TEMA 4.2. Prueba de Aplicaciones OO

Calidad del Software

Dr. José Luis Abellán Miguel

Grado en Ingeniería Informática

Dr. José Luis Abellán Miguel - Tlf: (+34) 968 278 657
Universidad Católica San Antonio de Murcia www.ucam.edu

Índice

- Introducción
- Modelos de Prueba AOO y DOO
- Estrategias de Pruebas OO
- Métodos de Prueba OO
- Métodos de Prueba Aplicables en el Nivel de Clase

Bibliografía

- Pressman, R. ***Ingeniería del Software: Un enfoque práctico***. 7ª edición. Madrid: McGraw Hill, 2010. ISBN: 9701054733 (disponible en la biblioteca UCAM) → **Capítulo 19**

Introducción (1/3)

- La naturaleza de los programas OO cambia en la estrategia y en las tácticas de las pruebas.
- Para probar adecuadamente los sistemas OO, deben realizarse tres cosas:
  - Ampliar la definición de prueba para incluir las técnicas de descubrimiento de error aplicadas al análisis orientado a objetos y a modelos de diseño
  - Cambiar significativamente la estrategia para prueba de unidad e integración
  - Explicar las características únicas del software OO mediante el diseño de casos de prueba

Introducción (2/3)

- Es necesario probar un sistema OO en varios niveles diferentes:
  - Descubrir errores que puedan ocurrir conforme las clases colaboran unas con otras y conforme los subsistemas se comunican a través de capas arquitectónicas
- "De lo pequeño hacia lo grande":
  1. **Pruebas de clase:** pruebas que ejercitan las operaciones de clase y que examinan si existen errores conforme una clase colabora con otras clases
  2. **Pruebas de integración:** las clases se integran para formar un subsistema, se aplican **pruebas de hebra, de uso y de grupo** mediante enfoques basados en fallo para ejercitar por completo clases colaboradoras
  3. **Pruebas de validación:** Finalmente, se usan casos de uso (desarrollados como parte del modelo de requerimientos) para descubrir errores de validación del software

Introducción (3/3)

- La construcción de software orientado a objetos:
  - Revisar desde los modelos de análisis, diseño hasta la codificación (definición de modelos detallados de clases, relaciones entre ellas, diseño de objetos, etc).
- Ejemplo: Atributo incorrecto de una clase
  - Detección en fase de análisis → Evitará:
    - Subclases para alojar el atributo innecesario o las excepciones.
    - Relaciones de clase incorrectas o extrañas.
    - Caracterización incorrecta del comportamiento del sistema
  - Propagación a fase de diseño:
    - Asignación inadecuada de la clase a algunas tareas
    - Procedimientos para las operaciones sobre el atributo extraño
    - Modelo de mensajería será incorrecto
  - Propagación a la fase de codificación!!

Modelos de Prueba AOO y DOO (1/2)

- Los modelos de análisis (AOO) y diseño (DOO) no pueden probarse de la manera convencional porque no pueden ejecutarse
  - Revisiones técnicas para examinar su exactitud y consistencia

1. Exactitud de los modelos AOO y DOO

- Exactitud sintáctica: desde la representación de los modelos de análisis y diseño a los métodos de análisis y diseño específicos que se elijan en el proyecto (convenciones de modelado adecuadas).
- Exactitud semántica: conformidad del modelo con el dominio de problemas del mundo real: expertos de dominios de problemas, quienes examinarán las definiciones y jerarquía de clase en busca de omisiones y ambigüedad

Modelos de Prueba AOO y DOO (2/2)

2. Consistencia de los modelos OO

- Examinarse cada clase y sus conexiones con otras clases
  → Modelo clase-responsabilidad-colaboración (CRC)

nombre clase: credit sale	
tipo clase: evento transacción	
características clase: no tangible, atómica, secuencial, permanente, guardada	
responsabilidades:	colaboradores:
leer tarjeta crédito	tarjeta crédito
obtener autorización	autoridad crédito
cantidad postcompra	comprobante producto
	libro ventas
	archivo auditoría
generar factura	factura

Estrategias de Pruebas OO (1/3)

Prueba de Unidad

- Cada clase y cada instancia de una clase (objeto) encapsulan los atributos (datos) y las operaciones (también conocidas como métodos o servicios) que manipulan dichos datos
  - La unidad comprobable más pequeña es la clase encapsulada (Prueba de Unidad como Prueba de Clase)
    - Ya no es posible probar una sola operación aislada sino como parte de una clase
    - **La prueba de clase** para el software OO se activa mediante las operaciones encapsuladas por la clase y por el comportamiento de estado de la misma

