Técnicas de Documentación y Validación

Unidad 4 – V & D de Implementación - Testing de Unidad

Contenidos

- Definiciones
- Niveles de prueba
- tipos de pruebas
- Actores
- Automatización
- Junit
- testNG
- Mocking

Algunas definiciones

"Proceso orientado a demostrar que un programa no tiene errores"

"Tarea de demostrar que un programa realiza las funciones para la cual fue construido"

"Es la tarea de probar que un programa realiza lo que se supone que debe hacer. Aun haciendo lo esperado, puede contener fallas"

"Es la ejecución de programas de software con el objetivo de detectar defectos y fallas"

Test exitoso -> detecta errores

Test NO exitoso -> No detecta errores

Algunas definiciones (cont.)

- Error: Una equivocación humana al desarrollar alguna tarea en una de las etapas del desarrollo.
- Defecto: Manifestación del error
- Falla: Desvío respecto al comportamiento esperado del sistema

¿Por qué realizar pruebas?

- Imagen (confiabilidad, prestigio)
- Costo
  - Requerimientos ($1x)
  - Diseño ($5x)
  - Implementación ($20x)
  - Prueba ($50x)
  - Mantenimiento ($200x)
  - Legales ($Nx)

Test	Objetivo	Participantes	Ambiente	Método
Unitario	Detectar errores en datos, lógica y algoritmia	Programadores	Desarrollo	Caja blanca
Integración	Detectar errores de interfaces y relaciones entre componentes	Programadores / Integradores	Desarrollo	Caja blanca, top down, bottom up
Funcional	Detectar errores en la implementación de los requerimientos	Testers, analistas	Desarrollo	Funcional
Sistema	Detectar fallas en la cobertura de requerimientos	Testers, analistas	Desarrollo	Funcional
Aceptación	Detectar fallas en el despliegue / implantación del sistema	Testers, analistas, cliente	Productivo	Funcional

Tipos de testing

- Test de Humo
- Test de Facilidad
- Test de Volumen
- Test de Stress
- Test de usabilidad
- Test de Seguridad
- Test de Performance

- Test de Configuración
- Test de Compatibilidad
- Test de Instalabilidad
- Test de Confiabilidad
- Test de Recuperación
- Test de Documentación
- Test de Mantenibilidad

¿Por qué automatizar?

- Pruebas manuales largas
- Metabugs. Pruebas propensas a errores
- Ahorro recursos humanos
- Generar conocimiento para desarrollo a partir de los tests
- Detección de manera temprana y frecuente
- Aumentar confiabilidad del ambiente

¿Por qué NO automatizar? Excusas...

- Humanas:
  - Actitud de los programadores
  - Hábitos
  - Temores
- Financieras
  - Inversión inicial
- Técnicas
  - Sistemas legados
  - Evolución del código

¿Qué sí y qué No?

Automatizar

- Pruebas unitarias
- Pruebas de componentes
- Integración con sistemas legados

Manual

- Pruebas de usabilidad
- Pruebas exploratorias
- Pruebas que no fallarán
- Pruebas aisladas y de fácil ejecución manual

Práctica 4A

- Máximo 40’
- Para los requerimientos de la narrativa asignada, identificar tipos de tests
- Describa El objetivo de cada uno

Pruebas unitarias

Introducción a JUnit

¿Qué es JUnit

- Es un framewok que permite realizar la ejecución de clases Java de manera controlada, y de esta forma evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.
- Su flexibilidad lo hacen sumamente útil en pruebas de regresión
- El propio *framework* incluye formas de ver los resultados (*runners*) que pueden ser en modo texto, gráfico (AWT o Swing) o como tarea en Ant.
- IDEs incorporan plug-ins para su rápida utilización

¿Por qué usar JUnit?

- Simple.
- Código Java.
- Herramienta gratuita.
- Fácil de aplicar en un proyecto.
- Tests implementados por el desarrollador.
- Autoevaluación de resultados y feedback inmediato
- Permite escribir código de calidad rápidamente.
- Jerarquía de tests en árbol.
- Software más estable.

