Pruebas unitarias

¡Comencemos!

¿Qué aprenderás?

- Comprender la importancia de las pruebas unitarias para su uso en diversos códigos de Java.
- Comprender cómo cambia el flujo de desarrollo de un código al implementar pruebas unitarias.

Introducción

Es común encontrarse en situaciones donde se deben realizar cambios al sistema, pero ¿cómo validamos estos cambios y comprobamos que estén buenos sin la necesidad de acudir a recursos externos? La respuesta es simple y se llama “Prueba o Test Unitario”.

Este proceso evita la espera de verificaciones externas y nos permite comprobar funcionalidades nuevas que podemos adherir a nuestro código. Por lo tanto, si ocurre un fallo en producción no debemos esperar hasta final para realizar verificaciones, sino que podemos corregir problemas a tiempo. Las nuevas funcionalidades son parte del día a día en los/las desarrolladores/as y debemos entender muy bien estos conceptos para mejorar nuestro código.

¡Vamos con todo!

Test Unitarios

Las pruebas o test unitarios vienen a verificar las unidades individuales del código para que cumplan con el funcionamiento esperado. Además, ayudan a comprender el diseño del código en el que se está trabajando. Cuando se trabaja en un equipo de varios integrantes que realizan cambios casi a diario en el código de la aplicación, las pruebas unitarias toman un rol importante, ya que mantienen la calidad del código y la integración de las funcionalidades nuevas para utilizarlas ahora y no después. Para esto, comenzaremos definiendo conceptos importantes como lo son los ciclos de vida en los test unitarios.

Ciclo de vida en Test Unitarios

Cuando se adoptan las pruebas unitarias se modifican los procesos por los cuales pasan características añadidas al código. Por ejemplo, cuando un/a integrante del grupo de trabajo efectúa cambios en la aplicación, las pruebas unitarias deberían seguir corriendo frente a estos cambios. Por lo tanto, cada vez que trabajemos sobre la aplicación, debemos hacer una actualización sobre los últimos cambios y, en base a estos, comenzar a realizar pruebas.

A continuación, te mostraremos el diagrama del ciclo de las pruebas unitarias (Imagen 1) que se aplica en cada cambio:

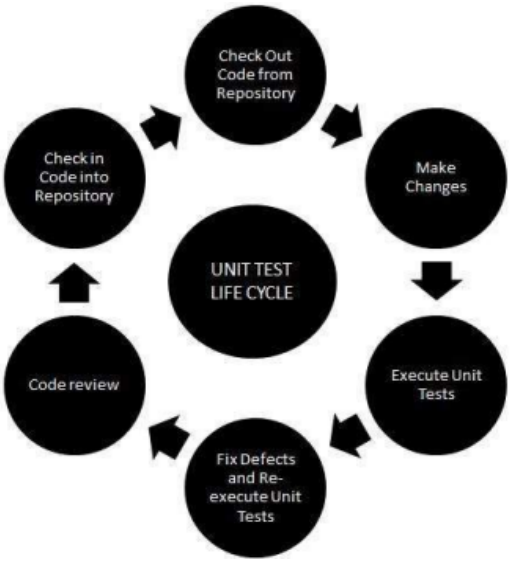

Imagen 1. Ciclo de vida en Test Unitario.
Fuente: Desafío Latam.

1. El proceso comienza descargando el código desde su respectivo repositorio o lugar donde se almacena el código fuente (“Check Out Code from Repository”).

2. Se realizan los cambios y se agregan nuevas funcionalidades (“Make Changes”).

3. Se escriben las pruebas unitarias y son ejecutadas (“Execute Unit Tests”).

4. Se corrigen las pruebas fallidas y se ejecutan nuevamente, al ser exitosas se sigue con el flujo (“Fix Defects and Re-execute Unit Tests”).

5. De forma optativa, se puede hacer una revisión de código por parte de otro miembro del equipo para validar los cambios (“Code Review”).

6. Se confirman y se suben los cambios al repositorio (“Check in Code into Repository”).

Ventajas de realizar pruebas unitarias

- Se escriben pequeñas pruebas, lo que obliga a que el código sea modular (de lo contrario sería difícil probarlo).
- Las pruebas sirven de documentación.
- El código es más fácil de mantener y refactorizar.
- Las pruebas unitarias son valiosas como red de seguridad cuando se necesita cambiar el código para agregar nuevas funciones o para corregir un error existente.
- Los errores son atrapados casi inmediatamente.
- Hace más eficiente la colaboración entre miembros de equipo, ya que se puede editar el código de cada uno con confianza. Las pruebas unitarias verifican si los cambios realizados hacen que el código se comporte de manera inesperada.

Desventajas de realizar pruebas unitarias

- Las pruebas deben mantenerse.
- Inicialmente, ralentiza el desarrollo.
- Las pruebas unitarias deben adoptarse por todo el equipo.
- Un reto que puede ser intimidante y no es fácil de aprender al comienzo.
- Difícil de aplicar al código heredado existente.

