

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 тестування (*Unit-тести*) та рефакторинг»

Виконала:
студентка 4 курсу
групи ДА-72
Грищенко О.Ю.

Київ – 2020

Мета роботи: оволодіти навичками створення програмного забезпечення за методологією 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 функціонування 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 додавання двох чисел. На першому кроці напишемо тест із перевіркою очікуваного результату при додаванні 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/ПИС/untitled0.py', wdir='C:/Users/
helen/OneDrive/Рабочий стол/пары/ПИС')
Reloaded modules: Calculator
F
=====
FAIL: test_add_method_result (__main__.TddTest)
-----
Traceback (most recent call last):
  File "C:/Users/helen/OneDrive/Рабочий стол/пары/ПИС/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)

```

Додамо до створеного класу Calculator до методу додавання результат додавання:

```

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/ПИС/untitled0.py', wdir='C:/Users/
helen/OneDrive/Рабочий стол/пары/ПИС')
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/ПИС/untitled0.py', wdir='C
helen/OneDrive/Рабочий стол/пары/ПИС')
Reloaded modules: Calculator
F
=====
FAIL: test_add_method_result (__main__.TddTest)
-----
Traceback (most recent call last):
  File "C:/Users/helen/OneDrive/Рабочий стол/пары/ПИС/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)

```

Наступним кроком зробимо тест для перевірки додавання двох слів замість чисел:

```
1 import unittest
2 from Calculator import Calculator
3
4 class TddTest(unittest.TestCase):
5     def setUp(self):
6         self.calc = Calculator()
7
8     def test_add_method_returns_correct_result(self):
9         result = self.calc.add(2, 2)
10        self.assertEqual(4, result)
11
12    def test_calculator_returns_if_both_not_numbers(self):
13        self.assertRaises(ValueError, self.calc.add, 'two', 'three')
```

```
In [13]: runfile('C:/Users/helen/OneDrive/Рабочий стол/пары/ПИС/untitled0.py', wdir='C:/Users/helen/OneDrive/Рабочий стол/пары/ПИС')
```

```
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/ПИС/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)
```

Змінімо відповідно клас Калькулятора вказавши, що додаємо лише ті змінні, які є числами:

```
1 class Calculator(object):
2     def __init__(self):
3         pass
4
5     def add(self, x, y):
6         number_types = (int, float, complex)
7
8         if isinstance(x, number_types) and isinstance(y, number_types):
9             return x + y
10        else:
11            raise ValueError
```

Запустимо тест:

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

OK
```

Аналогічно робимо якщо одне число із двох є слово:

```
1 import unittest
2 from Calculator import Calculator
3
4 class TddTest(unittest.TestCase):
5     def setUp(self):
6         self.calc = Calculator()
7
8     def test_add_method_returns_correct_result(self):
9         result = self.calc.add(2, 2)
10        self.assertEqual(4, result)
11
12    def test_calculator_returns_if_both_not_numbers(self):
13        self.assertRaises(ValueError, self.calc.add, 'two', 'three')
14
15    def test_calculator_returns_if_x_not_number(self):
16        self.assertRaises(ValueError, self.calc.add, 'two', 3)
17
18    def test_calculator_returns_if_y_not_number(self):
19        self.assertRaises(ValueError, self.calc.add, 2, 'three')
20
21 if __name__ == '__main__':
22     unittest.main()
23
```

```
In [19]: runfile('C:/Users/helen/OneDrive/Робочий стол/пары/ПІС,
helen/OneDrive/Робочий 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:

```
6 #функція, що генерує координати міст
7 def coordinates_for_cities(num_cities):
8     x = []
9     y = []
10    for i in range(0, int(num_cities), 1):
11        x.append(randint(0, 100))
12        y.append(randint(0, 100))
13    return [x, y]
```

Частина коду після рефакторингу:

```

6      #функція, що генерує координати міст
7      def coordinates_for_cities(num_cities):
8          X, Y = [], []
9          for i in range(0, int(num_cities), 1):
10             X.append(randint(0, 100))
11             Y.append(randint(0, 100))
12         return [X, Y]

```

2.2. Включення списків, словників і множин

Частина коду до рефакторингу:

```

14      #функція, що генерую початкову популяцію
15      def generate_individuals(population_size, num_cities):
16          cities = []
17          for i in range(1, int(num_cities)):
18              cities.append(i)
19          all_ways = []
20          for i in range(population_size):
21              temp = sample(cities, int(num_cities)-1)
22              temp.insert(0, 0)
23              all_ways.append(temp)
24          return all_ways

```

Частина коду після рефакторингу:

```

14      #функція, що генерую початкову популяцію
15      def generate_individuals(population_size, num_cities):
16          cities = [x for x in range(1, int(num_cities))]
17          all_ways = []
18          for i in range(population_size):
19              temp = sample(cities, int(num_cities)-1)
20              all_ways.append(temp.insert(0, 0))
21          return all_ways

```

2.3. Винесемо схожі елементи в окрему функцію:

Блок коду до рефакторингу:

```

1      while True:
2          length = float(input("Enter the length: "))
3          if length > 0:
4              break
5      while True:
6          width = float(input("Enter the width: "))
7          if length > 0:
8              break
9      print("The area is", length * width)

```

Блок коду після рефакторингу:

```
1 def input_positive_integer(prompt):
2     while True:
3         input_value = float(input(prompt))
4         if input_value > 0:
5             return input_value
6
7 length = input_positive_integer("Enter the length: ")
8 width = input_positive_integer("Enter the width: ")
9 print("The area is", length * width)
```

Висновок:

Виконуючи лабораторну роботу було розглянуто створення програмного забезпечення за методологією 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 рефакторингу на простих прикладах.