МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

Отчет по практической работе №3

по дисциплине
«**Тестирование и верификация программного обеспечения**»

Тема: «Методологии TDD и BDD»»

Выполнил: Шило Юрий Сергеевич
Студент группы ИКБО-33-22

Проверил: ассистент Петрова А.А.

РЕШЕНИЕ

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.

Реализации методологии TDD.

1 этап. Создание теста

Первый шаг в TDD - создание теста, описывающего ожидаемое поведение функции. В данном случае, создаётся тест игры кто хочет стать миллионером и используется библиотека NUnit для C#.

```
[TestFixture]
public class TDD_Tests
{
    private Parser parser;

    [Test]
    public void TestParserLoads()
    {
        Parser parser = new Parser();
        parser.getQuestionsFromJson(path); // тестовый JSON файл
        Assert.That( actual: 10,  expression: Is.EqualTo(parser.questions.Count),  message: "Должно быть загружено 10 вопроса");
    }

    [Test]
    public void TestGiveAnswers()
    {
        Parser parser = new Parser();
        parser.getQuestionsFromJson(path); // тестовый JSON файл
        Assert.That( actual: 1,  expression: Is.EqualTo(parser.questions[0].correctAnswerIndex),  message: "Правильный ответ под номером 2");
    }
}
```

Рисунок 1. Тесты с использованием методологии TDD

На рисунке 1 представлены следующие тесты:

- ‘test_factorial_of_0’: Проверяет, что было загружено 10 вопросов.
- ‘test_factorial_of_5’: Проверяет, правильно выбирается ответ.

2 этап. Запуск тестов

На этом этапе запускаются тесты. Так как программы не существует, все они провалятся.

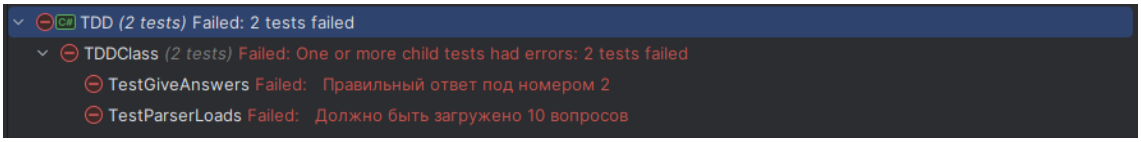


Рисунок 2. Ошибка, выдаваемая при выполнении тестов

3 этап. Реализация программы

Далее мы пишем код нашей программы, чтобы сделать реализованные раннее тесты успешными. Пример кода представлен на рисунке 3:

```
4 usages
public class Question
{
    1 usage
    public string question { get; set; }
    3 usages
    public string[] answers { get; set; }
    4 usages
    public int correctAnswerIndex { get; set; }
}

6 usages
public class Parser{...}

class RootClass
{
    static void Main(string[] args)
    {
        Parser parser = new Parser();
        parser.getQuestionsFromJson(filePath: "Game.json");
        foreach (Question question in parser.questions)
        {
            Console.WriteLine(question.question);
            for (int i = 0; i < question.answers.Length; i++)
                Console.WriteLine($"{i + 1}. {question.answers[i]}");

            Console.Write("Ваш ответ (введите номер варианта): ");
            int userAnswer = int.Parse(Console.ReadLine()) - 1;
            if (userAnswer == question.correctAnswerIndex)
                Console.WriteLine("Верно!\n");
            else
            {
                Console.WriteLine("Неправильно. Правильный ответ: " +
                                  $"{question.answers[question.correctAnswerIndex]}\n");
                break;  // Останавливаем игру при неправильном ответе
            }
        }
        Console.WriteLine("Игра окончена.");
    }
}
```

Рисунок 3. Код программы

4 этап. Повторный запуск тестов

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

Рисунок 4. Успешное прохождение всех тестов

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 тех ники 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.

Реализации методологии BDD.

