Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

Grado en Ingeniería Informática – Ingeniería del Software

Diseño y Pruebas 2

Curso 2023 – 2024

Conocimientos previos

Grupo C1.013

Miembros	Información de contacto
David Fuentelsaz Rodríguez	davfuerod@alum.us.es
Miguel Galán Lerate	miggaller@alum.us.es
Antonio Jiménez Ortega	antjimort@alum.us.es
Josué Rodríguez López	josrodlop19@alum.us.es
Óscar Zurita Urpina	osczururp@alum.us.es

Repositorio: https://github.com/DP2-C1-013/Acme-SF-D01-24.1.0

Fecha	Versión
14/2/2024	v1.0

Conceptos teóricos previos de testing

Tenemos ciertos conocimientos previos sobre el testing, especialmente del testing unitario, de qué va, cómo se hace y para qué sirve. La mayoría de estos conocimientos vienen de la asignatura Diseño y Pruebas 1.

En el testing buscamos probar si funciona como es esperado un software que hemos diseñado. Esto se puede hacer de muchas formas; directamente interactuando con el programa como si fuésemos un usuario final, o de forma programática, que es en lo que más nos centramos.

En el testing de forma programática hay, a su vez, muchos tipos de tests; unitarios, de integración, funcionales, aceptación, etc. Por ahora, solo hemos tenido experiencia con los unitarios, los cuales son tests rápidos, no dependen de condiciones externas y se comprueba el resultado de forma automática (si ha pasado el test o no).

A la hora de hacer tests unitarios de nuestros componentes software de forma programática, debemos establecer un SUT *subject under test,* el componente software a probar, y diseñar los casos de prueba positivos y negativos, intentando probar todos los posibles escenarios en los que nuestro software se puede usar, e intentando cubrir todo el código no trivial.

Los casos de prueba son los conjuntos de condiciones a los que se someterá el software para determinar si funciona satisfactoriamente o no. Los llamamos positivos si siguen el Happy path, es decir, prueban el funcionamiento ante unas condiciones normales y esperadas. Por otro lado, los casos de prueba negativos le dan al sistema inputs no esperados o invalidos, para ver cómo el software es capaz de manejarlos.

Conceptos técnicos previos

Aparte de saber todo lo descrito previamente a nivel teórico, también nos hemos enfrentado a aplicaciones reales en las que hemos tenido que aprender a implementar programáticamente testing unitario.

Hemos diseñado y desarrollado testing para controladores, servicios y repositorios del backend de una aplicación modelo vista controlador, usando JUnit, Mockito y algunas otras librerías de testing de Spring Boot.

Conocemos anotaciones útiles en estos entornos:

- @Test: indica que el trozo de código que le sucede es un caso de prueba
- @BeforeEach: anotado antes de una función, sirve para que esta se ejecute antes de cada tests. Es muy útil para definir un escenario de prueba común a todos los tests

- @BeforeAll: parecido a @BeforeEach, pero se ejecuta una única vez antes, antes de la ejecución de todas las pruebas.
- @Disabled: si anotamos un test con esta anotación, este no se ejecutará con el resto de pruebas.
- @AfterEach: un código con esta anotación se ejecutará una vez después de cada prueba.
- @AfterAll: indica que el código con esta anotación se ejecutará una única vez tras todas las pruebas
- @MockBean: sirve para indicar que una dependencia se va a “imitar” con un mock, es decir, se va a simular cómo deben funcionar métodos externos, para así lograr mayor independencia y rapidez en los tests unitarios. Útil en los tests de los controllers. Pertenece a Mockito

A la hora de diseñar un test seguimos el principio de Arrange, Act, Assert: primero establecemos el escenario de prueba, los datos con los cuales se va a ejecutar la prueba, luego actuamos sobre ellos, realizamos las acciones en las que consiste la prueba, y por último comprobamos el resultado, típicamente con un *assert.*