Pruebas Unitarias

Sirven para probar el correcto funcionamiento de una parte concreta del código (generalmente un método), y que cada parte funciona correctamente por separado.

La preparación de estas pruebas supone un incremento del tiempo destinado al proyecto. Según el proyecto que estemos desarrollando, habrá que valorar si es conveniente desarrollarlas.

Si están bien definidas, mejoramos la calidad del software, ya que conforme el software va evolucionando, nos aseguramos que las modificaciones realizadas siguen cumpliendo con los requisitos de validación establecidos.

Pruebas de integración

Prueban la conexión entre distintos componentes. Sería el siguiente paso, tras las pruebas unitarias. (Ejemplo: facturación -> contabilización)

Pruebas funcionales

Parecidas a las de integración, pero validando los componentes relacionados con una funcionalidad concreta. (Ejemplo: Cálculo nómina)

Pruebas de aceptación de usuarios

Pruebas definidas por personas y pasadas de forma manual.

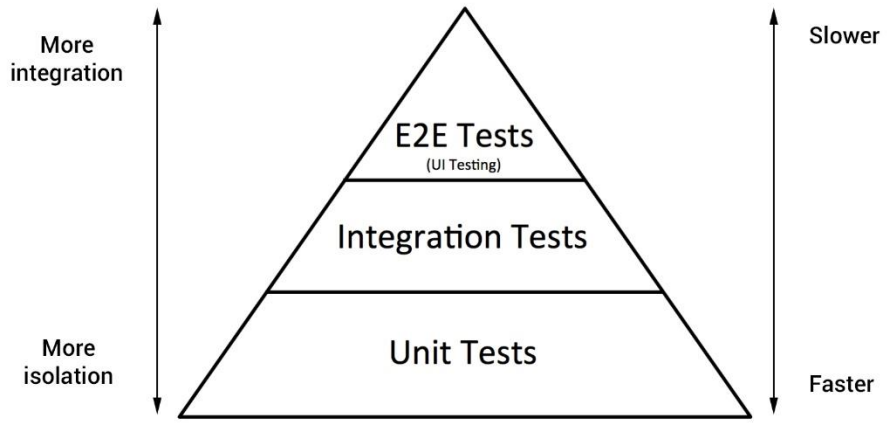
Pruebas de regresión

Prueban que las pruebas unitarias y funcionales siguen funcionando a lo largo del tiempo (cada vez que se libera una nueva versión, por ejemplo).

Pruebas de carga

Prueban la eficiencia del código ante una carga de trabajo.

Pirámide de Cohn



El principio FIRST

FIRST es el acrónimo de las cinco características que deben tener nuestros tests unitarios para ser considerados tests con calidad:

- Fast (rápido de ejecutar)
- Independent (independiente de otros tests unitarios)
- Repeatable (repetible en el tiempo y sin dependencias del servidor o la plataforma)
- Self-validating (validable por sí mismo, automatizable y verificable sin necesidad de consultar archivos externos de resultados)
- Timely (Oportuno. Planificar/desarrollar primero los tests y luego el código - TDD)

Condicionantes

Suelen probarse únicamente los métodos públicos o principales de cada clase.

Siempre tienen la misma estructura:

- Preparación de los datos de entrada
- Ejecución del test
- Comprobación del test (asserts), no debiendo haber más de un assert por test.

JUnit

Framework de java para implementar los tests unitarios, basado en anotaciones.

Las condiciones de aceptación se implementan mediante asserts:

- assertTrue(): Comprueba que la condición sea cierta.
- assertFalse(): Comprueba que la condición sea falsa.
- assertEquals(): Comprueba condición de igualdad.
- assertNotEquals(): Comprueba condición de desigualdad.
- assertNull(): Comprueba que el resultado sea nulo.
- assertException(): Comprueba que se lanza una determinada excepción
- fail() – Provoca un error en el test.

Todas las aserciones:

https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html

Equivalencias Junit 4 / Junit 5

@Test (Indica un test a verificar)

@DisplayName (En lugar del nombre del método, se mostrará el contenido de esta etiqueta)

@BeforeClass (static) / @BeforeAll (static) (Acciones antes de ejecutar todos los tests)

@AfterClass (static) / @AfterAll (static) (Acciones después de ejecutar todos los tests)

@Before /@BeforeEach (Acciones antes de ejecutar cada test)

@After / @AfterForEach (Acciones después de ejecutar cada test)

@Ignore / @Disabled (Test deshabilitado)

TDD (Test driven Development)

Primero los tests, y después el código.

Es un proceso continuo de: Codificación -> pruebas -> Desarrollo -> Refactorización.

Pasos a seguir:

- Escribir las pruebas.
- Ejecutar las pruebas. Deben fallar, ya que el código a testear no está todavía desarrollado.
- Escribir el código de nuestro producto.
- Ejecutar de nuevo las pruebas. En este caso las pruebas deberían de funcionar y darnos un resultado positivo.

Grado de cobertura de pruebas sobre el código (Coverage)

Es una medida porcentual que indica que porcentaje del software se está validando mediante tests unitarios.