МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий

Кафедра «Информационные системы»

**ОТЧЕТ**

о выполнении лабораторной работы №5

по дисциплине “Тестирование Программного Обеспечения”

Выполнил: ст. гр. ИС/б-20-2-о

Белик Г. М.

Проверил: доц. каф.

«Информационные системы»

Строганов В. А.

Севастополь

2022

**ЛАБОРАТОРНАЯ РАБОТА №5**

**«Исследование способов модульного тестирования программного обеспечения в среде NUnit»**

**Цель работы**

Исследовать эффективность использования методологии TDD при разработке программного обеспечения. Получить практические навыки использования фреймворка NUnit для модульного тестирования программного обеспечения.

**Задачи**

1. Реализовать на языке C# один из классов, спроектированных в лабораторной работе № 1. Методы класса при этом не реализовывать.
2. Разработать для созданного класса набор модульных тестов, включающий тесты для каждого метода.
3. Запустить набор тестов, проанализировать и сохранить результаты.
4. Поочередно реализовать методы класса, выполняя тестирование при каждом изменении программного кода.
5. После того, как весь набор тестов будет выполняться успешно, реализацию классов можно считать завершенной.

**Ход работы**

Для модульного тестирования был выбрал класс Matrix. Изначально создали модульные тесты для простых операций, затем для методов инициализации массива Massiv\_Initilization, матрицы Matriza\_Initialization и вычислений Calculation также были разработаны тесты, и реализованы методы класса. Далее все тесты проходили проверку только в случае правильного выполнения.

Ниже приведен пример процесса выполнения всех тестов (рисунок 1).

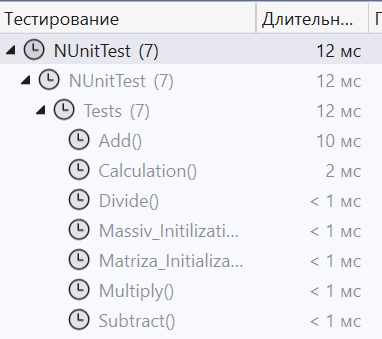


Рисунок 1 – Запуск Тестирования в NUnit

Затем наглядно представлен пример верного выполнения всех разработанных

тестов (рисунок 2).

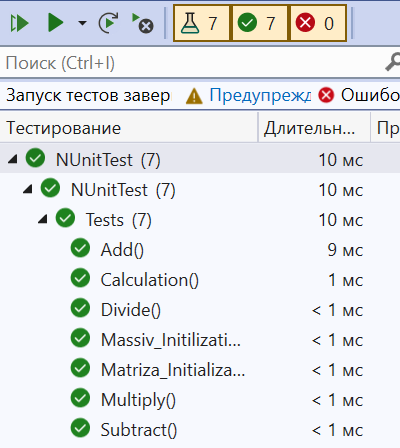


Рисунок 2 – Корректное прохождение всех тестов

**Текст программы класса UnitTest:**

namespace NUnitTest

{

[TestFixture]

public class Tests

{

[TestCase]

public void Massiv\_Initilization()

{

Matrix math = new Matrix();

int[] mass = new int[] { 1, 1, 1};

Assert.AreEqual(mass ,math.Massiv());

}

[TestCase]

public void Matriza\_Initialization()

{

Matrix math = new Matrix();

int[,] matr = new int[3, 3]{ { -1, 1, 2 }, { 0, 1, 2 },{ 3, 4, 5 } };

Assert.AreEqual(matr, math.Matriza());

}

[TestCase]

public void Calculation()

{

Matrix math = new Matrix();

int[] mass\_result = new int[] { -1, 0, 60 };

Assert.AreEqual(mass\_result, math.Calculation());

}

[TestCase]

public void Add()

{

Matrix math = new Matrix();

Assert.AreEqual(20, math.Add(10, 10));

}

[TestCase]

public void Subtract()

{

Matrix math = new Matrix();

Assert.AreEqual(0, math.Subtract(10, 10));

}

[TestCase]

public void Multiply()

{

Matrix math = new Matrix();

Assert.AreEqual(30, math.Multiply(5, 6));

}

[TestCase]

public void Divide()

{

Matrix math = new Matrix();

Assert.AreEqual(0, math.Divide(10, 0));

}

}

}

**Текст программы Matrix:**

namespace NUnitTest

{

internal class Matrix

{

public int[] Massiv()

{

int[] mass = new int[] { 1, 1, 1};

return mass;

}

public int[,] Matriza()

{

int[,] matr = new int[3, 3] { { -1, 1, 2 }, { 0, 1, 2 }, { 3, 4, 5 } };

return matr;

}

public int[] Calculation()

{

int[] mass\_result = new int[] { 1, 1, 1 };

int[,] matr = new int[3, 3] { { -1, 1, 2 }, { 0, 1, 2 }, { 3, 4, 5 } };

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

{

mass\_result[i] \*= matr[i, j];

if (matr[i, j] < 0)

{

mass\_result[i] = -1;

break;

}

}

return mass\_result;

}

public int Add(int a, int b)

{

return a + b;

}

public int Subtract(int a, int b)

{

return a - b;

}

public int Multiply(int a, int b)

{

return a \* b;

}

public int Divide(int a, int b)

{

if (b == 0)

return 0;

else

return a / b;

}

}

}

**Выводы**

В ходе лабораторной работы была исследована эффективность использования методологии TDD при разработке программного обеспечения. В результате выполнения сделаны выводы о пользе данного метода в случаях применения к сложным программам или критическим частям, так как написание тестовых модулей для каждой функции занимает неоправданно много времени. Также были приобретены практические навыки использования фреймворка NUnit для модульного тестирования программного обеспечения.