МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”

Інститут прикладного системного аналізу

кафедра системного проектування

Лабораторна робота №5

з дисципліни «Проектування інформаційних систем»

на тему *«Модульне тестуваня (Unit-тести) та рефакторинг»*

Виконала:

студентка 4 курсу

групи ДА-72

Грищенко О.Ю.

Київ – 2020

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

Завдання:

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.

Хід виконання лабораторної роботи:

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

Для демонстрації використання методологію Test Driven Development для створення класів архітектурної програмної моделі розробимо простий калькулятор. Для початку створимо тест для додавання двох чисел:

```
1  import unittest
2  from Calculator import Calculator
3
4  class TddTest(unittest.TestCase):
5
6     def test_add_method_result(self):
7         calc = Calculator()
8         result = calc.add(2,2)
9         self.assertEqual(4, result)
10
11 if __name__ == '__main__':
12     unittest.main()
```

Створимо клас Calculator із методом додавання із заглушкою:

```
1  class Calculator(object):
2      def __init__(self):
3          pass
4      def add(self, x, y):
5          pass
```

Запустимо тест:

```
In [6]: runfile('C:/Users/helen/OneDrive/Рабочий стол/пары/ПІС/untitled0.py', wdir='C:/Users/
helen/OneDrive/Рабочий стол/пары/ПІС')
Reloaded modules: Calculator
F
======================================================================
FAIL: test_add_method_result (__main__.TddTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\helen\OneDrive\Рабочий стол\пары\ПІС\untitled0.py", line 9, in
test_add_method_result
    self.assertEqual(4, result)
AssertionError: 4 != None

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
```

Зробимо перший ‘baby step’:

```
1    class Calculator(object):
2        def __init__(self):
3            pass
4        def add(self, x, y):
5            return x+y
6
```

Запустимо тест:

```
In [7]: runfile('C:/Users/helen/OneDrive/Рабочий стол/пары/ПІС/untitled0.py', wdir='C:/Users/
helen/OneDrive/Рабочий стол/пары/ПІС')
Reloaded modules: Calculator
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Змінимо в класі тесту очікуваний результат додавання та запустимо тест:

```
In [8]: runfile('C:/Users/helen/OneDrive/Рабочий стол/пары/ПІС/untitled0.py', wdir='C
helen/OneDrive/Рабочий стол/пары/ПІС')
Reloaded modules: Calculator
F
======================================================================
FAIL: test_add_method_result (__main__.TddTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\helen\OneDrive\Рабочий стол\пары\ПІС\untitled0.py", line 9, in
test_add_method_result
    self.assertEqual(5, result)
AssertionError: 5 != 4

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
```

Зробимо другий ‘baby step’ та перевіримо що буде, якщо ввести два слова замість чисел:

```python
import unittest
from Calculator import Calculator

class TddTest(unittest.TestCase):
    def setUp(self):
        self.calc = Calculator()

    def test_add_method_returns_correct_result(self):
        result = self.calc.add(2, 2)
        self.assertEqual(4, result)

    def test_calculator_returns_if_both_not_numbers(self):
        self.assertRaises(ValueError, self.calc.add, 'two', 'three')
```

```
In [13]: runfile('C:/Users/helen/OneDrive/Рабочий стол/пары/ПІС/untitled0.py', wdir='C:/Users/
helen/OneDrive/Рабочий стол/пары/ПІС')
F
======================================================================
FAIL: test_calculator_returns_error_message_if_both_args_not_numbers (__main__.TddTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\helen\OneDrive\Рабочий стол\пары\ПІС\untitled0.py", line 9, in
test_calculator_returns_error_message_if_both_args_not_numbers
    self.assertRaises(ValueError, self.calc.add, 'two', 'three')
AssertionError: ValueError not raised by add

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
```

Змінимо відповідно клас Калькулятора:

```python
class Calculator(object):
    def __init__(self):
        pass

    def add(self, x, y):
        number_types = (int, float, complex)

        if isinstance(x, number_types) and isinstance(y, number_types):
            return x + y
        else:
            raise ValueError
```

```
In [14]: runfile('C:/Users/helen/OneDrive/Рабочий стол/пары/ПІС/untit
helen/OneDrive/Рабочий стол/пары/ПІС')
Reloaded modules: Calculator
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Аналогічно робимо якщо одне число із двох є слово:

```python
import unittest
from Calculator import Calculator

class TddTest(unittest.TestCase):
    def setUp(self):
        self.calc = Calculator()

    def test_add_method_returns_correct_result(self):
        result = self.calc.add(2, 2)
        self.assertEqual(4, result)

    def test_calculator_returns_if_both_not_numbers(self):
        self.assertRaises(ValueError, self.calc.add, 'two', 'three')

    def test_calculator_returns_if_x_not_number(self):
        self.assertRaises(ValueError, self.calc.add, 'two', 3)

    def test_calculator_returns_if_y_not_number(self):
        self.assertRaises(ValueError, self.calc.add, 2, 'three')

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

```
In [19]: runfile('C:/Users/helen/OneDrive/Рабочий стол/пары/ПІС,
helen/OneDrive/Рабочий стол/пары/ПІС')
....
----------------------------------------------------------------------
Ran 4 tests in 0.003s

OK
```

2. Виконаємо рефакторинг коду:
   2.1. Множинне присвоювання і розпакування кортежів

Частина коду до рефакторингу:

```python
#функція, що генерує координати міст
def coordinates_for_cities(num_cities):
    X = []
    Y = []
    for i in range(0, int(num_cities), 1):
        X.append(randint(0, 100))
        Y.append(randint(0, 100))
    return [X, Y]
```

Частина коду після рефакторингу:

```
#функція, що генерує координати міст
def coordinates_for_cities(num_cities):
    X, Y = [], []
    for i in range(0, int(num_cities), 1):
        X.append(randint(0, 100))
        Y.append(randint(0, 100))
    return [X, Y]
```

2.2. Включення списків, словників і множин

Частина коду до рефакторингу:

```
#функія, що генерую початкову популяцію
def generate_individuals(population_size, num_cities):
    cities = []
    for i in range(1, int(num_cities)):
        cities.append(i)
    all_ways = []
    for i in range(population_size):
        temp = sample(cities, int(num_cities)-1)
        temp.insert(0, 0)
        all_ways.append(temp)
    return all_ways
```

Частина коду після рефакторингу:

```
#функія, що генерую початкову популяцію
def generate_individuals(population_size, num_cities):
    cities = [x for x in range(1, int(num_cities))]
    all_ways = []
    for i in range(population_size):
        temp = sample(cities, int(num_cities)-1)
        all_ways.append(temp.insert(0, 0))
    return all_ways
```

2.3. Винесемо схожі елементи в окрему функцію:
Блок коду до ре факторингу:

```
while True:
  length = float(input("Enter the length: "))
  if length > 0:
    break
while True:
  width = float(input("Enter the width: "))
  if length > 0:
    break
print("The area is", length * width)
```

Блок коду після рефакторингу:

```python
def input_positive_integer(prompt):
  while True:
    input_value = float(input(prompt))
    if input_value > 0:
      return input_value

length = input_positive_integer("Enter the length: ")
width = input_positive_integer("Enter the width: ")
print("The area is", length * width)
```

Висновок:

Виконуючи лабораторну роботу було розглянуто створення програмного забезпечення за метолологією TDD та процедури рефакторинга. Серед недоліків методології TDD є час розробки програмного забезпечення, але натомість цей метод має значущу перевагу в тому, що програмне забезпечення в результаті є продуманим із багатьох боків. Рефакторинг теж є важливим етапом у розробці ПЗ, бо забезпечує більшу продуктивність, читабельність та масштабованість. Під час виконання лабораторної роботи було розглянуто декілька способів ре факторингу на простих прикладах.