1 этап. Описание сценариев BDD

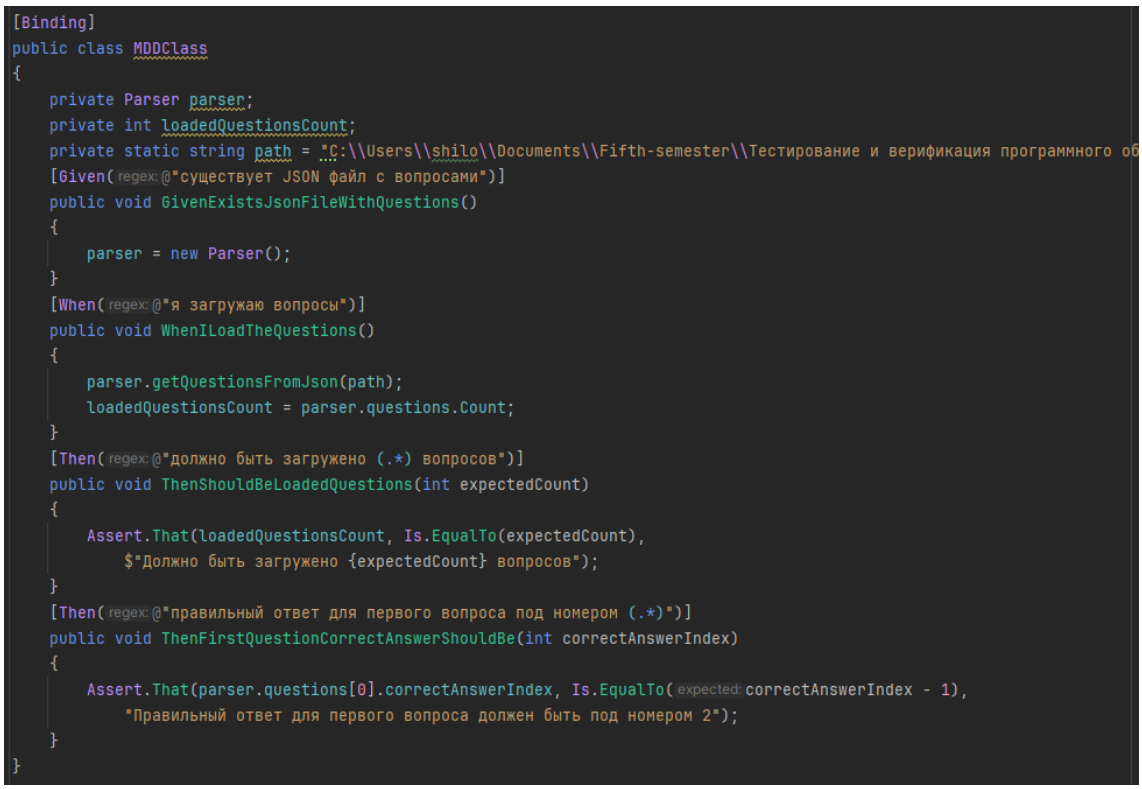
На первом этапе создаётся описание сценариев BDD для функциональности нашего консольного приложения. Ниже приведён пример сценариев на рисунке 5:

```
Feature: Загрузка вопросов из JSON
  Для того, чтобы начать игру "Кто хочет стать миллионером"
  Как пользователь
  Мне нужно загрузить список вопросов из JSON-файла

  Scenario: Успешная загрузка вопросов из JSON
    Given существует JSON файл с вопросами
    When я загружаю вопросы
    Then должно быть загружено 10 вопросов

  Scenario: Первый вопрос содержит правильный ответ
    Given существует JSON файл с вопросами
    When я загружаю вопросы
    Then правильный ответ для первого вопроса под номером 2
```

