Практическая работа № 1.  
Тестирование

**Цель практической работы**

Закрепление теоретических знаний по разработке модульных тестов к программам.

**Постановка задачи**

**Тести́рование** програ́ммного обеспе́че́ния — процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом.

**Верификация (verification) –** Соответствие продукта требованиям (спецификации). Т.е. верификация — это проверка того, что программа работает так, как было задумано.

Валидация (validation) – Соответствие продукта потребностям пользователей. Т.е. работа программы устраивает заказчика.

**Модульное** (компонентное) тестирование (Unit Testing) проводится самими разработчиками, т.к. предполагает полный доступ к коду, для тестирования какого-либо одного логически выделенного и изолированного элемента (модуля) системы в коде, проверяет функциональность и ищет дефекты в частях приложения, которые доступны и могут быть протестированы по-отдельности (модули программ, объекты, классы, функции и т.д.).

**Виды ошибок:**

**Error** – это ошибка **пользователя**, то есть он пытается использовать программу иным способом (например, вводит буквы в поля, где требуется вводить цифры). В качественной программе предусмотрены такие ситуации и выдаются сообщение об ошибке.

**Bug** (defect) – это ошибка **программиста** (или дизайнера или ещё кого, кто принимает участие в разработке), то есть когда в программе, что-то идёт не так, как планировалось. Например, внутри программа построена так, что изначально не соответствует тому, что от неё ожидается.

**Failure** – это сбой в работе компонента, всей программы или системы (может быть как аппаратным, так и вызванным дефектом).

**Пример** тестирования модуля проекта:

Создадим WindowsForms проект, добавим класс поиска целочисленных корней уравнения, который будем тестировать:

public class IntegerEquationSolver

{

public List<int> GetRootsOnInterval(int from, int to,

Func<double, double> function)

{

var roots = new List<int>();

for (int x = from; x <= to; x++)

{

if (Math.Abs(function(x)) <= 1e-12)

roots.Add(x);

}

return roots;

}

}

Добавим к решению тесты. Необходимо нажать ПКМ на Решение –> Добавить –> Создать проект –> Выбрать C#, Все платформы, Тестирование. Из отобразившегося списка проектов выбрать «Тестовый проект NUnit» (рисунок 1, необходимо предварительно скачать NUnit через менеджер Nuget пакетов).

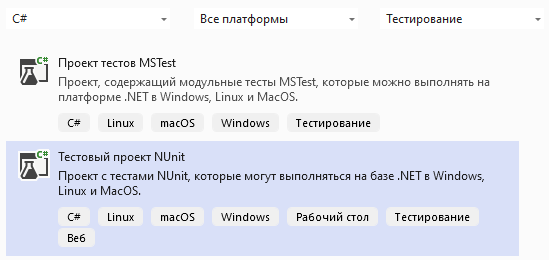


Рисунок 1 – Создание проекта с тестами

Создадим экземпляр класса для поиска корней уравнения и напишем тест «OneZeroRoot» в котором будем проверять случай, когда корень единственный и равен 0:

IntegerEquationSolver solver;

[SetUp]

public void Setup()

{

solver = new IntegerEquationSolver();

}

[Test]

public void OneZeroRoot()

{

var roots = solver.GetRootsOnInterval(-10, 10, (x) => x \* x);

Assert.AreEqual(1, roots.Count);

Assert.AreEqual(0, roots.First());

}

**Рассмотрим** код:

Assert.AreEqual(expected, actual);

С помощью метода AreEqual мы сравниваем ожидаемый (верный) результат с полученным (actual). Тест считается пройденным, если значения совпадают.

Методы сравнения ожидаемых и полученных значений:

**AllItemsAreInstancesOfType** – проверка типа значений

**AllItemsAreNotNull** – проверка значений на Null

**AllItemsAreUnique** – проверка на уникальность значений в коллекции

**AreNotEqual** – проверка на то, что коллекция НЕ содержит заданные элементы;

**AreNotEquivalent** - проверка на то, что коллекция НЕ содержит заданные элементы или их порядок другой.

**Contains** – проверка на то, что коллекция содержит заданный элемент;

**DoesNotContain** – проверка на то, что коллекция НЕ содержит заданный элемент;

**IsOrdered** - проверка на то, что элементы коллекции упорядочены;

**IsEmpty** – пустая ли коллекция;

**IsNotEmpty** – НЕ пустая ли коллекция;

**IsSubsetOf** – является ли коллекция подмножеством другой коллекции;

**IsNotSubsetOf** – НЕ является ли коллекция подмножеством другой коллекции;

**IsSupersetOf** – является ли другая коллекция подмножеством данной;

**IsNotSupersetOf** – НЕ является ли другая коллекция подмножеством данной.

Пример тестов метода с аргументами:

[TestCase(-2, 2, 0)]

[TestCase(-4, 0, 1)]

[TestCase(0, 4, 1)]

[TestCase(-4, 4, 2)]

public void RootsCount(int from, int to, int expectedRootsCount)

{

var roots = solver.GetRootsOnInterval(from,to,(x) => x \* x - 9);

Assert.AreEqual(expectedRootsCount, roots.Count);

}

**Рекомендации от Майкрософт по написанию тестов:** <https://learn.microsoft.com/ru-ru/dotnet/core/testing/unit-testing-best-practices>

**Задание на практическую работу**

1. Разработать класс GeoEngine, реализующий математические операции для работы с геометрическими объектами. Движок должен обеспечивать:

* вычисление векторного и скалярного произведения;
* проверка принадлежности точки выпуклому многограннику;
* вычисления расстояния между отрезками в пространстве;
* вычисления осевого (геометрического) момента инерции тела, заданного множеством точек.

1. Разработать модульные тесты для класса GeoEngine, не менее 3 тестов на каждый публичный метод.
2. **Необходимо учесть**: граничные случаи, разные по смыслу ситуации, пустые значения.
3. Защита работы включает демонстрацию работы программы, описание тестируемых ситуаций и прохождение тестов программой.

**Пример разработки приложения**

Разработать класс GeoEngine, реализующий математические операции для работы с геометрическими объектами. Движок должен обеспечивать:

* вычисление векторного и скалярного произведения;

Структура решения представлена на рисунке

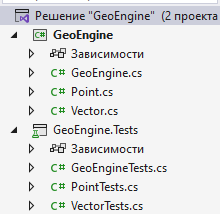


Рисунок 2 – Структура решения

**Шаг 1: Создание решения и проектов**

1. Откройте Visual Studio
2. Создайте новое решение (Solution):
   * Файл → Создать → Проект → Пустое решение (Blank Solution)
   * Назовите "GeometrySolution"
3. Добавьте проект библиотеки классов:
   * ПКМ по решению → Добавить → Новый проект → "Class Library" (.NET 8.0)
   * Назовите "GeoEngine"
4. Добавьте проект для тестов:
   * ПКМ по решению → Добавить → Новый проект → "NUnit Test Project" (.NET 8.0)
   * Назовите "GeoEngine.Tests"

**Шаг 2: Настройка зависимостей**

1. В проекте тестов добавьте ссылку на основную библиотеку:
   * ПКМ по "GeoEngine.Tests" → Добавить → Ссылка на проект → Выберите "GeoEngine"

**Шаг 3: Реализация основных классов**

1. Создадим класс Point для представления точки в 3D пространстве

**Point.cs:**

namespace GeoEngine;

public class Point

{

public double X { get; }

public double Y { get; }

public double Z { get; }

public Point(double x, double y, double z)

{

X = x;

Y = y;

Z = z;

}

public override bool Equals(object? obj)

{

if (obj is not Point other) return false;

return X.Equals(other.X) && Y.Equals(other.Y) && Z.Equals(other.Z);

}

public override int GetHashCode()

{

return HashCode.Combine(X, Y, Z);

}

public override string ToString()

{

return $"({X}, {Y}, {Z})";

}

}

2. Создадим класс Vector для представления векторов

**Vector.cs:**

namespace GeoEngine;

public class Vector

{

public double X { get; }

public double Y { get; }

public double Z { get; }

public Vector(double x, double y, double z)

{

X = x;

Y = y;

Z = z;

}

public Vector(Point start, Point end)

{

X = end.X - start.X;

Y = end.Y - start.Y;

Z = end.Z - start.Z;

}

public double Length => Math.Sqrt(X \* X + Y \* Y + Z \* Z);

public override bool Equals(object? obj)

{

if (obj is not Vector other) return false;

return X.Equals(other.X) && Y.Equals(other.Y) && Z.Equals(other.Z);

}

public override int GetHashCode()

{

return HashCode.Combine(X, Y, Z);

}

public override string ToString()

{

return $"[{X}, {Y}, {Z}]";

}

}

3. Реализуем основной класс GeoEngine

**GeoEngine.cs:**

namespace GeoEngine;

public static class GeoEngine

{

/// <summary>

/// Вычисляет скалярное произведение двух векторов

/// </summary>

/// <param name="v1">Первый вектор</param>

/// <param name="v2">Второй вектор</param>

/// <returns>Скалярное произведение</returns>

public static double DotProduct(Vector v1, Vector v2)

