Temario Mantenimiento del Software

¿Qué es JUnit?

Es un marco de trabajo para escribir pruebas de unidad

¿Qué nos permite JUnit?

- Definir y ejecutar pruebas (test) y grupos de pruebas (test suites)
- Utilizar pruebas como una forma efectiva de especificación
- Apoyo a la refactorización
- Integrar código revisado en nuestros ejecutables

Estructura de las clases de prueba JUnit

- @BeforeClass
- @AfterClass
- @Before
- @After
- @Test
- @Test(timeout=1000)
- @Test(expected = RuntimeException.class)

Tipos de Assert

- AssertTrue
- AssertFalse
- AssertEquals
- AssertSame
- AssertNotSame
- AssertNull
- AssertNotNull
- Fail

¿Qué son las pruebas funcionales?

Son las pruebas en las que derivamos los casos de pruebas a partir de las especificaciones del sistema (Pruebas de caja negra). Las pruebas funcionales son especialmente buenas encontrando fallos de “lógica perdida”. Pueden ser aplicadas a todos los niveles:

- Unidad (a partir de la especificación de interfaces de módulo)
- Integración (a partir de las especificaciones de la API o sistema)
- Sistema (a partir de las especificaciones de requisitos del sistema)
- Regresión (a partir de los requisitos del sistema y el historial de bugs)

Criterios de cobertura

- **Cobertura de estados**: Cada estado debe ser visitado al menos por un caso de prueba
- **Cobertura de transiciones**: Cada transición entre estados es recorrida por almenos un caso de prueba
- **Cobertura de bucles**: Número de veces que se recorren los bucles (0,1, n)
- **Cobertura de nodos**: Los casos de pruebas pasan por todos los nodos del grafo
- **Cobertura en ramas**: Los casos de pruebas pasan al menos una vez por cada una de las bifurcaciones del sistema
- **Cobertura de sentencias**: Cada instrucción (o nodo) debe ejecutarse al menos una vez
- **Cobertura de condiciones simples:** Cada condición simple ha de tomar al menos una vez el valor true y false en al menos un caso de prueba
- **Cobertura de condiciones compuestas**: Descomponemos cada condición en condiciones simples y hacemos casos de prueba para que cada una se haga false y true al menos una vez
- **Cobertura de MC/DC**: Probamos combinaciones relevantes de las condiciones, para pasar de 2^n a n+1 combinaciones como máximo
- **Cobertura de caminos**: Cada arista ha de recorrerse al menos una vez
- **Cobertura de llamadas a procedimiento**
- **Cobertura de Complejidad ciclomática**
- **Definiciones**: Para cada definición hay al menos un caso de prueba que ejecuta un par DU que lo contiene
- **Pares DU**: Para cada par DU hay al menos un caso de prueba que lo ejecute
- **Caminos DU**: Cada camino DU simple ha de ser ejecutado al menos una vez

¿Qué son las pruebas estructurales?

Son pruebas que se realizan a partir de la implementación de un sistema (Pruebas de caja blanca). Estas pruebas incluyen casos que no pueden identificarse de la especificación. Son mejores para encontrar defectos del código que para explorar el comportamiento del sistema. Pueden estar enfocadas a:

- Flujo de control
- Flujo de datos

¿Qué son las pruebas de integración?

Son aquellas que se realizan en el ámbito del desarrollo de software una vez que se han aprobado las pruebas unitarias, lo que prueban es que todos los elementos unitarios que componen el software funcionan juntos correctamente probándolos en grupo. Se centra principalmente en probar la comunicación entre los componentes y sus comunicaciones ya sea hardware o software. Los objetivos de las pruebas de integración son distinguir errores de integración que deberían haber sido eliminados con pruebas de unidad y entender la naturaleza de los errores de integración (para prevenirlos y detectarlos)

Estrategias de integración:

- Probamos tras integrar todos los módulos
    - Ventajas
        - No requiere ningún tipo de andamiaje
    - Desventajas
        - Observabilidad mínima
        - Capacidad de diagnóstico nulo
        - Eficacia cero
        - Retroalimentación inadecuada
        - Alto coste de reparación
- Orientadas a la estructura: Los modulos se desarrollan, integran y se prueban según la estructura jerárquica del proyecto
    - De arriba abajo
    - De abajo arriba
    - Bocadillo
- Orientadas a la funcionalidad: Los módulos se integran según las características de la aplicación
    - Hebras
    - Módulos críticos (es necesario una estimación de riesgos)

¿Qué es el andamiaje?