Las dificultades que vienen con las pruebas unitarias están en el cómo se vende el concepto a una organización. A veces, se rechazan las pruebas unitarias porque el desarrollo conlleva más tiempo y retrasa la entrega. En algunos casos esto puede ser cierto, pero no hay forma de saber si esos retrasos son una consecuencia real de las pruebas unitarias o por un diseño deficiente.

Sin embargo, al lograr un buen diseño y con requisitos bien definidos, las pruebas unitarias entregan muchos beneficios, a continuación, detallaremos algunos:

1. **Facilita realizar cambios:** Como punto de partida las pruebas unitarias hacen que el desarrollo sea mayormente ágil. Cuando se agregan nuevas características a un sistema a veces necesita refactorizar el diseño y el código antiguo. Sin embargo, cambiar el código ya probado generalmente es arriesgado. Pero si ya se tienen las pruebas unitarias implementadas, entonces se procede a refactorizar con confianza. Las pruebas unitarias colaboran con la agilidad porque se basa en pruebas que permiten realizar cambios con mayor facilidad al refactorizar.

2. **Mejora la calidad del código:** Además de mejorar la calidad del código, las pruebas unitarias identifican los defectos que pueden surgir antes de que el código pase a las pruebas de integración. Escribir pruebas antes de escribir la lógica de negocios permite simplificar el problema ayudando a exponer los casos de borde para escribir un mejor código.

3. **Encuentra errores en el diseño:** Los problemas se encuentran en una etapa temprana, dado que los/las desarrolladores/as que prueban un código individual antes de la integración realizan estas pruebas unitarias. Los errores encontrados se resuelven en cualquier momento sin afectar las otras partes del código, y puede incluir errores en la implementación del programador y fallas o partes faltantes de la especificación de la unidad.

4. **Proporciona documentación:** Las pruebas unitarias pueden incluso ser usadas como documentación del sistema. Para los desarrolladores que buscan entender qué característica proporciona y cómo se utiliza cada unidad, las pruebas muestran la funcionalidad de esta. Por otra parte, si las pruebas son claras y expresan sus intenciones de forma concisa, el código será fácil de entender.

Gestores en el ciclo de vida

La gestión del ciclo de vida brinda un marco para el desarrollo de software y permite gestionar sistemas a lo largo del tiempo. Por otra parte, el desarrollo del ciclo de vida en aplicaciones involucra a personas, herramientas y procesos que gestionan una aplicación desde que se diseña hasta que deja de estar disponible.

Los gestores son sistemas automatizados que buscan simplificar los procesos de construcción (limpiar, compilar y generar ejecutables del proyecto a partir del código fuente). En esta unidad abordaremos Apache Maven como gestor.

Maven es una herramienta de gestión y comprensión de proyectos de software que se basa en el concepto "POM" o "Project Object Model". Permite gestionar todo aquello que está relacionado a la compilación, los informes y la documentación a partir de una información central.

El archivo `pom.xml` es el núcleo de configuración para un proyecto en Maven. El POM es enorme y puede ser desalentador en cuanto a su complejidad de código, sin embargo, no es necesario entender todo, si no que aquello que utilizaremos.

Ejercicio guiado: Ciclo de vida

A continuación, ordena del 1 al 6 las etapas del ciclo de vida de las pruebas unitarias.

- Rellenar columna “Posición en el ciclo de vida” según el lugar correspondiente.

Conceptos	Posición en el ciclo de vida
Se confirman y se suben los cambios al repositorio (“Check in Code into Repository”).	
Se corrigen las pruebas fallidas y se ejecutan nuevamente, al ser exitosas se sigue con el flujo (“Fix Defects and Re-execute Unit Tests”).	
Se escriben las pruebas unitarias y son ejecutadas (“Execute Unit Tests”).	
Se puede hacer una revisión de código por parte de otro miembro del equipo para validar los cambios (“Code Review”).	
Se descarga el código desde su respectivo repositorio o lugar donde se almacena el código fuente (“Check Out Code from Repository”).	
Se realizan los cambios y se agregan nuevas funcionalidades (“Make Changes”).	

Tabla 1. Ciclo de vida en Test Unitario.
Fuente: Desafío Latam.

Solución Tabla 1

Conceptos	Posición en el ciclo
Se confirman y se suben los cambios al repositorio (“Check in Code into Repository”).	6°
Se corrigen las pruebas fallidas y se ejecutan nuevamente, al ser exitosas se sigue con el flujo (“Fix Defects and Re-execute Unit Tests”).	4°
Se escriben las pruebas unitarias y son ejecutadas (“Execute Unit Tests”).	3°
De forma optativa, se realiza revisión de código por parte de otro miembro del equipo para validar los cambios (“Code Review”).	5°
Se descarga el código desde su respectivo repositorio o lugar donde se almacena el código fuente (“Check Out Code from Repository”).	1°
Se realizan los cambios y se agregan nuevas funcionalidades (“Make Changes”).	2°

Tabla 2. Solución tabla 1.
Fuente: Desafío Latam.