Estrategias de Pruebas OO (2/3)

Prueba de Integración

- El software orientado a objetos no tiene una estructura de control jerárquica, las estrategias de integración tradicionales, descendente y ascendente no sirven
  - Prueba basada en **hebra**: conjunto de clases requeridas para responder a una entrada o evento del sistema
    - Cada hebra se integra y prueba de manera individual
  - Prueba basada en **uso**: Pruebas independientes → Dependientes (usan las independientes)→ Dependientes de las dependientes→...→Todo el sistema
  - Prueba de **grupo**: se ejercita un grupo de clases colaboradoras examinando el modelo CRC
    - Descubrir errores en las colaboraciones

Estrategias de Pruebas OO (3/3)

Prueba de Validación (Sistema)

- Acciones visibles para el usuario y en las salidas del sistema reconocibles por él mismo:
  - Casos de uso del modelo de requerimientos
    - Escenario que tiene una alta probabilidad de descubrir errores en los requerimientos de interacción de usuario
  - Pruebas de caja negra
  - Casos de prueba del modelo de comportamiento del objeto
  - Diagrama de flujo de evento creado como parte del AOO

Métodos de Prueba OO (1/6)

- Es necesario probar un sistema OO en varios niveles
  - Descubrir errores que puedan ocurrir conforme las clases colaboran y los subsistemas se comunican a través de las capas arquitectónicas
- Enfoque global en el diseño de casos de prueba OO (Berard'93)
  1. Cada caso de prueba debe identificarse de manera única asociado con la clase que se va a probar y establecer el propósito de la prueba
  2. Desarrollar una lista de pasos de prueba para cada una de ellas:
     - Una lista de estados especificados para la clase que se probará
     - Una lista de mensajes y operaciones que se ejercitarán como consecuencia de la prueba
     - Una lista de excepciones que pueden ocurrir conforme se prueba la clase
     - Una lista de condiciones externas (cambios en el entorno)
     - Información complementaria que ayudará a implementar la prueba
  - Se enfoca en el diseño de secuencias apropiadas de operaciones para ejercitar los estados de una clase.

Métodos de Prueba OO (2/6)

- **Implicaciones del diseño de casos de prueba de conceptos OO**
  - Los atributos y las operaciones de una clase están encapsulados → Es improductivo probar operaciones afuera de la clase.
    - Puede ser difícil adquirir una instantánea del estado de un objeto → Escribir operaciones internas que reporten los valores para los atributos de clase
  - El uso de herencia requiere que cada nuevo contexto de uso requiera nuevas pruebas (la superclase puede estar en un contexto de uso diferente a las subclases)
    - La herencia múltiple complica más la prueba por originar más contextos para la prueba

Métodos de Prueba OO (3/6)

- **Aplicabilidad de los métodos convencionales de diseño de casos de prueba**
  - Los métodos de prueba de caja blanca pueden aplicarse a las operaciones definidas para una clase
    - Las técnicas de ruta básica ayudar a garantizar que se probaron todos los enunciados en una operación
    - El esfuerzo aplicado a la prueba de caja blanca puede redirigirse mejor para probar en un nivel de clase
  - Los métodos de prueba de caja negra son apropiados para los sistemas OO
    - Los casos de uso pueden facilitar el diseño de las pruebas de caja negra

Métodos de Prueba OO (4/6)

- **Prueba basada en fallo**
  - Diseñar pruebas que tengan una alta probabilidad de descubrir fallos plausibles
  - **Fallos plausibles**: aspectos de la implementación del sistema que pueden resultar en defectos
    - Entendimiento de los modelos AOO y DOO
    - Los casos de prueba se diseñan a fin de ejercitar el diseño o el código
    - La prueba de integración busca fallos plausibles en operaciones y en las conexiones de mensaje.

      Resultado inesperado, uso de operación/mensaje equivocado e invocación incorrecta.