Código que se produce para dar soporte a las actividades de desarrollo (no es parte del producto que ve el usuario), incluye test harnesses (arnés), drivers (clase de prueba) y stubs

¿Qué es Mockito?

Es un framework libre para Java. Se utiliza para simplificar las pruebas de unidad (integración) y proporcionar control sobre el contexto de ejecución permite remplazar objetos reales con los que colabora el objeto probado con sustitutos falsos. Todas las operaciones sobre el podemos verificarlas con Mockito.verify(T mock).

- Tipos:
    - **Dummy**: Objeto vacío pasado en una invocación
    - **Fake**: Objeto con una implementación funcional, pero simplificada, solo para permitir una prueba (ej: Base de datos)
    - **Stub**: Objeto con un comportamiento “enlatado” para una prueba concreta
    - **Mock**: Objeto con la habilidad de tener un comportamiento programado esperado y de verificar las interacciones en las que interviene
    - **Spy**: Mock creado como un proxy de un objeto real (unos métodos son simulados y otros redirigidos al objeto real)
- Funciones para especificar comportamiento de un mock
    - **ThenReturn**(T valueToBeReturned)
    - **ThenThrow** (Throwable toBeThrown)
    - **ThenThrow** (Class<? Extends Throwable> toBe Thrown)
    - **Then**(Answer answer)
    - **ThenAnswer**(Answer answer)
    - **ThenCallRealMethod**() -> Invoca un método real para mock/spy

- Las pruebas se dividen en tres partes:
    - **Prepara** (arrange, given): Inicialización y configuración del software a probar y de los mocks
    - **Actúa** (act,when): Operación sujeta a pruebas (recomendado usar una sola operación)
    - **Comprueba** (assert, then): Fase de comprobación y verificación
- Modos de verificación
    - **Times**(int)
    - **Never**()
    - **atLeastOnce**()
    - **atLeast**(int)
    - **atMost**(int)
    - **only**()
    - **timeout**(int)

¿Qué es un matcher?

Es un comparador de argumentos de mockito

- Matchers predefinidos
    - Any(), Any (Class <T> class) -> Cualquier objeto o null
    - AnyBoolean(), AnyByte(), AnyChar()…
    - Eq(T value)
    - isNull
    - isNotNull()
    - isA(Class <T> class)
    - refEq(T value, String …
    - matches (String regex)
    - startsWith(String), endWith(String)
    - aryEq
    - cmpEq
    - gt(value), geq(value), it(value), leq (value) -> con Comparable <T>

Si utilizamos un matcher para uno de los argumentos de un método, todos los argumentos deben de ser matchers.

Pruebas de GUIs

La principal característica de cualquier interfaz gráfica de usuario es que es guiada por eventos. La esencia de las pruebas de sistema para aplicaciones con GUI es ejercitar esta naturaleza guiada por eventos. Los modelos UML proporcionan diagramas muy apropiados para ello.

Lo primero será identificar todos los eventos de entrada del usuario y todos los eventos de salida del sistema.

Las pruebas GUI son complicadas pero imprescindibles y dependiendo de la complejidad, las pruebas pueden ser muy numerosas. Para que estas pruebas sean efectivas han de ser automatizadas, esto supone un coste más alto, aunque los beneficios vienen cuando el sistema es estable

Herramientas de pruebas de GUI:

- Abbot: es una extensión de JUnit que permite escribir pruebas programadas, permite especificación con pruebas XML, orientado al probador y con grabación. Utiliza varios atributos para identificar componentes de la GUI dinámicamente sin depender de posiciones.
- Java.awt.Robot: es una clase de muy bajo nivel y necesita demasiado código para simular acciones de usuario (controlando el ratón y el teclado del entorno de la aplicación)
- FEST: Construido sobre Abbot, facilita la creación y mantenimiento de pruebas funcionales para GUIs. Tiene extensiones a Selenium y tes formas de localizar componentes: por nombre, por tipo y por específica

Selenium

Es un conjunto de diferentes herramientas, cada una con una aproximación distinta para soportar la automatización de pruebas.

