

Grado en Ingeniería del Software
Doble Grado en Matemática Computacional e Ingeniería del Software
Doble Grado en Física Computacional e Ingeniería de Software

Verificación de Software



Práctica Técnicas TDD (Test Driven Development)

Alonso Álvarez García
Rafael Socas Gutiérrez

Curso 2023/24



Datos de los alumnos

| # | Nombre y apellidos | Curso |
|---|-----------------------|---------|
| 1 | Natalia García Huerta | INSO 4A |
| 2 | Lara Loira Garrido | MAIS 4A |
| 3 | Chantal López Cervera | INSO 4B |
| 4 | Javier Orti García | MAIS 4A |
| 5 | | |

Tarea 1: Preparación del entorno

1
Punto

- Nos apoyaremos en el entorno de desarrollo (IDE) Visual Studio Code y en el intérprete Python 3. Para instalarlo se seguirá el siguiente tutorial <https://code.visualstudio.com/docs/python/python-tutorial>.
- Crear una carpeta con donde se guardarán el código a testear y los tests correspondientes. Los ficheros iniciales a incluir en esa carpeta se aportan junto con este enunciado
- En el IDE, abrir la carpeta y crear un entorno virtual (Ctrl+Shift+P) de tipo Venv (ver tutorial anterior).
- Instalar el runner nose que mejora la interfaz con el usuario al realizar los test
> `pip install pynose`
- Incluir el módulo pinocchio permite colorear el resultado de los test y así ayudar al diagnóstico
> `pip install pinocchio`
- Por último, instalar módulo coverage que nos ayudará a conocer el número de líneas que se testean
> `pip install coverage`

```
PS C:\Users\Lara\Desktop\Uni\Cuarto\Segundo Cuatrimestre\Verificación de Software\Prácticas\Práctica 3> pip install pynose
Collecting pynose
  Downloading pynose-1.5.1-py3-none-any.whl.metadata (19 kB)
    Downloading pynose-1.5.1-py3-none-any.whl (116 kB)
      ━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 1.4 MB/s eta 0:00:00
Installing collected packages: pynose
Successfully installed pynose-1.5.1
PS C:\Users\Lara\Desktop\Uni\Cuarto\Segundo Cuatrimestre\Verificación de Software\Prácticas\Práctica 3> pip install pinocchio
Collecting pinocchio
  Downloading pinocchio-0.4.3-py3-none-any.whl.metadata (975 bytes)
Requirement already satisfied: colorama in c:\users\lara\appdata\roaming\python\python310\site-packages (from pinocchio) (0.4.5)
  Downloading pinocchio-0.4.3-py3-none-any.whl (12 kB)
Installing collected packages: pinocchio
Successfully installed pinocchio-0.4.3
Collecting coverage
  Downloading coverage-7.4.4-cp310-cp310-win_amd64.whl.metadata (8.4 kB)
  Downloading coverage-7.4.4-cp310-cp310-win_amd64.whl (209 kB)
    ━━━━━━━━━━━━━━━━━━━ 209.2/209.2 kB 4.2 MB/s eta 0:00:00
Installing collected packages: coverage
Successfully installed coverage-7.4.4
```

Código a testear y robustecer

Se pretende testear y robustecer el código Python de un módulo unitario de la aviónica de un Airbus A350. Este módulo, tiene la función de calcular el alcance del radar meteorológico del avión. Con la información obtenida, se presentan la información a escala con la ruta de vuelo en la cabina del avión. El alcance de del radar meteorológico se calcula de la siguiente forma:

$$\text{Alcance} = Co * (T - \tau) / 2$$

Donde:

- *Alcance*: Alcance del radar meteorológica en [km]
- *Co*: velocidad de la luz $3 \cdot 10^5$ [km/s].
- *T*: intervalo de repetición de pulsos [s]. Su rango va de 0 a 0.7 s.
- *tau*: ancho del pulso [μ s]. Su rango va de 0 a 4 μ s.
- IMP: *T* siempre tiene que ser mayor que *tau*.

Siguiendo la filosofía TDD se parte de un código extremadamente simple que no cumpla casi ningún test, solo lo esencial. El fichero se aporta con el enunciado de la práctica

radar_meteorologico.py

```
def alcance_del_radar(T: float, tau: float) -> float:
    """Calcula el alcance del radar meteorológico"""
    """ entrada: T, intervalo de repetición de pulsos [segundos]"""
    """ Entrada: tau, ancho del pulso [microsegundos]"""
    """ Salida: Alcance del radar meteorológico [kilómetros]"""

    #se ponen todas las unidades en segundos, solo en necesario tau,
    #T ya lo está
    tau=tau/pow(10,6)

    Co=3*pow(10,5) #Velocidad de la luz, 300.000 km/s
    return Co*(T-tau)/2 # Calculo del alcance de radar
```

Tarea 2: Test a realizar

1
Punto

Recuerde que la filosofía de TDD se basa en desarrollar el código en base a que superen los test propuestos. En esa línea, cuantos más números y más variados sean los test, más robusto será el código resultante.

