Desarrollo Web Integrado

Sesión 3

Test-drive Development (TDD)

Integración de TDD en Spring

Inicio

Universidad Tecnológica del Perú

Utilidad

Universidad Tecnológica del Perú

¿Qué vimos la sesión anterior?

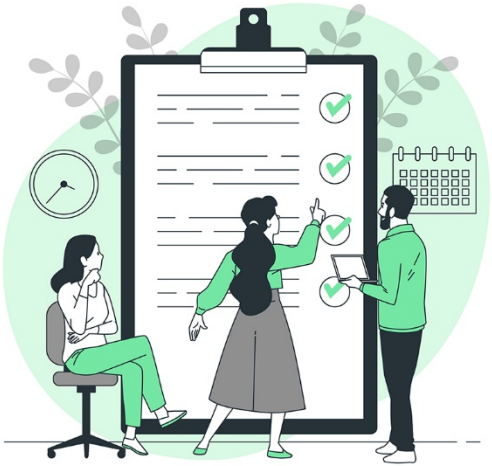
- **Controladores**
  Configuración de endpoints y controladores.
  Dependency injection.

¿Preguntas?

Utilidad

Universidad Tecnológica del Perú

Saberes Previos



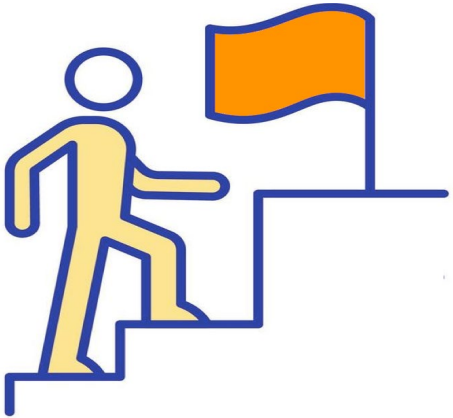
TDD

¿Qué es?

Utilidad

Universidad Tecnológica del Perú

Logro de Aprendizaje



Al finalizar la sesión, el estudiante implementa una API REST usando TDD

Universidad Tecnológica del Perú

Importancia

¿Cuál es la importancia de lo que veremos en la sesión de hoy tanto, para su vida académica como profesional?

Universidad Tecnológica del Perú

¿Qué vamos a ver en la sesión de hoy?

- **API REST con TDD**
  - Pruebas Unitarias
  - TDD
  - Mokito
  - API REST en Spring Boot con TDD

Universidad Tecnológica del Perú

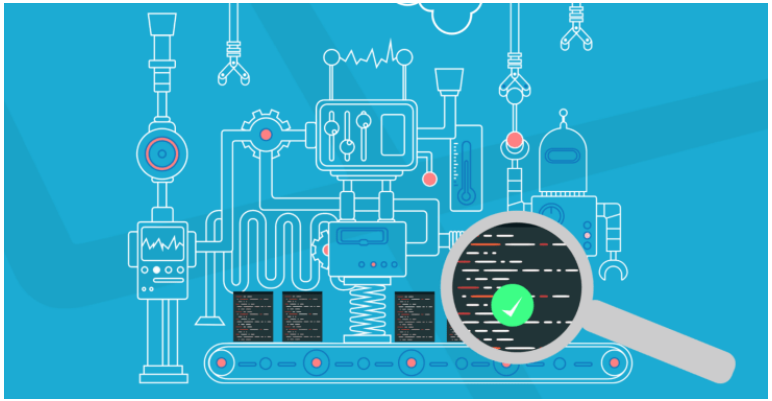
Prueba **UNITARIA**

Pruebas Unitarias

Universidad Tecnológica del Perú

Las **pruebas unitarias** son aquellas cuyo objetivo es **probar partes indivisibles** del software de forma aislada

En un lenguaje orientado a objetos en general y en el lenguaje Java en particular, estas unidades básicas e indivisibles son las clases, por lo tanto **las pruebas unitarias están enfocadas a probar clases**.



