**Estructuras de superficie y de fondo en pruebas:** la prueba de estructura de superficie es análoga a la prueba de caja negra. La de estructura de fondo es similar a la de caja blanca.

**Pruebas de Entornos Especializados, Arquitecturas y Aplicaciones**

**Pruebas de interfaces gráficas de usuario**

* Se puede utilizar el método de grafo de estado finito para derivar una serie de pruebas que ejerciten objetos específicos de datos y programas que resultan relevantes para la GUI
* Existen herramientas automáticas para determinar fallas en la interfaz.

**Pruebas de arquitectura cliente-servidor**

Los siguientes aspectos se combinan para que la prueba de las arquitecturas de software C/S resulte considerablemente más difícil que la prueba de aplicaciones independientes:

* + Rendimiento
  + Presencia de diferentes plataformas de hardware
  + Complejidad de Comunicaciones de red
  + Concurrencia de clientes
  + Coordinación entre el Cliente y el servidor

La prueba del software C/S se presenta en 3 niveles:

* Aplicaciones de cliente individuales se prueban en una modalidad “desconectada”
* El software de cliente y las aplicaciones asociadas del servidor se prueban en conjunto, pero las operaciones de red no se ejercitan de manera explicita
* Se prueba toda la arquitectura C/S, incluida la operación y el desempeño de la red.

Pruebas para aplicaciones C/S:

* De funcionalidad de la aplicación
* De servidor
* De base de datos
* De transacción
* De comunicaciones de red

**Prueba de la documentación y las funciones de ayuda**

Incide considerablemente en la aceptación del programa.

Se aborda en dos fases:

* Revisión e inspección: se examina la claridad editorial del documento.
* Prueba en vivo: se emplea la documentación junto con el programa real.

**Prueba de sistemas de tiempo real**

El diseñador de casos de prueba deberá considerar:

* Procesamiento de interrupciones
* Paralelismo de tareas
* Temporización de datos
* Impacto en las fallas del hardware

La estrategia efectiva para la prueba de un sistema en tiempo real consiste en 4 pasos:

* **Prueba de tareas**
  + Se prueba cada tarea independientemente.
  + Descubre errores de lógica y funcionamiento.
* **Prueba de Comportamiento**
  + Simulaciones de comportamiento y examinarlo como consecuencia de sucesos externos.
  + Se categorizan las clases de sucesos para determinar fallas.
  + Se realizan en orden y frecuencia aleatoria.
* **Prueba intertareas** 
  + Se focaliza en los errores relativos al tiempo.
  + Se prueban tareas asíncronas comunicadas con diferentes tasas de datos
  + Se prueban colas de mensajes y almacenes de datos para descubrir errores de tamaño.
* **Prueba de sistema**
  + Interfaz software/hardware
    - Estados de transición y especificaciones de control.
    - Listado posibles de pares interrupciones – acciones.

**Patrones de prueba**

Describen bloques de construcción o situaciones frecuentes y que los responsables de probar el software pueden reutilizar al afrontar la prueba de algún sistema nuevo o revisado.

Proporcionan:

* Terminología
* Mayor comprensión en las soluciones
* Estimula el razonamiento iterativo