{

return v1.X \* v2.X + v1.Y \* v2.Y + v1.Z \* v2.Z;

}

/// <summary>

/// Вычисляет векторное произведение двух векторов

/// </summary>

/// <param name="v1">Первый вектор</param>

/// <param name="v2">Второй вектор</param>

/// <returns>Новый вектор - результат векторного произведения</returns>

public static Vector CrossProduct(Vector v1, Vector v2)

{

double x = v1.Y \* v2.Z - v1.Z \* v2.Y;

double y = v1.Z \* v2.X - v1.X \* v2.Z;

double z = v1.X \* v2.Y - v1.Y \* v2.X;

return new Vector(x, y, z);

}

/// <summary>

/// Вычисляет угол между двумя векторами в радианах

/// </summary>

public static double AngleBetween(Vector v1, Vector v2)

{

double dot = DotProduct(v1, v2);

double lengths = v1.Length \* v2.Length;

if (lengths < 1e-10)

throw new ArgumentException("Один из векторов имеет нулевую длину");

// Убедимся, что значение в пределах [-1, 1] из-за ошибок округления

double cos = dot / lengths;

cos = Math.Clamp(cos, -1, 1);

return Math.Acos(cos);

}

/// <summary>

/// Проверяет, являются ли векторы коллинеарными

/// </summary>

public static bool AreCollinear(Vector v1, Vector v2, double tolerance = 1e-10)

{

var cross = CrossProduct(v1, v2);

return Math.Abs(cross.X) < tolerance &&

Math.Abs(cross.Y) < tolerance &&

Math.Abs(cross.Z) < tolerance;

}

}

**Шаг 4: Написание Unit-тестов**

1. Тесты для класса Point

**PointTests.cs:**

namespace GeoEngine.Tests;

[TestFixture]

public class PointTests

{

[Test]

public void Constructor\_SetsCoordinates()

{

var point = new Point(1.5, 2.5, 3.5);

Assert.That(point.X, Is.EqualTo(1.5));

Assert.That(point.Y, Is.EqualTo(2.5));

Assert.That(point.Z, Is.EqualTo(3.5));

}

[Test]

public void Equals\_ReturnsTrueForSamePoints()

{

var p1 = new Point(1, 2, 3);

var p2 = new Point(1, 2, 3);

Assert.That(p1.Equals(p2), Is.True);

}

}

2. Тесты для класса Vector

**VectorTests.cs:**

namespace GeoEngine.Tests;

[TestFixture]

public class VectorTests

{

[Test]

public void Constructor\_FromPoints\_CreatesCorrectVector()

{

var start = new Point(1, 2, 3);

var end = new Point(4, 6, 9);

var vector = new Vector(start, end);

Assert.That(vector.X, Is.EqualTo(3));

Assert.That(vector.Y, Is.EqualTo(4));

Assert.That(vector.Z, Is.EqualTo(6));

}

[Test]

public void Length\_CalculatesCorrectly()

{

var vector = new Vector(3, 4, 0);

Assert.That(vector.Length, Is.EqualTo(5));

}

[Test]

public void AngleBetween\_ZeroLengthVector\_ThrowsException()

{

var v1 = new Vector(1, 0, 0);

var v2 = new Vector(0, 0, 0);

Assert.Throws<ArgumentException>(() => GeoEngine.AngleBetween(v1, v2));

}

}

3. Тесты для GeoEngine

**GeoEngineTests.cs:**

namespace GeoEngine.Tests;

[TestFixture]

public class GeoEngineTests

{

[Test]

public void DotProduct\_OrthogonalVectors\_ReturnsZero()

{

var v1 = new Vector(1, 0, 0);

var v2 = new Vector(0, 1, 0);

var result = GeoEngine.DotProduct(v1, v2);

Assert.That(result, Is.EqualTo(0));

}

[Test]

public void CrossProduct\_StandardBasisVectors\_ReturnsThirdBasisVector()

{

var v1 = new Vector(1, 0, 0);

var v2 = new Vector(0, 1, 0);

var result = GeoEngine.CrossProduct(v1, v2);

Assert.That(result, Is.EqualTo(new Vector(0, 0, 1)));

}

[Test]

public void AngleBetween\_ParallelVectors\_ReturnsZero()

{

var v1 = new Vector(1, 0, 0);

var v2 = new Vector(2, 0, 0);

var angle = GeoEngine.AngleBetween(v1, v2);

Assert.That(angle, Is.EqualTo(0).Within(1e-10));

}

[Test]

public void AreCollinear\_ParallelVectors\_ReturnsTrue()

{

var v1 = new Vector(1, 2, 3);

var v2 = new Vector(2, 4, 6);

