

# Análisis de Caso

---

Test Driven Development (TDD)

Ashley Rodriguez.

## **Situación Inicial**

Una pequeña empresa llamada EduCode está desarrollando un sistema de gestión de cursos en línea. Hasta ahora, el equipo ha implementado distintas características —creación de cursos, registro de estudiantes, calificaciones, pero los bugs suelen aparecer en las etapas finales o después de lanzamientos. Esto se debe a que las pruebas tradicionalmente se efectúan al concluir el desarrollo, encontrando fallos demasiado tarde. Para mejorar su proceso, la gerencia propone instaurar TDD: en cada funcionalidad nueva se diseñarán primero las pruebas automatizadas, luego se escribirá el código justo para que pasen (Green), y finalmente se refactorizará el código para mantenerlo limpio y eficiente, sin que las pruebas dejen de pasar.

### **1. Plan de Adopción de TDD**

#### **Diferencia con el testing tradicional:**

El TDD es una metodología que se basa en escribir primero las pruebas antes de implementar cualquier funcionalidad. A diferencia del enfoque tradicional, donde las pruebas se hacen después del desarrollo. La fase Red obliga a los desarrolladores a pensar primero en los requisitos y las pruebas que se realizan, esto aclara los requisitos y reduce suposiciones ayudando a diseñar solo lo necesario.

#### **Ventajas del TDD:**

- Detección temprana de errores.
- Diseño modular y mantenible.
- Facilita el Refactor.
- Retroalimentación inmediata.

#### **Limitaciones del TDD:**

- Se requiere más tiempo en las primeras etapas.
- Complejidad en pruebas para integraciones externas.
- se pueden escribir más pruebas de las necesarias.
- Riesgo de falsos positivos o negativos.

### **2. Ciclo Red-Green-Refactor (CalcularPromedio ()).**

- **Red:** Crea una prueba unitaria que falle por ausencia de implementación o retorno incorrecto.

```
Ver  Ir  ...  <  ->  tdd  [pom.xml 1]
J CursoService.java  J CursoServiceTest.java 2
src > test > java > com > example > J CursoServiceTest.java > ...
1  package com.example;
2
3
4  import java.util.Arrays;
5  import java.util.List;
6  import static org.junit.jupiter.api.Assertions.assertEquals;
7  import org.junit.jupiter.api.Test;
8
9  public class CursoServiceTest {
10
11      @Test
12      public void CalcularPromedio(){
13          CursoService cursoService = new CursoService();
14          List<Double> notas = Arrays.asList(7.0, 5.5, 2.3);
15          double promedio = cursoService.calcularPromedio(notas);
16          assertEquals(4.9, promedio, 0.1);
17      }
18  }
19
```

```
[INFO] 2 errors
[INFO] -----
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 1.389 s
[INFO] Finished at: 2025-06-27T02:05:30-04:00
[INFO] -----
on 65193  Java: Ready
```

- **Green:** Codifica únicamente lo necesario para que la prueba pase.

```
Ver  Ir  ...  <  ->  tdd  [CursoServiceTest.java 2]
J CursoServiceTest.java 2  J CursoServiceTest.java 2
src > main > java > com > example > J CursoServiceTest.java > Java > CursoService > calcularPromedio(List<Double> notas)
1  package com.example;
2
3  import java.util.List;
4
5
6  public class CursoService {
7
8      public double calcularPromedio(List<Double> notas) {
9          double suma = 0.0;
10         for (Double nota : notas) {
11             suma += nota;
12         }
13
14         return suma / notas.size();
15     }
16 }
17
```

- **Refactor:** Limpia y mejora el método manteniendo las pruebas en verde.  
\*En **CursoService**, se refactorizó el método **calcularPromedio ()** en donde se redujo el ciclo **for** y se mantiene la lógica funcional, pero con una estructura más mantenible.

```
main / java > com > example > CursoService.java > ...
1 package com.example;
2
3 import java.util.List;
4
5
6 public class CursoService {
7
8     public double calcularPromedio(List<Double> notas) {
9
10         return notas.stream().mapToDouble(Double::doubleValue).average().orElse(0.0);
11     }
12 }
13
14
```

```
PROBLEMAS 8 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS
[INFO] Running com.example.CursoServiceTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.049 s -- in com.example.CursoServiceTest
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.554 s
65193 Java: Ready Lin 20 col
```

### 3. Pruebas Adicionales por Funcionalidad (VerificarCupos ()).

- **Red:** Crea una prueba unitaria que falle por ausencia de implementación o retorno incorrecto

```

> test > java > com > example > J VerificarCuposTest.java > ...
1  package com.example;
2
3  import static org.junit.jupiter.api.Assertions.assertFalse;
4  import static org.junit.jupiter.api.Assertions.assertTrue;
5  import org.junit.jupiter.api.Test;
6  public class VerificarCuposTest {
7
8      @Test
9
10     void CupoDisponible() {
11         VerificarCupos verificarCupos = new VerificarCupos();
12         boolean resultado = verificarCupos.verificarCupos(40,35);
13         assertTrue(resultado, " Hay Cupos Disponibles ");
14     }
15
16     @Test
17     void CupoNoDisponible() {
18         VerificarCupos verificarCupos = new VerificarCupos();
19         boolean resultado = verificarCupos.verificarCupos(40,46);
20         assertFalse(resultado, " No hay Cupos Disponibles ");
21     }
22 }