JUnit

- Diseño: Patrones Command y Composite
- TestCase (Command). La clase que posee métodos para testear debe extender de TestCase.
- Convención de nombres: TEST_SetProperties() SetPropertiesTEST
- Utilizan:
  - setUp()
  - tearDown()
- TestSuite: Composición en forma de árbol de tests, bien ejemplares de TestCase u otros TestSuites

¿Cómo proceder?

1. Escribir la clase con los Tests
2. Implementar el método setUp().
3. Implementar el método tearDown().
4. Definir métodos TEST_XXX() cómo públicos para probar y verificar el resultado.
5. Implementar una clase suite (sin métodos)
6. Implementar un Runner para la Suite con un método main()

Tutorial 4A

- Analizar la funcionalidad de las clases Programa, Programación y Tarea y la interfaz ejecutable
- Implementar utilizando el archivo tutorial 4A las clases JUnit correspondientes para programa
- Ejecutar las pruebas para analizar el orden de invocación

Asserts

Método assertxxx() de JUnit	Qué comprueba
assertTrue(expresión)	comprueba que expresión evalúe a true
assertFalse(expresión)	comprueba que expresión evalúe a false
assertEquals(esperado,real)	comprueba que esperado sea igual a real
assertNull(objeto)	comprueba que objeto sea null
assertNotNull(objeto)	comprueba que objeto no sea null
assertSame(objeto_esperado,objeto_real)	comprueba que objeto_esperado y objeto_real sean el mismo objeto
assertNotSame(objeto_esperado,objeto_real)	comprueba que objeto_esperado no sea el mismo objeto que objeto_real
fail()	

Tutorial 4B

- Modificar el contenido de las clases utilizando el archivo tutorial 4B
- Ejecutar las pruebas para analizar el orden de invocación
- Agregar errores en las clases para comprobar el funcionamiento

Práctica 4B

- Elegir una clase de la narrativa e implementarla
- Implementar una suite asociada a la prueba de esa clase
- Implementar al menos:
  - Dos tests que incluyan un AssertTrue
  - Un test de excepción
  - Dos tests test que incluyan assertEquals
  - Dos test que incluyan AssertNotNull

Pruebas unitarias

Introducción a testNG

TestNG

- Framework inspirado en Junit y Nunit
- Aplicable desde testeo de unidad hasta testeo de integración.
- Provee soporte para anotaciones
- Test dirigido por datos
- Tests parametrizables
- Uso de funciones JDK estándares

Test(i)NG – Escribiendo tests

- Escribir la lógica del test e insertar anotaciones
- Agregar información de los tests en un archivo XML (testng.xml o build.xml)
- Ejecutar TestNG
- Tags
  - <suite> Puede contener uno o mas tests
  - <test> Puede contener uno o más clases
  - <class> Una clase TestNG que puede contener uno o más métodos (@Test)

TestNG -Anotaciones

- @BeforeSuite El método se ejecutará antes de los tests
- @AfterSuite El método se ejecutará después de los tests
- @BeforeTest El método se ejecutará antes de cualquier test dentro <test>
- @AfterTest El método se ejecutará después de cualquier test dentro <test>
- @BeforeGroups El método se ejecutará una vez antes del primer test del grupo
- @AfterGroups El método se ejecutará al finalizar todos los test del grupo

TestNG – Anotaciones (cont.)

- @BeforeClass El método se ejecutará antes que cualquier método de la clase sea invocado.
- @AfterClass El método se ejecutará luego que todos los método de la clase hayan sido invocados.
- @BeforeMethod El método se ejecutará antes que cada método de test
- @AfterMethod El método se ejecutará luego de cada método de test

Modificadores

- alwaysRun:
  - @Beforex (excepto groups) se ejecutará independientemente a que grupo pertenezca.
  - @Afterx Se ejecutará independientemente que los métodos anteriores hayan fallado o ignorados
- dependsOnGroups Grupos a los cuales depende el método
- dependsOnMethod Métodos de los cuales depende el método
- enabled des/habilita la ejecución de los métodos de la clase
- Groups Grupos a los cuales pertenece la clase/método
- inheritGroups Indica si el método heredará los grupos especificados en @Test

TestNG – Anotaciones (cont.)