Assert.That(GeoEngine.AreCollinear(v1, v2), Is.True);

}

}

**Шаг 6: Запуск тестов**

1. Откройте Test Explorer (Тест → Обозреватель тестов)
2. Нажмите "Запустить все"
3. Убедитесь, что все тесты проходят успешно (рисунок 3)

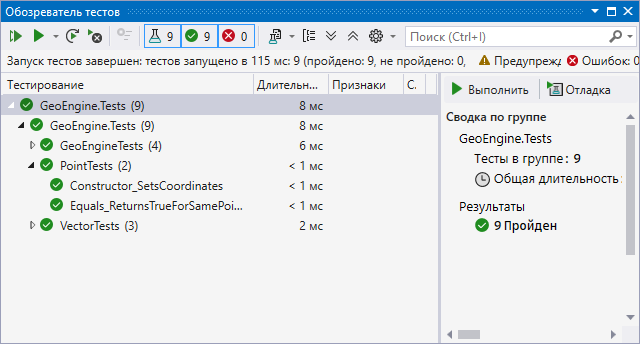


Рисунок 3 – Результат прохождения тестов

# Содержание пояснительной записки

1. Постановка задачи. Приводится теоретический материал, использованный при написании приложения.

2. Формулировка задания и вариант. Приводится задание на лабораторную работу и вариант этого задания.

3. Описание выполняемых действий. Необходимо привести описание последовательности разработки программы, реализации используемых методов, алгоритмов, блок-схем.

4. Анализ результатов. Привести анализ входных и выходных данных. Показать результаты выполнения программного кода. Предоставить скриншоты обработки тестовых примеров. Сделать выводы.

5. Листинг программы. Привести листинг разработанного программного кода, содержание файлов входных и выходных данных.

# Используемое программное обеспечение

1. Среда программирования MS Visual Studio Community 2022 (Свободно распространяемое программное обеспечение (в учебных целях));
2. Microsoft Office Standard 2007 (Open License: 42267924);
3. Open Office (Свободно распространяемое программное обеспечение).
4. Браузер (Свободно распространяемое программное обеспечение).

# Список литературы

* + - 1. Мейер Б. Объектно-ориентированное программирование и программная инженерия [Электронный ресурс]/ Мейер Б. – Электрон. текстовые данные. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. – 285 c.
      2. Биллиг, В. A. Основы объектного программирования на С# (C# 3.0, Visual Studio 2008) [Электронный ресурс] : учебное пособие / В. A. Биллиг. — Электрон. текстовые данные. — Москва, Саратов : Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2017. — 583 c. — 978-5-4487-0145-0. — Режим доступа: http://www.iprbookshop.ru/72339.html
      3. Павловская, Т. А. Программирование на языке высокого уровня C# [Электронный ресурс] / Т. А. Павловская. — 2-е изд. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 245 c. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/73713.html
      4. Агапов, В. П. Основы программирования на языке С# [Электронный ресурс] : учебное пособие / В. П. Агапов. — Электрон. текстовые данные. — М. : Московский государственный строительный университет, ЭБС АСВ, 2012. — 128 c. — 978-5-7264-0576-6. — Режим доступа: http://www.iprbookshop.ru/16366.html
      5. Медведев, М. А. Программирование на СИ# [Электронный ресурс] : учебное пособие / М. А. Медведев, А. Н. Медведев ; под ред. А. В. Присяжный. — Электрон. текстовые данные. — Екатеринбург : Уральский федеральный университет, ЭБС АСВ, 2015. — 64 c. — 978-5-7996-1561-1. — Режим доступа: http://www.iprbookshop.ru/69667.html
      6. Казанский А.А. Объектно-ориентированное программирование на языке Microsoft Visual С# в среде разработки Microsoft Visual Studio 2008 и .NET Framework. 4.3 [Электронный ресурс]: учебное пособие и практикум/ Казанский А.А.— Электрон. текстовые данные.— М.: Московский государственный строительный университет, ЭБС АСВ, 2011.— 180 c
      7. Уйманова Н.А. Основы объектно-ориентированного программирования [Электронный ресурс]: практикум/ Уйманова Н.А., Таспаева М.Г.— Электрон. текстовые данные.— Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2017.— 156 c.— Режим доступа: http://www.iprbookshop.ru/78808.html.— ЭБС «IPRbooks»
      8. Новиков П.В. Объектно-ориентированное программирование [Электронный ресурс]: учебно-методическое пособие к лабораторным работам/ Новиков П.В.— Электрон. текстовые данные.— Саратов: Вузовское образование, 2017.— 124 c.— Режим доступа: http://www.iprbookshop.ru/64650.html.— ЭБС «IPRbooks»