Pruebas Unitarias

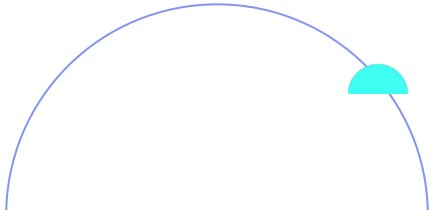
Pruebas Unitarias

Son **pruebas automatizadas** que verifican la funcionalidad de unidades individuales de código, como funciones o métodos.

Su objetivo es detectar errores, en una etapa temprana del desarrollo, para asegurar que cada unidad de código funcione como se espera.

Características

- **Mismo lenguaje:** Las pruebas unitarias se escriben en el mismo lenguaje de programación que el código que están probando.
- **Bibliotecas especializadas:** Utilizan bibliotecas específicas del lenguaje para facilitar la creación y ejecución de las pruebas.
- **Integración directa:** Las pruebas se integran directamente con el código fuente. Esto permite probar funciones, métodos y clases de forma aislada.
- **Automatización:** Proveen una manera automatizada de verificar que el código se comporte como se espera. Comparan salidas reales con resultados esperados.



Beneficios de las Pruebas Unitarias

Detección temprana de errores

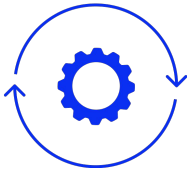
Facilitan la identificación de errores pronto en el ciclo de desarrollo.

Refactorización segura

Permiten realizar cambios en el código con confianza.

Documentación

Proporcionan ejemplos prácticos de uso del código.



Mantenimiento

Mejoran la mantenibilidad del *software* al asegurarse de que los cambios no introduzcan errores.

Frameworks de Pruebas Unitarias

Cada lenguaje de programación suele tener un *framework* de pruebas unitarias asociado que facilita la creación, ejecución y gestión de pruebas automatizadas. Este entorno ayuda a los desarrolladores a **verificar la funcionalidad del código y detectar errores** tempranamente.

Además de las bibliotecas suelen existir **entornos para ejecutar las pruebas** y visualizar sus resultados

Bibliotecas comunes	
Python	unittest, pytest.
Java	JUnit.
C#	NUnit.
JavaScript	Jest, Mocha.

Pruebas Unitarias en Python

Framework: unittest

python Copiar código

```python
import unittest

def sumar(a, b):
    return a + b

class TestSumar(unittest.TestCase):
    def test_suma(self):
        self.assertEqual(sumar(2, 3), 5)

if __name__ == '__main__':
    unittest.main()
```

Pruebas Unitarias en Java

Framework: JUnit

java Copiar código

```java
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class SumarTest {
    @Test
    public void testSuma() {
        assertEquals(5, Sumar.suma(2, 3));
    }
}
```

Pruebas Unitarias en C#

Framework: NUnit

csharp Copiar código

```csharp
using NUnit.Framework;

public class SumarTests {
    [Test]
    public void TestSuma() {
        Assert.AreEqual(5, Sumar.Suma(2, 3));
    }
}
```

Pruebas Unitarias en JavaScript

Framework: Jest

javascript Copiar código

```javascript
function sumar(a, b) {
    return a + b;
}

test('suma de 2 + 3 es igual a 5', () => {
    expect(sumar(2, 3)).toBe(5);
});
```

Mejores prácticas en Pruebas Unitarias

- **Aislamiento:** Pruebas independientes entre sí.
- **Nombres descriptivos:** Claridad en la intención de la prueba.
- **Cobertura:** Asegurarse de cubrir todos los casos posibles.
- **Frecuencia:** Ejecutar pruebas con cada cambio en el código.

Test-Driven Development (TDD)

TDD es una **metodología de desarrollo donde se escriben primero las pruebas unitarias antes de la implementación del código.**

Proceso

Sigue un ciclo de "Rojo-Verde-Refactor":

- **Rojo:** Escribir una prueba que falle (porque la funcionalidad no está implementada aún).
- **Verde:** Implementar la funcionalidad mínima necesaria para que la prueba pase.
- **Refactor:** Mejorar el código mientras se asegura que las pruebas sigan pasando.