Selenium IDE es un plugin para Firefox para construir scripts de prueba, tiene una funcionalidad de grabación que registra las acciones del usuario y volver a ejecutarlas, también puede crear los scripts con el asistente que tiene integrado y exportar las pruebas realizadas a varios lenguajes que sean ejecutados con WebDrive

- Comandos:
    - Open: Abre una pagina usando una URL
    - Click/ClickAndWait: Ejecuta un click y opcionalmente espera a que la página se cargue
    - verifyTitle/assertTitle: Verifica el titulo de la pagina
    - verifyElementPresent: Verifica que el elemento de la IU está presente tal y como está definido en su HTML
    - waitForPageToLoad: Pausa la ejecución hasta que la nueva página está cargada (llama a clickandWait)
    - waitForElementPresent: Pausa la ejecución hasta que el elemento de la interfaz indicado está presente.

Selenium WebDriver es el resultado de la unión de los proyectos Selenium y WebDriver. Proporciona una API de programación para desarrollar scripts de prueba usando diferentes lenguajes de programación. Los test se pueden ejecutar en los distintos navegadores soportados por selenium (Firefox,Chrome, Explorer, Safari y Opera)

- Drivers
    - HTMLUnit Driver: Es la implementación más rápida y ligera de WebDriver. Se basa en una implementación Java de un navegador sin GUI
    - FirefoxDriver
    - InternetExplorerDriver: Se controla mediante .dll
    - Chrome Driver
    - Otros…

- Metodos:
    - Driver.get(http://...)
    - Driver.findElement(By.
        - ID
        - TagName
        - Nombre
        - XPath
        - UsandoJavaScript
        - Por Css
        - Por LinkText
    - Driver.sendKeys(String): introduce texto en un campo de texto
    - Driver.click
    - Driver.getText()
    - Driver.getAttribute/SetAttribute
    - Driver.submit
    - Driver.switch: para cambiarnos entre frames, Windows o pop up Windows
    - Driver.navigate(): para movernos entre paginas to(String),forward,backward
    - Select(Ej: Select x = new Select(driver.find….)
- Esperas explícitas: se utiliza para establecer el tiempo de espera para solo una instancia en particular.
- Esperas implícitas: se utiliza para establecer el tiempo de espera predeterminado en todo el programa

Selenium Grid es muy buena para pruebas de compatibilidad con diferentes navegadores, se utiliza ampliamente en pruebas de regresión o para automatizar tareas repetitivas. Permite la ejecución remota y distribuida de scripts de Selenium y puede ejecutar en paralelo, así como en Android o iOS.

Tipos de pruebas de sistema

- De capacidad: Cuando los sistemas deben tratar grandes volúmenes de datos. (Construiremos un arnés que genere gran cantidad de datos para ver cómo reaccionan)
- De estrés: Sistemas que deben responder en tiempo real. (Realizaremos un tráfico de datos muy grande y en condiciones extremas)
- De usabilidad: Sistemas con una interfaz de usuario significativa donde es importante evitar errores. (Muchos usuarios prueban esta aplicación en un entorno controlado). Pruebas sobre modelos mentales de los usuarios, para evaluar opciones de diseño de interfaces o para validar la usabilidad del sistema
- De rendimiento
- De seguridad: La mayoría de los sistemas abiertos, para evitar usuarios malintencionados (Comprobar errores típicos que intenten romper el sistema)
- De fiabilidad: Necesitamos probar que ciertos sistemas fallarán con muy poca frecuencia. (Creamos una prueba relevante y recopilamos información para apoyar la afirmación estadística)
- De conformidad: (Conjuntos de pruebas para comprobar el funcionamiento correcto del sistema)
- De disponibilidad/reparabilidad (Se interesa en encontrar fallos e intentar resolverlos)
- De documentación (Para que los usuarios no se confundan)

- De configuración

Pruebas de aceptación

Buscan guiar en la decisión de si el producto puede ser distribuido. Para ello se buscan medidas de calidad y se aplican criterios como fiabilidad, disponibilidad, tiempo medio entre fallos…

Pruebas de regresión:

Las pruebas deben de ser ejecutadas tras cada cambio en la aplicación y pueden suponer una gran parte de los costes del mantenimiento del software

Particularidades del software para la web

A menudo el uso de este software supone combinar numerosas tecnologías que son muy exigentes por lo que la aplicación web necesita ser construida mejor y con un mayor número de pruebas. No se puede aplicar grafos de flujo de control o de llamadas, el diagrama de estados es mucho más difícil de modelar, todo esto complica más el diseño y mantenimiento de una web. Es necesaria la integración dinámica de nuevos componentes software.