Aquí se aportan dos test (1 y 4), se proponen 18 (16 por definir) y 6 categorías de test. Complete esta tabla, e incluso si amplía en el número de test y categorías propuestas será valorado.

| Número test | Rangos | | Alcance (km) | Categoría del test |
|-------------|--------------|---------------------|--------------|----------------------------------|
| | [0-0.7] | [0-4] | | |
| | T (segundos) | tau (microsegundos) | | |
| 1 | 0.5 | 2 | 74999.700 | Valores válidos |
| 2 | | | | |
| 3 | | | | |
| 4 | 0.2 | 5 | No valida | Valores positivos fuera de rango |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | Valores negativos |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | T menor que tau |
| 11 | | | | |
| 12 | | | | |
| 13 | | | | Entrada de strings |
| 14 | | | | |
| 15 | | | | |
| 16 | | | | |
| 17 | | | | Entrada de booleanos |
| 18 | | | | |

| Número test | Rangos | | Alcance (km) | Categoría del test |
|--------------|---------------------|-------------------|--------------|----------------------------------|
| | [0-0,7] | [0-4] | | |
| T (segundos) | tau (microsegundos) | | | |
| 1 | 0,5 | 2 | 74999,7 | Valores válidos |
| 2 | 0.6 | 1 | 89999.85 | |
| 3 | 0.3 | 0.5 | 44.999.925 | |
| 4 | 0,2 | 5 | No válida | Valores positivos fuera de rango |
| 5 | 0.8 | 2.0 | No válida | |
| 6 | 0.5 | 5.0 | No válida | |
| 7 | -0.1 | 2.0 | No válida | Valores negativos |
| 8 | 0.5 | -1.0 | No válida | |
| 9 | -0.1 | -1.0 | No válida | |
| 10 | 0.000002 | 3.0 | No válida | T menos que tau |
| 11 | 0.000005 | 6.0 | No válida | |
| 12 | 0.5 | "2.0" | No válida | Entrada de strings |
| 13 | "0.5" | 2.0 | No válida | |
| 14 | "0.5" | "2.0" | No válida | |
| 15 | "letra" | "palabra" | No válida | |
| 16 | "123" | "no es un numero" | No válida | |
| 17 | True | 0.5 | No válida | Entrada de booleanos |
| 18 | 0.5 | True | No válida | |

Tareas 3: Ejecución de los test, análisis de resultados y aplicación filosofía TDD

6
Puntos

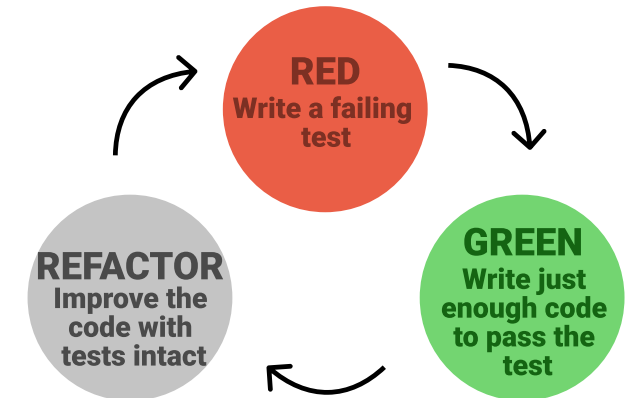
Una vez, se tiene un código base muy sencillo y un conjunto de test, ahora los pasos con técnicas TDD son:

Con un fichero de test, **test_radar_meteorologico.py** (que se proporciona y ya tiene implementado el test 1 y 4) ejecute los test para ver los resultados obtenidos.

1. Los test se ejecutan con los siguientes comandos Python:

```
> python -m nose -v test_radar_meteorologico.py --with-spec --spec-color --with-coverage  
> coverage report -m
```
2. Programe los test en el fichero **test_radar_meteorologico.py**, para ello apóyese en los métodos **assert** de **unittest** (Python unittest Assert Methods) disponibles en <https://www.pythontutorial.net/python-unit-testing/python-unittest-assert/>
3. Modifique el código de **radar_meteorologico.py** para conseguir que se ejecuten todos los test de manera correcta y que todas las líneas de código sean analizadas (comando **coverage report -m**).
4. Vuelva al 1 hasta que un 100% de test superados en el punto 2.