```

```

[INFO] Changes detected - recompiling the module!
[INFO] Compiling 2 source files to C:\Users\ashle\Downloads\tdd\target\classes
[INFO] -----
[ERROR] COMPILATION ERROR :
[INFO] -----
[ERROR] /C:/Users/ashle/Downloads/tdd/src/main/java/com/example/VerificarCupos.java:[5,12] class, interface, or enum expected
[ERROR] /C:/Users/ashle/Downloads/tdd/src/main/java/com/example/VerificarCupos.java:[7,5] class, interface, or enum expected
[INFO] 2 errors
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 1.241 s
[INFO] Finished at: 2025-06-27T02:35:44-04:00
[INFO] -----
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.8.1:compile (default-compile) on project tdd: Compilation failure: Compilation failure:
[ERROR] /C:/Users/ashle/Downloads/tdd/src/main/java/com/example/VerificarCupos.java:[5,12] class, interface, or enum expected
[ERROR] /C:/Users/ashle/Downloads/tdd/src/main/java/com/example/VerificarCupos.java:[7,5] class, interface, or enum expected
[ERROR] > Build failed

```

- **Green:** Codifica únicamente lo necesario para que la prueba pase.

```

src > main > java > com > example > J VerificarCupos.java > ...
1  package com.example;
2
3
4  public class VerificarCupos {
5      public boolean verificarCupos(int cuposMax, int cuposOcupados) {
6          return cuposOcupados < cuposMax;
7      }
8
9  }
10

```

- **Refactor:** Limpia y mejora el método manteniendo las pruebas en verde.

\* el método ya está limpio, corto y claro, así que no hace falta refactorizar.

```

[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running com.exampleCursoServiceTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.053 s -- in com.exampleCursoServiceTest
[INFO] Running com.example.VerificarCuposTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.011 s -- in com.example.VerificarCuposTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time: 2.718 s
[INFO] Finished at: 2025-06-27T02:36:21-04:00
[INFO]
[INFO] -----

```

## 4. Cubriendo Escenarios de Error

- Test de caso límite datos nulos.

```

src > test > java > com > example > J CursoServiceNullTest.java > Java > CursoServiceNullTest
1  package com.example;
2
3  import java.util.List;
4
5  import static org.junit.jupiter.api.Assertions.assertThrows;
6  import org.junit.jupiter.api.Test;
7
8  public class CursoServiceNullTest {
9
10     @Test
11     public void promedioListaNula_lanzaExcepcion() {
12         CursoServiceNull curso = new CursoServiceNull();
13         List<Double> notas = null;
14
15         assertThrows(IllegalArgumentException.class, () -> {
16             curso.calcularPromedio(notas);
17         }, " No puede ser nula");
18     }
19 }

```

```

src > main > java > com > example > J CursoServiceNull.java > Java > CursoServiceNull
1  package com.example;
2
3
4  import java.util.List;
5
6  public class CursoServiceNull {
7
8      public double calcularPromedio(List<Double> notas) {
9          if (notas == null) {
10              throw new IllegalArgumentException("No puede ser nula");
11          }
12          return notas.stream().mapToDouble(Double::doubleValue).average().orElse(0.0);
13      }
14 }

```

```
Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider

-----
T E S T S
-----
Running com.example.CursoServiceNullTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.054 s -- in com.example.CursoServiceNullTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

-----
BUILD SUCCESS
-----
Total time: 2.167 s
Finished at: 2025-06-27T03:08:25-04:00
-----
```

## 5. Organización y Ejecución en Visual Studio Code.

Para llevar a cabo el análisis de TDD, se utilizó Visual Studio Code como entorno de desarrollo, integrando las herramientas de Maven y JUnit para ejecutar pruebas de forma automatizada.

Para configurar el entorno:

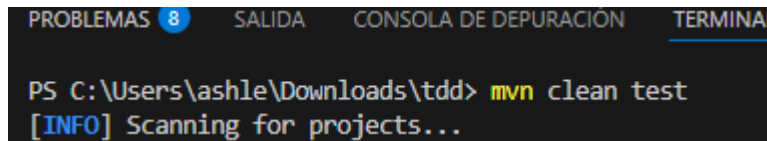
1. Se añadió la dependencia de **JUnit 5** al archivo pom.xml, para escribir y ejecutar las pruebas unitarias.

```
<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.10.0</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

2. Se configuró el compilador de Maven como gestor de dependencias y compilación. para usar Java 11 que es el que sirve para la compilación:

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>11</maven.compiler.source>
  <maven.compiler.target>11</maven.compiler.target>
</properties>
```

3. Las pruebas se ejecutaron usando el comando “**mvn clean test**” en la terminal:



```
PROBLEMAS 8 SALIDA CONSOLA DE DEPURACION TERMINAL
PS C:\Users\ashle\Downloads\tdd> mvn clean test
[INFO] Scanning for projects...
```

#### 4. Extensiones utilizadas en VSC:

**\*Extension Pack for Java** (Microsoft): Incluye Language Support for Java, Maven for Java, Java Test Runner, entre otros.

Esto permitió:

- Identificar errores desde el inicio con la metodología (TDD).
- Garantizar que ninguna modificación rompiera el diseño ya validado.
- Realizar pruebas para comprobar que todo funcionara con el método red-Green-Refactor.

Gracias a esta configuración, se mantuvo un ciclo continuo de prueba y desarrollo, reduciendo errores, se garantizó una integración más segura y se impulsó la mejora progresiva del sistema.

## **6. Reflexión Final**

El enfoque del TDD tuvo un impacto visible en el diseño del sistema. Al escribir primero las pruebas, cada nueva funcionalidad se diseñó pensando en cómo se espera que se comporte, lo que evitó confusiones y errores por malinterpretación. automatizar las pruebas desde el inicio reduce tiempo en las etapas finales y con las herramientas instaladas como Maven y VSC se pudo implementar el TDD de forma eficiente.

La práctica del ciclo Red-Green-Refactor permitió construir el sistema en pequeños pasos seguros, donde cada refactorización fue acompañada por pruebas que garantizaban que no hubiera errores.