- Validación de datos de entrada del cliente:
    - Restricciones HTML: Longitud, valor, modo de transferencia, elemento de campo…
    - Restricciones de Scripts: Tipo de dato, formato de dato, valor de dato…
- Validación de datos de salida
    - Respuestas validas (V)
    - Fallos y errores (F)
    - Exposicíon (E): entradas no reconocidas que implican un comportamiento anormal y se expone a los usuarios

- Para probarlo
    - Tratamos la página web como caja negra
    - Realizamos pruebas de configuración y compatibilidad

¿Para qué producir documentación (de calidad)?

- monitorizar y estimar el proceso
- mejorar el proceso
- reutilizar los conjuntos de pruebas

Tipos de documentos

- De planificación
- De especificación
- Informes

TRABAJOS

Selendroid

Es un framework de automatización de pruebas móviles de código abierto. Esta automatización puede realizarse tanto en aplicaciones nativas como en aplicaciones web y tanto en dispositivos físicos como emuladores.

Tiene una arquitectura:

- Selendroid-Cliente: Cliente de java
- Selendroid-Server: Se ejecuta a la vez que la aplicación en el dispositivo Android
- AndroidDriver-App: Es un driver que reconstruye la aplicación Android a web
- Selendroid-Standalone: Maneja diferentes dispositivos

Se utiliza porque no hace falta modificar la aplicación que se está testeando, solo necesitas el apk instalado en un ordenador y que los dispositivos esten firmados con la misma clave. Permite interactuar con múltiples dispositivos y simuladores a la vez (así compruebas la compatibilidad). También te permite simular la interacción que tendría una persona con el dispositivo y reconoce dispositivos nuevos automáticamente permitiéndote añadirlo a las pruebas sin tener que reiniciarlas

Appium

Es una herramienta que deriva de selenium y que permite realizar pruebas en dispositivos móviles. Es necesario tener instalado adb para que permita la conexión con el móvil

Diferencias entre Selenium y Appium.

Al derivar de selenium presenta numerosas similitudes, pero añade funciones como Locators de clase MobileBy, presskeycode para facilitarnos funciones de selenium, tap , currentActivity que devuelve un string con el nombre de la actividad actual. Además de otras funciones como para instalar una app, saber si está instalada una app o si se encuentra bloqueado el dispositivo.

Otras diferencias son que no estarán disponibles las funciones como driver.get() o driver.navigate a menos que esteneos en el navegador del móvil.

FindBugs

Es un programa de código abierto que busca errores en programas escritos en código JAVA. Utiliza un análisis estático para identificar cientos de tipos de errores potenciales (no tienen por qué ser un error, aunque lo identifique como patrón de error)

M.U.S.I.C

(Mutante: una copia del sistema en el que se ha realizado un pequeño cambio

Matanza: cuando al menos uno de los casos de prueba detecta el mutante)

Análisis de mutación:

- Creación de mutantes (automática)
- Ejecución de casos de prueba en el original y los mutantes (automática)

- Cálculo de la mutación sobre el score (semiautomática)

Para optimizar esto, se paraleliza la ejecución de mutantes en paralelo y se reducen los mutantes generados (mediante mutación selectiva)

Funcionalidad: Identifica situaciones donde un mutante no será ejecutado (reduce el número de ejecuciones) y cuando una mutación implica un bucle infinito para parar su ejecución

Generación automática de pruebas

Su objetivo es proporcionar al desarrollador un conjunto de pruebas efectivas e importantes. En las pruebas de mutaciones, se introducen mutantes en el programa y los casos de pruebas son evaluados respecto a cuántos defectos se pueden distinguir con el original.

