Практическая работа №3.
Методологии TDD и BDD.

ШАГ №1.
Ознакомится с понятиями и примерами по методологиям.

TDD, или Test-Driven Development (Разработка, управляемая тестами), - это методология разработки программного обеспечения, которая подразумевает создание тестов для функциональности ПО до того, как эта функциональность будет фактически реализована. TDD представляет собой циклический процесс, который помогает разработчикам создавать высококачественное, надежное ПО.

Пример реализации методологии TDD.

1 этап. Создание теста

Первый шаг в TDD - создание теста, описывающего ожидаемое поведение функции. В данном случае, создаётся тест для функции расчета факториала числа и используется библиотека unittest для Python.

```
import unittest

class TestFactorial(unittest.TestCase):
    def test_factorial_of_0(self):
        self.assertEqual(factorial(0), 1)

    def test_factorial_of_5(self):
        self.assertEqual(factorial(5), 120)

    def test_factorial_of_negative_number(self):
        with self.assertRaises(ValueError):
            factorial(-1)

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

Рисунок 1. Тесты

На рисунке 1 представлены следующие тесты:

- ‘*test_factorial_of_0*’: Проверяет, что факториал 0 равен 1.
- ‘*test_factorial_of_5*’: Проверяет, что факториал 5 равен 120.
- ‘*test_factorial_of_negative_number*’: Проверяет, что функция вызывает исключение ‘*ValueError*’, если передано отрицательное число.

2 этап. Запуск тестов

На этом этапе запускаются тесты. Так как функция ‘*factorial*’ еще не существует, все они провалятся.

3 этап. Реализация функции

Далее программируется функция ‘*factorial*’, чтобы сделать реализованные раннее тесты успешными. Пример кода представлен на рисунке 2:

```
def factorial(n):
    if n < 0:
        raise ValueError("Факториал не определен для отрицательных чисел")
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```

Рисунок 2. Код функции

4 этап. Повторный запуск тестов

Теперь запуская тесты снова необходимо убедится, что функция factorial прошла их успешно. Если что-то пошло не так, тесты сообщат об ошибке.

5 этап. Рефакторинг (по желанию)

После успешного прохождения тестов можно провести рефакторинг кода, чтобы улучшить его читаемость и эффективность, при этом убедившись, что тесты остаются успешными.

Таким образом, TDD позволяет разрабатывать функциональность шаг за шагом, начиная с определения ожидаемого поведения в виде тестов, что помогает создавать качественный код с меньшим количеством ошибок.

BDD, или Behavior Driven Development (Разработка, ориентированная на поведение), - это методология разработки программного обеспечения, которая сосредотачивается на описании поведения программы с точки зрения её пользователей и интересующих сторон. BDD представляет собой эволюцию техники TDD (Test-Driven Development), в которой акцент делается на спецификациях поведения и участии бизнес-аналитиков и представителей заказчика в процессе разработки.

Рассмотрим пример BDD на основе разработки функциональности для калькулятора, который должен выполнять базовые арифметические операции. Для описания сценариев BDD используется язык Gherkin, а далее пишутся автоматизированные тесты для этих сценариев.

1 этап. Описание сценариев BDD

На первом этапе создаётся описание сценариев BDD для функциональности калькулятора. Ниже приведён пример сценариев на рисунке 3:

```
Функциональность: Базовые арифметические операции в калькуляторе

  Сценарий: Сложение двух чисел
    Дано калькулятор запущен
    Когда я ввожу "2 + 2"
    Тогда результат должен быть "4"

  Сценарий: Вычитание двух чисел
    Дано калькулятор запущен
    Когда я ввожу "5 - 3"
    Тогда результат должен быть "2"

  Сценарий: Умножение двух чисел
    Дано калькулятор запущен
    Когда я ввожу "3 * 4"
    Тогда результат должен быть "12"

  Сценарий: Деление двух чисел
    Дано калькулятор запущен
    Когда я ввожу "10 / 2"
    Тогда результат должен быть "5"
```

Рисунок 3. Пример сценариев

Каждый сценарий описывает одну из базовых арифметических операций (сложение, вычитание, умножение, деление) и ожидаемый результат.