- @DataProvider Método proveedor de datos.
  - Devuelve un Object[][] donde cada elemento [] puede representar una lista de parámetros. El @Test que lo usará debe debe usar el name del dataProvider.
  - name Nombre del dataProvider
  - parallel Si los tests que utilizarán serán ejecutados en paralelo
- @Factory Objetos que serán usados por clases TestNG. Retorna Object[]
- @Listeners Listeners de una clase de test
  - value Arreglo de clases que extienden de org.testing.ITestNGListener
- @Describe como se pasan parámetros a un método @Test
  - value Lista de variables para los parámetros del método

TestNG – Anotaciones (cont.)

- @Test Marca una clase/método como parte del Test
  - alwaysRun: El método se ejecutará aun cuando fallen los anteriores
  - dataProvider nombre del proveedor para este método
  - dataProviderClass Clase donde buscar el dataProvider
  - dependsOnGroup Lista de grupos que dependen de este método
  - description Descripción de este método
  - enabled Métodos de esta clase están habilitados o no
  - expectedExceptions Lista de excepciones que se esperan recibir

TestNG – Anotaciones (cont.)

- @Test Marca una clase/método como parte del Test
  - groups Lista de grupos a los cuales pertenece el método
  - invocationCount Número de veces que se invocará el método
  - invocationTimeOut Tiempo máximo para ejecutar invocationCount
  - priority Prioridad del método (bajo número, alta prioridad)
  - successPercentage Porcentaje esperado de éxitos
  - singleThreaded Todos los métodos de la clase se ejecutarán en un único hilo
  - timeOut Tiempo que tomará como máximo el test

Ejemplo testng.xml

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="SuiteEntrega" verbose="1" >

 <test name="Conformidad">
  <classes>
   <class name="test.SistemaVM.FuncionalidadA"/>
   <class name=" test.SistemaVM.FuncionalidadB "/>
  </classes>
 </test>