La recomendación es no automatizar la realización de los casos de prueba

Evosuite es una herramienta de apoyo a los tester que trata de optimizar la cobertura añadiendo casos de prueba siguiendo algoritmos de búsqueda de ejecución simbólica dinámica. Sus objetivos son:

- Cobertura total de casos de prueba
- Generación de assert basados en mutantes
- Mantener el conjunto de pruebas tan pequeño como se pueda

Los problemas son su costo computacional, desgaste muy alto en tiempo y esfuerzo, puede generar casos de pruebas no esenciales y que solo trata con aplicaciones de un thread

Evosuite no es capaz de predecir errores ya que su único recurso es el código fuente de la clase probada, solo detecta errores de eficiencia

Pitest

Es una librería para mutación de código compatible con otros frameworks como JUnit, constantemente actualizado y capaz de ejecutar numerosas pruebas de mutantes en un tiempo bajo. Para ello crea un proceso padre y gestiona los hijos mediante la paralelización. Si un hijo entra en bucle es capaz de finalizar este proceso y no escribe nada en disco.

Concolic Testing

Es una técnica de verificación del software que lleva a cabo la ejecución simbólica, una técnica clásica que trata las variables del programa como variables simbólicas, alo largo de una ejecución completa. Su objetivo principal es encontrar errores en el software del mundo real y verificar modelos

Moles: es una herramienta diseñada por Microsoft que facilita el testing de soluciones proviniendo de aislamiento mediante desvíos y delegados, siendo estos últimos básicamente un encapsulamiento de los métodos. (Suele venir acompañado de Pex)

Pex: Es una herramienta diseñada por Microsoft para el framework .net. Pex es capaz de generar automáticamente conjunto de casos de uso (concolic test), para realizar una alta cobertura del código

Los problemas encontrados son que Pex tiene problemas al ejecutarse en clases parametrizadas y que solamente alterna los valores de las variables pasadas por parámetros.

Iron Wasp

Es un escáner de seguridad orientado a aplicaciones web. Es un software gratuito de funcionalidad automática que provee de funcionalidades como secuencias de login o detección de falses negativos y falsos positivos

Nessus: herramienta de pruebas de seguridad (la más utilizada) y permite búsquedas de vulnerabilidades de forma rápida y sencilla. Problema, es de pago.

Nikto: herramienta de seguridad orientada para servidores web, es rápida y pone énfasis en la detección de archivos dañinos

Wapiti: Programa de caja negra que se adentra en las páginas web buscando vulnerabilidades.

Calabash, Cucumber & Gherkin

Gherkin: Es un lenguaje ubicuo que usaremos para describir funcionalidad (sin entrar en su implementación) en Calabash. Permite crear documentación a la vez que automatizamos los test.

Cucumber: Es una de las herramientas que podemos utilizar para automatizar nuestras pruebas en el desarrollo dirigido por comportamiento.

Calabash: Es un framework que se encarga de establecer un puente entre las pruebas de Cucumber y nuestras apps. Nos permite realizar interacción entre componentes gráficos de nuestra aplicación

Pruebas con Karma y Jasmine con Angular JS

Angular JS es un framework de JavaScript de código abierto que se utiliza para crear y mantener aplicaciones web

Jasmine es un framework de desarrollo dirigido por comportamiento de código abierto para testear JavaScript (basado en JUnit). Está pensado para arquitecturas asíncronas con la integración de espías para la implementación de test doubles y dispone de una gran cantidad de matchers predefinidos

Karma es una herramienta de automatización de pruebas para Javascript para la línea de comandos, de esta manera los desarrolladores pueden ver a tiempo real sus fallos.

TDD vs ATDD

TDD emplea test de unidad automatizados para dirigir el diseño del software

ATDD se emplea para conseguir que los clientes se involucren en el proceso de diseño antes de que el desarrollo comience, es una extensión de TDD

BDD es un enfoque colaborativo al desarrollo de software que pretende eliminar la brecha de comunicación entre el negocio y la tecnología informática. La principal ventaja es el entendimiento entre clientes y desarrolladores

Discovery/Specification work shops: son encuentros frecuentes y cortos donde stakeholdes, developers y tester se reúnen para discutir las características del sistema