Test Driven Development (TDD)



Test Unitarios


```
def alcance_del_radar(T: float, tau: float) -> float:
    """Calcula el alcance del radar meteorológico"""
    """ entrada: T, intervalo de repetición de pulsos [segundos]"""
    """ Entrada: tau, ancho del pulso [microsegundos]"""
    """ Salida: Alcance del radar meteorológico [kilómetros]"""

    if isinstance(T, bool) or isinstance(tau, bool):
        raise TypeError("T y tau no pueden ser valores booleanos.")
    if not isinstance(T, (int, float)) or not isinstance(tau, (int, float)):
        raise TypeError("T y tau deben ser números.")

    #se ponen todas lass unidades en segundos, solo en necesario tau, T ya lo está
    tau=tau/pow(10,6)

    Co=3*pow(10,5) #Velocidad de la luz, 300000 km/s

    if T < 0 or T > 0.7:
        raise ValueError("T debe estar entre 0 y 0.7 segundos.")
    if tau < 0 or tau > 4e-6:
        raise ValueError("tau debe estar entre 0 y 4 microsegundos.")

    if T <= tau:
        raise ValueError("T siempre tiene que ser mayor que tau.")

    return Co*(T-tau)/2 # Calculo del alcance de radar
```

```
from unittest import TestCase
from radar_meteorologico import alcance_del_radar

class TestRadarMeteorologico(TestCase):

    def test_valores_validos(self):
        """ Test de valores validos """
        self.assertAlmostEqual(alcance_del_radar(0.5, 2), 74999.700)
        self.assertAlmostEqual(alcance_del_radar(0.6, 1), 89999.85)
        self.assertAlmostEqual(alcance_del_radar(0.3, 0.5), 44999.925)

    def test_valores_fuera_rango(self):
        """ Test ValueError cuando hay valores positivos fuera de rango """
        self.assertRaises(ValueError, alcance_del_radar, 0.2, 5.0)
        self.assertRaises(ValueError, alcance_del_radar, 0.8, 2.0)
        self.assertRaises(ValueError, alcance_del_radar, 0.2, 5.0)

    def test_valores_negativos(self):
        """ Test ValueError cuando hay de valores negativos """
        self.assertRaises(ValueError, alcance_del_radar, -0.1, 2.0)
        self.assertRaises(ValueError, alcance_del_radar, 0.5, -1.0)
        self.assertRaises(ValueError, alcance_del_radar, -0.1, -1.0)

    def test_T_menor_tau(self):
        """ Test ValueError cuando T es menor que tau """
        self.assertRaises(ValueError, alcance_del_radar, 0.000002, 3.0)
        self.assertRaises(ValueError, alcance_del_radar, 0.000005, 6.0)
```

```
def test_strings(self):
    """ Test TypeError cuando hay entrada de strings """
    self.assertRaises(TypeError, alcance_del_radar, 0.5, "2.0")
    self.assertRaises(TypeError, alcance_del_radar, "0.5", 2.0)
    self.assertRaises(TypeError, alcance_del_radar, "0.5", "2.0")
    self.assertRaises(TypeError, alcance_del_radar, "letra", "palabra")
    self.assertRaises(TypeError, alcance_del_radar, "123", "no es un número")

def test_booleanos(self):
    """ Test TypeError cuando hay entrada de booleanos """
    self.assertRaises(TypeError, alcance_del_radar, True, 2.0)
    self.assertRaises(TypeError, alcance_del_radar, 0.5, True)
```

Práctica: Técnicas TDD (Test Driven Development)

```
(base) PS C:\Users\chant\OneDrive\Escritorio\Practica3TDD> python -m nose -v test_radar_meteorologico.py --with-spec --spec-color --with-coverage
>> coverage report -m
>>
```

Radarmeteorologico

- Test ValueError cuando T es menor que tau
- Test TypeError cuando hay entrada de booleanos
- Test TypeError cuando hay entrada de strings
- Test ValueError cuando hay valores positivos fuera de rango
- Test ValueError cuando hay de valores negativos
- Test de valores validos

| Name | Stmts | Miss | Cover |
|------------------------|-------|------|-------|
| radar_meteorologico.py | 17 | 0 | 100% |
| TOTAL | 17 | 0 | 100% |

Ran 6 tests in 0.006s


OK

| Name | Stmts | Miss | Cover | Missing |
|-----------------------------|-------|------|-------|---------|
| radar_meteorologico.py | 17 | 0 | 100% | |
| test_radar_meteorologico.py | 27 | 0 | 100% | |
| TOTAL | 44 | 0 | 100% | |


(base) PS C:\Users\chant\OneDrive\Escritorio\Practica3TDD>

```
(base) PS C:\Users\chant\OneDrive\Escritorio\Practica3TDD> coverage report -m
>>
Name                               Stmts  Miss  Cover   Missing
-----
radar_meteorologico.py             17      0   100%
test_radar_meteorologico.py         27      0   100%
-----
TOTAL                             44      0   100%
(base) PS C:\Users\chant\OneDrive\Escritorio\Practica3TDD> |
```



 Calle Playa de Liencres, 2 bis
(entrada por calle Rozabella)
Parque Europa Empresarial
Edificio Madrid
28290 Las Rozas, Madrid

 900 373 379  info@u-tad.com

 [SOLICITA MÁS INFORMACIÓN](#)



CENTRO ADSCRITO A:



PROYECTO COFINANCIADO POR:

