Міністерство освіти і науки України
Національний Технічний Університет України
«Київський Політехнічний Інститут»
Навчально-науковий комплекс
«Інститут прикладного системного аналізу»
Кафедра системного проектування

Лабораторна робота №5

з дисципліни

«Проектування інформаційних систем»

Модульне тестування (Unit-тести) та рефакторинг

Виконав:
студент групи ДА-72
Коваль С.С.

Київ – 2020

Мета роботи: оволодіти навичками створення програмного забезпечення за метолологією TDD та ознайомитися з процедурами рефакторингу.

Задача:

1. Використовувати методологію Test Driven Development для створення класів архітектурної програмної моделі.

2. Скласти тестові сценарії, які продемонструють функціювання всіх методів проектованої моделі.

3. Виконати юніт-тестування складових частин (внутрішніх класів), що реалізують об'єкт моделювання.

4. Виконати "зовнішнє" юніт-тестування для API.

5. Провести рефакторинг коду програми, для поліпшення реалізації.Завдання

Хід роботи

Для написання тестів було використано мову Python та модуль unittest.

Для використання unittest тестів достатьо підключити модуль unittest

```
import unittest
```

Далі можемо переходили до створення програми.

Зробимо примітивні тести для арифметичних функцій : додавання, віднімання, ділення та множення.

```
import unittest
import calc


class CalcTest(unittest.TestCase):
    def test_add(self):
        self.assertEqual(calc.add(1, 2), 3)

    def test_sub(self):
        self.assertEqual(calc.sub(4, 2), 2)

    def test_mul(self):
        self.assertEqual(calc.mul(2, 5), 10)

    def test_div(self):
        self.assertEqual(calc.div(8, 4), 2)


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

calc.py

```
def add(a, b):
    return a + b


def sub(a, b):
    return a - b


def mul(a, b):
    return a * b


def div(a, b):
    return a / b
```

```
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK

Process finished with exit code 0
```

Змінимо тест, щоб спрацьовував assert

```
def test_div(self):
    self.assertEqual(calc.div(8, 4), 3)
```

```
.F..
======================================================================
FAIL: test_div (__main__.CalcTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\Sergey\PycharmProjects\calcunit\main.py", line 16, in test_div
    self.assertEqual(calc.div(8, 4), 3)
AssertionError: 2.0 != 3

----------------------------------------------------------------------
Ran 4 tests in 0.001s

FAILED (failures=1)
```

Рефакторинг

Наприклад, створимо функцію порівняння двох чисел a та b:

```
def greaterThan(a, b):
    if(a == b):
        return False
    if (a < b):
        return False
    else
        return True
```

Ця функцыя виконує свою задачу, але не оптимальным шляхом. Можна її переписати ось так:

```
def greaterThan(a, b):
    return a > b
```

В такому разі вона буде виконувати ту саму дію, але коду в рази менше і він більш зрозумілий. В цьому і складається ідея рефакторингу.

Висновки

Під час виконання даної лабораторної роботи, а також вивчення теоретичної частини для її розуміння, було досліджено unit тестування коду на мові програмування Python. Також вивчено різницю між стандартним підходом та Test Driven Development, коли тест пишеться до написання коду методу. Також було вивчено основні проблеми в коді, які потребують рефакторингу, а також самі методи рефакторингу.