**Capitulo 23 - Pruebas orientadas a objetos**

Objetivo general: Encontrar el *mayor número de errores* con una cantidad *razonable de esfuerzo* aplicado sobre un *tiempo realista. Intentar “romper” el software*

PRUEBAS ORIENTADAS A OBJETOS

* Se cambia la **estrategia** y las **tácticas** de prueba
* Al madurar el Análisis orientado a objetos (AOO) , el diseño orientado a objetos (DOO) y existir mayor reutilización, es un nuevo contexto, por lo tanto se **deben repetir las pruebas**

**¿Que hace referencia una prueba orientada a objetos (OO)?**

Probar un sistema OO para descubrir errores que ocurren cuando las clases colaboran entre sí y los subsistemas se comunican por medio de las capas arquitectónicas

**¿Quién son los encargados de realizar las pruebas OO?**

Los ingenieros de software y especialistas en pruebas

**¿Por qué es importante?**

Es necesario ejecutar el sistema antes de que llegue al cliente con la intención de descubrir errores

**¿Cuáles son los pasos a seguir?**

Similar a las pruebas convencionales pero **tácticamente** diferente, se comienza en lo “pequeño” (pruebas de clase) y luego cuando las clases colaboran entre sí

**¿Que se obtiene al realizar las pruebas?**

Se *diseñan* y *documentan* casos de pruebas para probar clases, colaboraciones y comportamiento, se **definen resultados esperados** y se registran los **resultados reales**

**¿Cómo nos aseguramos que la prueba se realizo correctamente?**

Diseñar casos de prueba de forma disciplinada con el objetivo de intentar **“romper” el software** y revisarlos con meticulosidad

**Pruebas de los modelos de AOO y DOO**

Los modelos de análisis y diseño no pueden probarse convencionalmente ya que no se pueden ejecutar, sin embargo para garantizar la calidad se pueden utilizar revisiones técnicas formales para examinar la exactitud y consistencia.

**Exactitud de los modelos AOO y DOO**

La notación y sintaxis elegida para representar los modelos de análisis y diseño deben corresponderse con el método específico de análisis y diseño elegido para el proyecto. Cada modelo se revisa para asegurarse que se mantuvieron las convenciones propias del modelado

**Consistencia de los modelos AOO y DOO**

Un modelo inconsistente tiene representaciones en una parte que no se reflejan correctamente en otras partes del modelo. Para evaluar la consistencia, se examina cada clase y sus conexiones a otras clases. Pueden utilizarse modelo clase responsabilidad colaboración (CRC) y diagrama objeto relación.

**Modelo clase responsabilidad colaboración (CRC):** Se compone de: nombre de clase, responsabilidad (operaciones) y colaboradores (otras clases a las que se envían mensajes).

**Modelo objeto relación:** Proporciona una representación gráfica de conexiones entre clases.

Para evaluar el modelo de clases se recomiendan los siguientes pasos:

1-Revisar el modelo CRC y el modelo objeto-relación.

2-Inspeccionar la descripción de cada tarjeta CRC.

3-Invertir la conexión para asegurarse de que cada colaborador que solicita un servicio recibe las peticiones de una fuente razonable.

4-Determinar si otras clases se requieren y si las responsabilidades se han repartido adecuadamente entre las clases.

5-Determinar si las responsabilidades muy solicitadas, deben combinarse en una sola responsabilidad.

6-Se aplican iterativamente los pasos 1 a 5 para cada clase.

El diseño de sistema se revisa examinando el modelo objeto-comportamiento desarrollado durante el AOO así como también la concurrencia y la asignación de tareas. Las especificaciones detallada de las operaciones se revisan usando técnicas de inspección convencionales.

**Estrategias de pruebas OO**

La estrategia clásica para la prueba de software de ordenador, comienza por probar lo pequeño y funciona hacia fuera haciendo probar lo grande.

**Pruebas de Unidad en el contexto de la OO**

Concepto de unidad cambia, la encapsulación, define clases y objetos, cada clase y cada objeto(instancia de clase), envuelven atributos y operaciones(métodos).

Software convencional, unidad más pequeña: **módulo**

Sofware OO, unidad más pequeña: **Clase u objeto**

Pruebas de unidad en software convencional equivale a las pruebas de clase en el software OO

Las pruebas de clases para el software OO se conducen mediante operaciones encapsuladas por la clase y el comportamiento de la clase

**Pruebas de integración en el contexto OO**

Existen dos estrategias diferentes para las pruebas de integración de los sistemas OO:

*Pruebas basadas en hilos:* Integran el conjunto de clases requeridas para responder una entrada o suceso al sistema. Cada hilo se integra y prueba individualmente.

*Pruebas de regresión:* Aseguran que no ocurran efectos laterales.

*Pruebas basadas en el uso:* Se comienza la construcción del sistema probando aquellas clases, que utilizan pocas o ninguna clase servidoras. Después de probar las clases independientes se prueban las clases dependientes hasta el sistema completo.