 <test name="Conformidad">
  <packages>
   <package name="test.SistemaHALnalidadA"/>
  </packages>
 </test>
</suite>
```

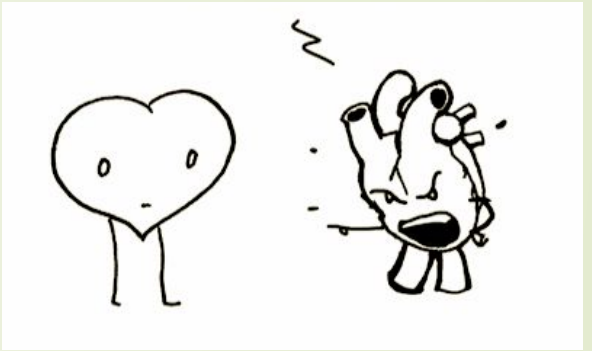
Manos a la obra

- Elegir una clase
- Implementar una suite asociada a la prueba de esa clase
- Implementar al menos:
  - Implementar al menos un @DataProvider
  - Identificar métodos que requieran test (al menos 4)
  - Utilizar parámetros
  - Uilizar dependsOnMethod

Pruebas unitarias

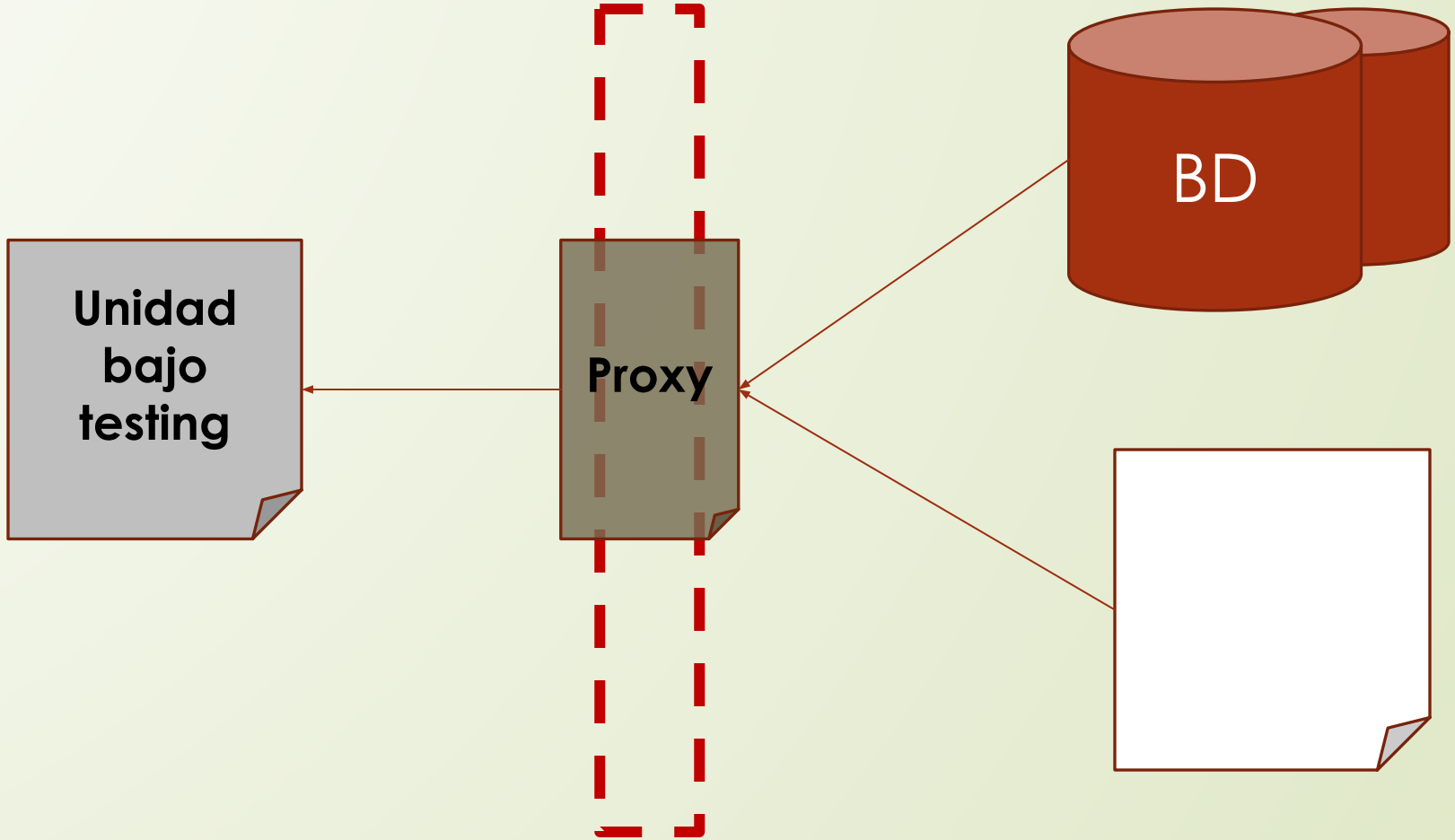
Mocking

Mocking



- Técnica que permite reemplazar el comportamiento complejo (o impráctico) de un objeto por otro que lo simule.
- Usado en testeo de unidades para eliminar dependencias.
- Aplicaciones:
  - Generación de entradas
  - Comportamiento de otros objetos o lógicas
  - Comprobación de salidas

Mocking de entradas



Mocking Modelo/Salida

Tutorial 4C

Siguiendo las indicaciones del archivo Tutorial 4C implementar el proxy de entradas