Universidad Tecnológica del Perú

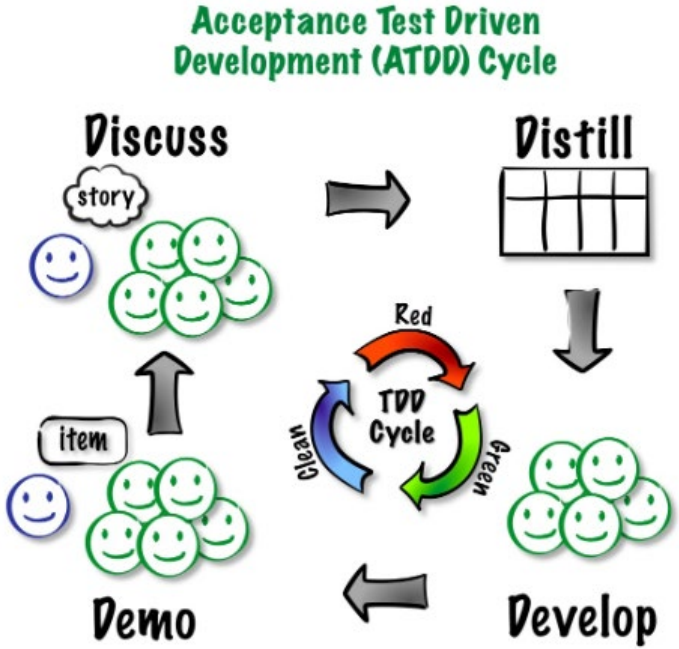
TDD

Test-Driven Development

Desarrollo dirigido por pruebas

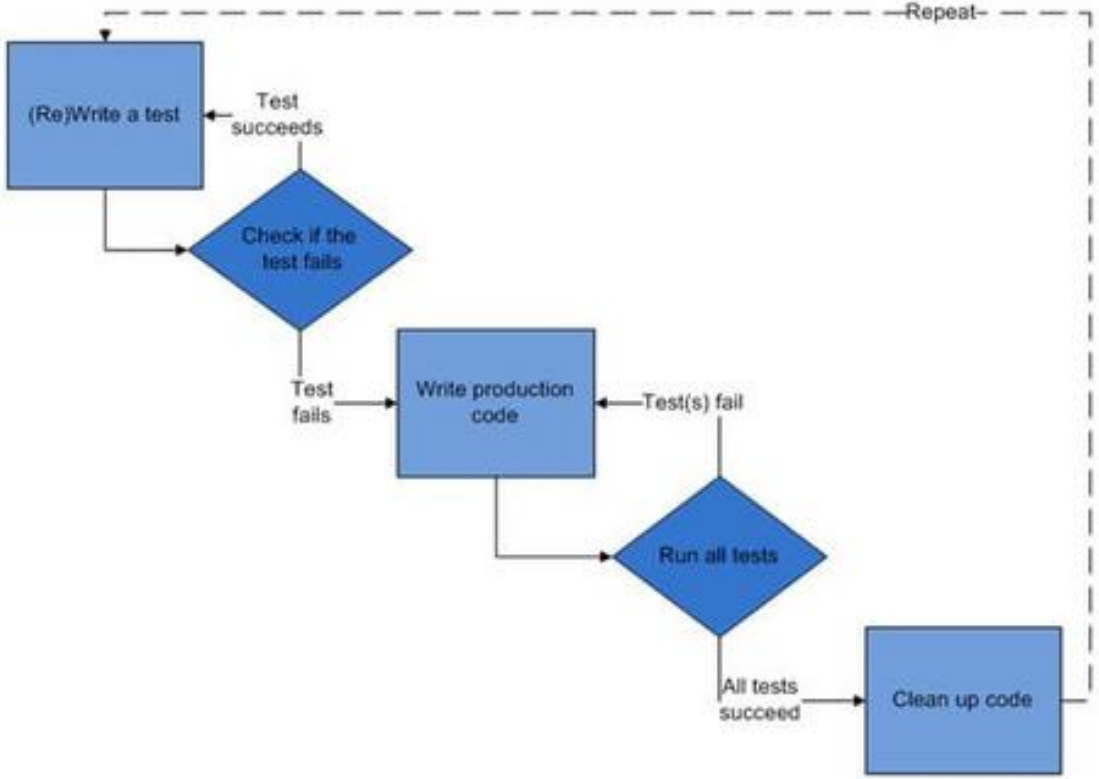
TDD

Es una práctica de programación que consiste en ***escribir primero las pruebas*** (generalmente unitarias), ***después escribir el código fuente que pase la prueba*** correctamente y, por último, refactorizar el código escrito



Transformación

TDD



Este ciclo también se lo conoce como **rojo** (hacer que la prueba falle), **verde** (hacer que la prueba pase) y **refactor**.

Transformación

TDD

1. El cliente escribe su historia de usuario.
2. Se escriben junto con el cliente los criterios de aceptación de esta historia, desglosándolos mucho para simplificarlos todo lo posible.
3. Se escoge el criterio de aceptación más simple y se traduce en una prueba unitaria.
4. Se comprueba que esta prueba falla.
5. Se escribe el código que hace pasar la prueba.
6. Se ejecutan todas las pruebas automatizadas.
7. Se refactoriza y se limpia el código.
8. Se vuelven a pasar todas las pruebas automatizadas para comprobar que todo sigue funcionando.
9. Volvemos al punto 3 con los criterios de aceptación que falten y repetimos el ciclo una y otra vez hasta completar nuestra aplicación.

TDD

Supongamos que el cliente nos pide que desarrollemos una calculadora que sume números

TDD

Universidad Tecnológica del Perú

Acordamos con el cliente que el criterio de aceptación sería que si introduces en la calculadora dos números y le das a la operación de suma, la calculadora te muestra el resultado de la suma en la pantalla.

TDD

Partiendo de este criterio, comenzamos a definir el funcionamiento del algoritmo de suma y convertimos el criterio de aceptación en una prueba concreta, por ejemplo, un algoritmo que si introduces un 3 y un 5 te devuelve un 8:

```
public void testSuma()
{ assertEquals(8, Calculadora.suma(3,5)); }
```

Al escribir el test estoy diseñando cómo va a funcionar el software, pienso que para cubrir la prueba voy a necesitar una clase Calculadora con una función que se llame Suma y que tenga dos parámetros.

Con TDD sólo hacemos lo que realmente necesitamos en ese momento.

TDD

Por supuesto, si intentamos pasar este test nos dará un error, porque la clase Calculadora aún no existe.

TDD

Ahora pasamos a escribir el código de la clase, es fácil porque ya sabemos exactamente cómo se va a comportar:


```
public class Calculadora
{
        public static int suma (int a, int b)
        {
                int c = a + b;
                return c;
        }
}
```

Universidad Tecnológica del Perú

Ahora ejecutamos la prueba y ya tenemos el código funcionado con la prueba pasada.

Una vez todo esté funcionando, pasamos a refactorizar y a eliminar código duplicado, este ejemplo es extremadamente sencillo, y en un caso real no haríamos tantos pasos para algo tan evidente, pero el código mejorado podría ser por ejemplo:


```
public class Calculadora
{
        public static int suma (int a, int b)
        {
                return a + b;
        }
}
```

TDD

Ahora deberíamos volver al punto 3 con tests más complicados y repetir el proceso, por ejemplo, podíamos pasar a que el algoritmo admita sumar números decimales, etc.

FUENTE: https://www.paradigmadigital.com/dev/tdd-como-metodologia-de-diseno-de-software/#:~:text=TDD%20o%20Test%2DDriven%20Development,%C3%BAltimo%2C%20refactorizar%20el%20c%C3%B3digo%20escrito.

Universidad Tecnológica del Perú

Ahora veamos un ejemplo:

¿Cómo implementamos una API REST con TDD?

Práctica

Practiquemos

Intégrate a tu equipo de trabajo conformado en la sesión de clase, de forma aleatoria por la plataforma Zoom, e…

Implementa el controlador para exponer los servicios implementados en el ejemplo trabajado y evalúa los mismos con Postman

Terminado el tiempo, los equipos de trabajo regresan a la sala principal de la plataforma Zoom para compartir el trabajo realizado, a fin de recibir la retroalimentación respectiva que permita el logro del aprendizaje

15 minutos

Cierre

Universidad Tecnológica del Perú

¿Qué aprendiste hoy?

Participa por audio o chat | ✋ Levanta la mano para participar por audio


UTP
Universidad
Tecnológica
del Perú