      Se prueba la clase cliente no la servidor (errores en el código que llama, no en el código llamado)

Métodos de Prueba OO (5/6)

- **Diseño de pruebas basadas en escenario (1/3)**
- Las pruebas basadas en fallo pierden dos tipos principales de errores:
  - Especificaciones incorrectas
    - El producto no hace lo que el cliente quiere. Puede hacer lo correcto u omitir funcionalidad importante
  - Interacciones entre subsistemas
    - El comportamiento de un subsistema crea circunstancias (por ejemplo, eventos, flujo de datos) que hacen que otro subsistema falle
  - **La prueba basada en escenario se concentra en lo que hace el usuario, no en lo que hace el producto.**

Métodos de Prueba OO (5/6)

- **Diseño de pruebas basadas en escenario (2/3)**
  - **La prueba basada en escenario se concentra en lo que hace el usuario, no en lo que hace el producto.**
    - Capturar las tareas (por medio de **casos de uso**) que el usuario tiene que realizar y luego aplicar éstas y sus variantes como pruebas
    - Descubren errores de interacción (casos de prueba más complejos y realistas que las pruebas basadas en fallo)

      Tiende a ejercitar múltiples subsistemas en una sola prueba

Métodos de Prueba OO (5/6)

❑ Diseño de pruebas basadas en escenario (3/3)

◘ Ejemplo: diseño de pruebas basadas en escenario para un editor de texto

- **Caso de uso: corrección del borrador final**
  - **Antecedentes: I**mprimir el borrador “final”, leerlo y descubrir algunos errores que no se vieron en la imagen de la pantalla
    1. Imprimir todo el documento.
    2. Moverse en el documento, cambiar ciertas páginas.
    3. Conforme cada página cambia, imprimirla.
    4. En ocasiones se imprime una serie de páginas.
- Pruebas basadas en escenario:
  - Imprimir páginas solas , rango de páginas y comprobar coherencia con función de edición

Métodos de Prueba OO (6/6)

- **Pruebas de las estructuras superficial y profunda**
  - Estructura superficial (prueba de caja negra)
    - Estructura observable externamente de un programa OO
    - Las pruebas se basan todavía en tareas de usuario
  - Estructura profunda (prueba de caja blanca)
    - Se hace referencia a los detalles técnicos internos de un programa OO

    La estructura que se comprende al examinar el diseño y/o el código.

    Ejercitar dependencias, comportamientos y mecanismos de comunicación que se establezcan como parte del modelo de diseño para el software OO.

Métodos de Prueba en el Nivel de Clase (1/2)

Prueba aleatoria

- Ejemplo: clase **Account:**

Métodos: *open( )*, *setup( )*, *deposit( )*, *withdraw( )*, *balance( )*, *summaries( )*, *creditLimit( )* y *close( ).*

- Existen restricciones en el orden de aplicación de las operaciones:

  Open → setup → deposit → withdraw → close

- En general podemos definir más comportamientos: open → setup → deposit → [deposit|withdraw|balance|summarize|creditLimit]^*n* →withdraw →close
- Secuencias de operaciones que pueden generarse al azar.
  - *Caso de prueba* 1*:*

open>setup>deposit>deposit>balance>summarize>withdraw>close

  - *Caso de prueba 2:*

open>setup>deposit>withdraw>deposit>balance>creditLimit>withdraw>close

Métodos de Prueba en el Nivel de Clase (2/2)

Prueba de partición (1/2)

- Reduce el número de casos de prueba requeridos para ejercitar la clase
  - Las entradas y salidas se categorizan y los casos de prueba se diseñan para ejercitar cada categoría
- La partición con base en estado
  - Categoriza las operaciones de clase a partir de su capacidad para cambiar el estado de la clase
    - Clase **Account**: operaciones que cambian el estado: *deposit* y *withdraw* y que no lo cambian: *balance*, *sumaries, creditLimit*
    - Las pruebas se diseñan para que ejerciten por separado las operaciones que cambian el estado (p1) y las que no (p2):

Caso de prueba p_1: **open • setup • deposit • deposit • withdraw • withdraw • close**

Caso de prueba p_2: **open • setup • deposit • summarize • creditLimit • withdraw • close**

Métodos de Prueba en el Nivel de Clase (2/2)

Prueba de partición (2/2)

- La partición con base en atributo
  - Categoriza las operaciones de clase con base en los atributos que usan.
  - Clase **Account**: los atributos *balance* y *creditLimit* pueden usarse para definirse tres particiones:
    - Operaciones que usan *creditLimit*
    - Operaciones que modifican *creditLimit*
    - Operaciones que no usan ni modifican *creditLimit* → *balance*
- Partición basada en categoría jerarquiza las operaciones de clase con base en la función genérica que cada una realiza
  - Clase **Account** pueden categorizarse en:
    - Operaciones de inicialización (*open*, *setup*), de cálculo (*deposit, withdraw*), consultas (*balance, summarize, creditLimit*) y de terminación (*close*).