2 этап. Автоматизация сценариев BDD

На следующем шаге создаются автоматизированные тесты для каждого сценария, используя фреймворк для тестирования, который поддерживает BDD, например, Behave для Python (рисунок 4).

```python
from behave import given, when, then
from calculator import Calculator

@given('калькулятор запущен')
def step_given_calculator_running(context):
    context.calculator = Calculator()

@when('я ввожу "{expression}"')
def step_when_enter_expression(context, expression):
    context.result = context.calculator.calculate(expression)

@then('результат должен быть "{expected_result}"')
def step_then_verify_result(context, expected_result):
    assert context.result == expected_result, f'Ожидался результат
```

Рисунок 4. Пример тестов

В представленном примере «calculator» - это объект класса Calculator, который реализует логику калькулятора, а функции given, when, then соответствуют ключевым словам Gherkin.

3 этап. Реализация калькулятора

На реализуем логику калькулятора в классе Calculator. Это может выглядеть, например, так как представлено на рисунке 5:

```
class Calculator:
    def calculate(self, expression):
        try:
            result = eval(expression)
            return str(result)
        except Exception as e:
            return str(e)
```

Рисунок 5. Логика калькулятора

4 этап. Запуск тестов

Результаты тестов будут показывать, проходят ли они успешно или нет.

Таким образом, BDD позволяет создавать автоматизированные тесты на основе описания ожидаемого поведения программы, что делает спецификации более читаемыми для всех участников проекта и помогает удостовериться, что программа соответствует требованиям и ожиданиям пользователей.

ШАГ №2.

Определить персональное задание по студенческому шифру.

(номер с заданием определяется по последним двум цифрам студенческого шифра. Если цифра превышает 25, то необходимо вычесть 25 до тех пор, пока число не попадёт в диапазон от 01 до 25. Например, последние две цифры номера Вашего студенческого билета 37: 37-25=12. Ваше задание будет под номером 12)

01. Игра «Поле чудес» (угадать загаданное слово).

02. Игра «Города» (база городов, игроки по очереди вводят город за отведенное время, проверяется на правильность).

03. Игра «Крестики-нолики».

04. Научный калькулятор (решение уравнений, матричные операции, интегрирование и т. д.).

05. Матричный калькулятор (операции с матрицами, в том числе с разреженными).

06. Программа-будильник (заданная мелодия в заданное время).

07. Конвертер различных величин (американская система, старорусская, СИ).

08. Бесследный уничтожитель файлов и папок (забивать файл несколько раз заданными символами, потом уничтожать).

09. Электронный терапевт (есть некая база, пользователю задаются вопросы и ставится диагноз).

10. Генератор паттернов (генерация исходного кода по заданному паттерну проектирования с предварительным просмотром).

10. Игра «Как стать миллионером».

11. Игра «Угадай мелодию» (3 игрока, по очереди проигрывается мелодия и спрашивается, что это за мелодия).

12. Конвертер валют с возможностью предсказания курсов (несколько валют и история курса, нужно с помощью экстраполяции узнать курс на следующие дни).

13. Приложение – хранитель паролей (сайты, кредитные карты), доступ в виде игры.

14. Шифратор и дешифратор файлов в заданной папке (шифруем все файлы и расшифровываем несколькими методами).

15. Игра «Морской бой».

16. Игра в тестирование программ (2 игрока). Дается код с ошибками и необходимо найти все ошибки.

17. Программа для конвертации графических файлов (размеры файлов и их форматы разные).

18. Карточная игра в дурака (2 игрока).

19. Приложение для создания миксов мелодий из нескольких файлов (выбираем файлы, начало и конец для каждого из них и получаем итоговый микс).

20. Игра «Угадай паттерн» (игрокам предоставляется код или UML-схема и надо угадать паттерн).

21. Мини-социальная сеть (логин, добавление в друзья с заданным аккаунтом, посылка сообщения другу).

22. Генератор мелодий по заданным аккордам.

23. Программа для напоминания о днях рождения друзей.

24. Игра «Упрощенный футбольный симулятор».

25. Программа – консольный друг (диалог с компьютером на разные темы).

ШАГ №3.

Разработать программный продукт под Вашим номером двумя методами TDD и BDD. В отчете необходимо подробно описать все этапы разработки, представленные выше в примерах.

Во время защиты студент должен продемонстрировать работоспособность своего приложения.