*Prueba de agrupamiento:* Se agrupan clases colaboradoras y se prueban diseñando casos de prueba que intentan revelar errores de las colaboraciones.

**Pruebas de validación en el contexto OO**

Al igual que el software convencional se centra en las acciones visibles al usuario y salidas reconocibles desde el sistema. El probador debe utilizar los casos de uso que son parte del modelo de análisis.

Pueden utilizarse los métodos convencionales de prueba de caja negra para realizar las pruebas de validación.

**Diseños de casos de prueba para software OO**

1. Cada caso de prueba debe ser identificado separadamente y explícitamente asociado a la clase que se va a probar
2. Debe declararse el propósito de la prueba
3. Debe desarrollarse una lista de pasos a seguir: definir una lista de estados, una de mensajes y operaciones, de excepciones, de condiciones externas e información adicional.

A diferencia de pruebas convencionales (entrada-proceso-salida), las pruebas OO se enfocan en las secuencias de operaciones de diseño apropiadas para probar los estados de una clase

**Implicaciones de los conceptos de OO al diseño de casos de prueba**

La encapsulación puede dificultar la obtención de información. A menos que se proporcionen operaciones incorporadas para conocer los valores para los atributos de la clase.

La herencia conduce a retos adicionales, cada nuevo contexto de uso requiere repetir la prueba, aun cuando se haya logrado la reutilización. Además, la herencia múltiple complica mucho más las pruebas, incrementando el número de contextos para los que se requiere la prueba.

**Pruebas basadas en errores**

Objetivo: Diseñar pruebas que tengan una alta probabilidad de revelar fallos. El producto (sistema) debe adaptarse a los requerimientos del cliente, por lo tanto la planificación preliminar requerida para llevar a cabo la prueba comienza con el modelo de análisis. Los casos de prueba se diseñan para probar el diseño o código.

Pruebas de integración buscan fallos probables en operaciones o mensajes de conexión. Además, intentan encontrar errores en el objeto *cliente* no en el servidor. Determinar si el error existe en el código de invocación no en el invocado. Por último, se aplican tanto en los atributos como en las operaciones.

**Impacto de la programación OO en las pruebas**

* Algunos fallos se vuelven menos probables (no vale la pena probar)
* Algunos fallos se vuelven más probables (vale la pena probar)
* Aparecen nuevos tipos de fallos.
* Los errores de integración se vuelven más probables.

**Diseño de pruebas basadas en el escenario**

Las pruebas basadas en los errores no localizan dos tipos de errores:

**Especificaciones incorrectas**: El producto no realiza lo que el cliente desea (cosas incorrectas u omitir funcionalidad)

**Interacción entre subsistemas:** El comportamiento de un subsistema crea sucesos, flujo de datos que hace que el otro subsistema falle.

Pruebas basadas en el escenario se centran en lo que el usuario hace, no en lo que el producto hace. Además, revelan errores de interacción y tienden a validar subsistemas en una prueba sencilla.

**Pruebas superficiales y profundas**

Estructura superficial: Se refiere a lo visible al exterior del programa, estructura que es obvia al usuario final. Las pruebas se basan en las tareas de los usuarios, capturar las tareas, involucra comprensión, observación y conversar con usuarios representativos y no representativos.

Estructura profunda: Se refiere a los detalles técnicos de un programa OO. Estructura que se comprende examinando el diseño y/o código.

**Métodos de prueba aplicables a nivel de clase**

Pruebas de software comienzan en lo “pequeño” (una sola clase y métodos encapsulados) y luego progresen hacia lo “grande”. La verificación y partición al azar son métodos que pueden usarse para ejercitar a una clase durante una prueba OO.

Verificación al azar: Se general al azar secuencias de operaciones de forma aleatoria para probar diferentes registros de operaciones de instancias de clases

Partición al nivel de clase: Reduce el número de casos de prueba requeridos para validar la clase, de la misma forma que la partición equivalente.

**Partición basada en estados**: Clasifica las operaciones de clase basada en su habilidad de cambiar el estado de la clase.

**Partición basada en atributos:** clasifica las operaciones de clase basada en los atributos que ellas usan.

**Partición basa en categorías:** clasifica las operaciones de la clase basadas en la función genérica que cada una lleva a cabo.

**Diseño de casos de prueba interclases**

* Comienza al realizarse la integración del sistema
* Es la etapa en que se realiza la verificación de colaboraciones entre clases

**Pruebas derivadas de los modelos de comportamiento**

* El DTE (diagrama de transición de estados) para una clase puede usarse para ayudar a derivar una secuencia de pruebas que ejercitarán el comportamiento dinámico de la clase y aquellas clases que colaboran con ella.
* Las pruebas a diseñarse deben alcanzar una cobertura de todos los estados.
* Se pueden derivar casos de prueba para asegurarse que todos los comportamientos para la clase han sido adecuadamente ejercitados
* Si el comportamiento de una clase resulta de una colaboración con una o más clases, se utilizan múltiples DTE para registrar el comportamiento del sistema.