Pruebas unitarias y TDD

Pruebas unitarias

{desafío} latam_

Implementar una suite de pruebas unitarias en lenguaje Java utilizando JUnit para asegurar el buen funcionamiento de una pieza de software

- Unidad 1: Flujo, ciclos y métodos
- Unidad 2: Arreglos y archivos
- Unidad 3: Programación orientada a objetos
- Unidad 4: Pruebas unitarias y TDD

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *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.*

¿Qué entendemos por pruebas?

/* Pruebas unitarias */

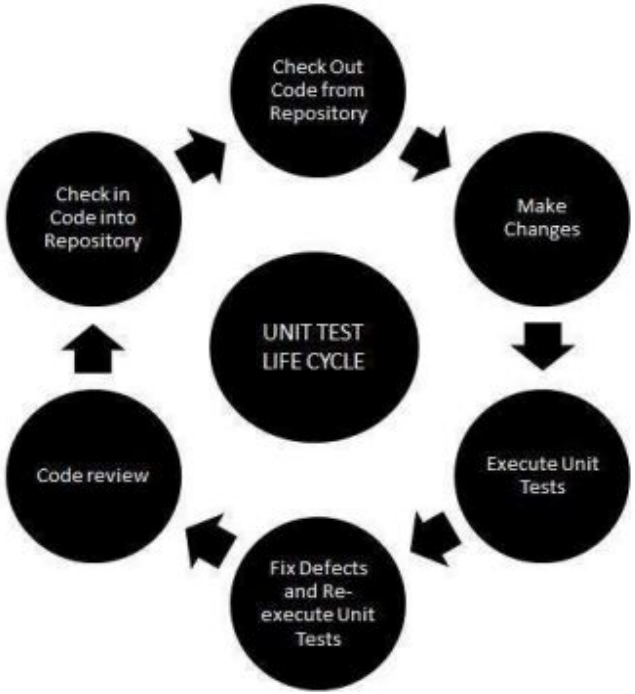
Test unitarios

- Verifican las unidades individuales del código para que cumplan con el funcionamiento esperado.
- Ayudan a comprender el diseño del código en el que se está trabajando.
- Mantienen la calidad del código y la integración de las funcionalidades nuevas para utilizarlas ahora y no después.

Ciclo de vida

Cuando se adoptan las pruebas unitarias se modifican los procesos por los cuales pasan características añadidas al código.

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.



Ciclo de vida

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”).

Ciclo de vida

Ventajas

- 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.

Ciclo de vida

Desventajas

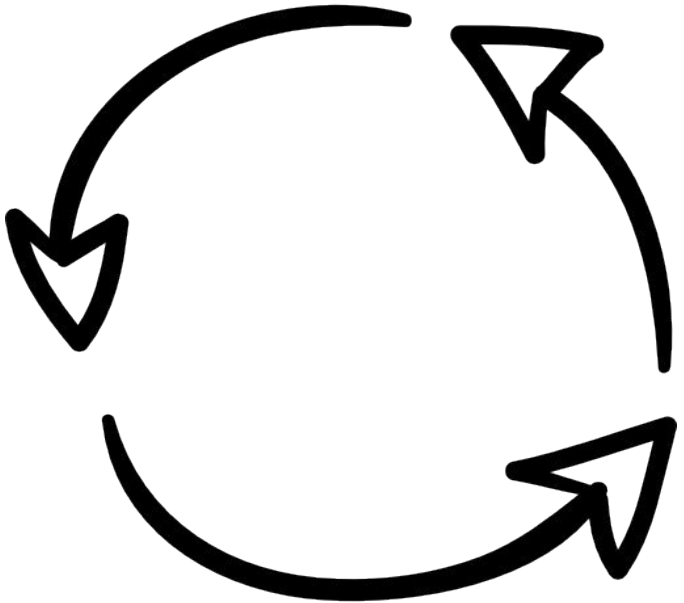
- Las pruebas deben mantenerse.
- Inicialmente, ralentiza el desarrollo.
- Las pruebas unitarias deben adoptarse por todo el equipo.
- Un reto que puede ser intimidante y no es fácil de aprender al comienzo.
- Difícil de aplicar al código heredado existente.

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.

Pruebas unitarias

Beneficios

- Facilita realizar cambios
- Mejora la calidad del código
- Encuentra errores en el diseño
- Proporciona documentación



Ejercicio guiado

Ciclo de vida

Ordena del 1 al 6 las etapas del ciclo de vida de las pruebas unitarias.

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”).	

Gestores en el ciclo de vida

Los gestores son sistemas automatizados que buscan simplificar los procesos de construcción:

- limpiar
- compilar
- generar ejecutables del proyecto a partir del código fuente

*En esta unidad abordaremos **Apache Maven** como gestor.*

Maven

Características

- Núcleo de la aplicación pom.xml
- Describe pom en XML
- Permite gestionar informes y documentación

Añadir dependencia en Maven

Se agregan las librerías adicionales en el archivo pom.xml en la raíz del proyecto.

Dentro de las etiquetas dependencies.

```
<dependencies>

  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.4.2</version>
    <scope>test</scope>
  </dependency>

</dependencies>
```

Conozcamos Maven

Apache Maven

Enlace

¿Qué elementos componen el ciclo de vida de una prueba unitaria?

¿Qué es Maven?

Próxima sesión...

- *Conocer las anotaciones de JUnit para saber cuándo aplicarlas.*
- *Desarrollar pruebas unitarias en un proyecto Java usando características de JUnit.*

{desafío}
latam_

Academia de
talentos digitales