

2



# T5: TDD

Escribir TEST = Entender Código = Identif Fallos Mejor

## Recomendaciones

### Usa el Compilador

- Arregla los errores y Advertencias

### assert/check por test

- Si un test falla → sbt en q exactamente
- Si tienes 2 assert → no sbt cual es el q ha fallado

### KISS (Keep It Simple Stupid)

- lo más simple para q pase el test
- DRY - Don't Repeat Yourself
- OACO → Only And Once only (Duplicados)

### fake it (Till you Make it)

- PRIMER test no te sale → return 0
- poco a poco vas mejorando la salida

#### Ejemplo implementación PyUnit:

```
* First version
return 0

* Second version
return self.assertTrue(0 == self.failCount)

* Third version
return self.assertTrue(0 == self.failCount, self.failMessage)
```

### Triangulate

Abstrae solo cuando tengas 2 o más ejemplos

```
import unittest
class TestClass(unittest.TestCase):
    def test_plus(self):
        self.assertEqual(1, plus(1,1))

def plus(augend, addend):
    return 0

import unittest
class TestClass(unittest.TestCase):
    def test_plus(self):
        self.assertEqual(1, plus(1,1))
        self.assertEqual(1, plus(1,1))
```

### Implementación Obviamente

- Take it o Triangulación
- Si sabes q escribir y es rápido → hazlo

### Después del Primer Ciclo tienes...

- 1) Lista de test con los q necesitas trabajar
- 2) Historia contada sobre como se va a ver una operación
- 3) Los test compilan con "Codes" return 4
- 4) Los test corren, incluso con guardados de código
- 5) Gradualmente generalizar el código, cts vs variables 4 ≈ 96%
- 6) Añade los items a una TD-DO-List

### Refactorizar

- No puede cambiar la semántica del programa
- "Observación equivalente"
  - ↳ todos los test q pasaban ANTES del refactor → tienen q pasar DESPUÉS
  - ↳ debes ver cuantos fallos tenían los test al refactorizar

#### Niveles de Escala

- ↳ 2 Buclez son  $\Sigma$  =  $\Delta$  y merge
- ↳ 2 Rama de condicionales son  $\exists$  =  $\Delta$  testea = y elimina una
- ↳ 2 Métodos  $\Delta$  =  $\Delta$  y elimina
- ↳ 2 Clases  $\Delta$  =  $\Delta$  y elimina

## Resumen

- ① TDD no reemplaza el Sist stand. de Testing
  - Define una forma probada y efectiva de hacer test unitarios
  - Test son ejemplos de como funciona
  - Ningún código debe ir a producción si no tiene un test
- ② TDD no es nuevo
- ③ TDD = Menos θ de debuggando → entender código = tener errores fáciles
- ④ TDD niega el miedo
  - ↳ miedo →  $\nabla PC$  + Basta q le hagan muchos test
- ⑤ TDD crea → conjunto de "Test programadores"
  - ↳ test automatizados → El por el API
  - ↳ exhaustivo: Necesitar uno y otra vez
- ⑥ TDD Permite Refactor sin Miedo
- ⑦ (con cuidado) se pueden crear test de validación User Acceptance Test