Рисунок 5. Написанные сценарии

2 этап. Автоматизация сценариев BDD

На следующем шаге создаются автоматизированные тесты для каждого сценария, используя фреймворк для тестирования, который поддерживает BDD, в нашем случае SpecFlow для C# (рисунок 6).



```
[Binding]
public class MDDClass
{
    private Parser parser;
    private int loadedQuestionsCount;
    private static string path = "C:\\Users\\shilo\\Documents\\Fifth-semester\\Тестирование и верификация программного об
    [Given(regex:@"существует JSON файл с вопросами")]
    public void GivenExistsJsonFileWithQuestions()
    {
        parser = new Parser();
    }
    [When(regex:@"я загружаю вопросы")]
    public void WhenILoadTheQuestions()
    {
        parser.getQuestionsFromJson(path);
        loadedQuestionsCount = parser.questions.Count;
    }
    [Then(regex:@"должно быть загружено (.*) вопросов")]
    public void ThenShouldBeLoadedQuestions(int expectedCount)
    {
        Assert.That(loadedQuestionsCount, Is.EqualTo(expectedCount),
            $"Должно быть загружено {expectedCount} вопросов");
    }
    [Then(regex:@"правильный ответ для первого вопроса под номером (.*)")]
    public void ThenFirstQuestionCorrectAnswerShouldBe(int correctAnswerIndex)
    {
        Assert.That(parser.questions[0].correctAnswerIndex, Is.EqualTo(expected:correctAnswerIndex - 1),
            "Правильный ответ для первого вопроса должен быть под номером 2");
    }
}
```

Рисунок 6. Реализуемые тесты

3 этап. Реализация программы

Далее мы пишем код нашей программы, чтобы сделать реализованные раннее тесты успешными. Пример кода представлен на рисунке 7:

```
4 usages
public class Question
{
    1 usage
    public string question { get; set; }
    3 usages
    public string[] answers { get; set; }
    4 usages
    public int correctAnswerIndex { get; set; }
}

6 usages
public class Parser{ ... }

class RootClass
{
    static void Main(string[] args)
    {
        Parser parser = new Parser();
        parser.getQuestionsFromJson( filePath: "Game.json");
        foreach (Question question in parser.questions)
        {
            Console.WriteLine(question.question);
            for (int i = 0; i < question.answers.Length; i++)
                Console.WriteLine($"{i + 1}. {question.answers[i]}");

            Console.Write("Ваш ответ (введите номер варианта): ");
            int userAnswer = int.Parse(Console.ReadLine()) - 1;
            if (userAnswer == question.correctAnswerIndex)
                Console.WriteLine("Верно!\n");
            else
            {
                Console.WriteLine("Неправильно. Правильный ответ: " +
                                  $"{question.answers[question.correctAnswerIndex]}\n");
                break;  // Останавливаем игру при неправильном ответе
            }
        }
        Console.WriteLine("Игра окончена.");
    }
}
```

Рисунок 7. Код программы

4 этап. Запуск тестов

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

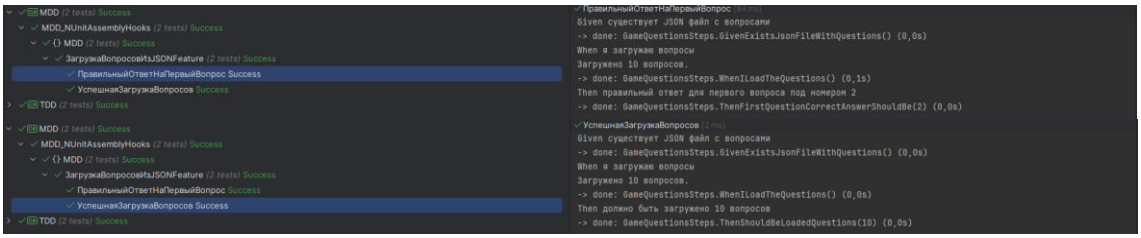


Рисунок 8. Успешное прохождение всех тестов

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.

ВЫВОД

В результате выполнения данной практической работы мы научились создавать Unit тесты используя две методологии TDD и BDD.