**Міністерство освіти і науки України**

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра обчислювальної техніки**

**Лабораторна робота №3**

з дисципліни

«Компоненти програмної інженерії. Якість та тестування

програмного забезпечення»

на тему

«Unit тестування з використанням методів White Box Testing»

Виконав: Перевірив:

студент групи ІП-93 Бабарикін І. В.

Завальнюк Максим Євгенович

Залікова книжка: 9312

Номер у списку: 10

Київ 2021

## Мета

Написати Unit тести з використанням методів White Box Testing.

## Основні кроки виконання

1. Вибрати бібліотеку для тестування за допомогою остачі від ділення номеру залікової книжки на 6.
2. Створити проект тестування, в який підключити бібліотеку.
3. Написати юніт-тести притримуючись методів White Box Testing.
4. Підготувати звіт про виконану роботу, який буде містити використані методи White Box Testing, сирцеві коди юніт-тестів та/або посилання на GitHub де розміщено проект, результати тестування і їх аналіз.

## Виконання роботи

Для початку я обрахував свій варіант – 9312 mod 6 = 0. Отже, бібліотека, яку необхідно мені тестувати – BinaryFlag.

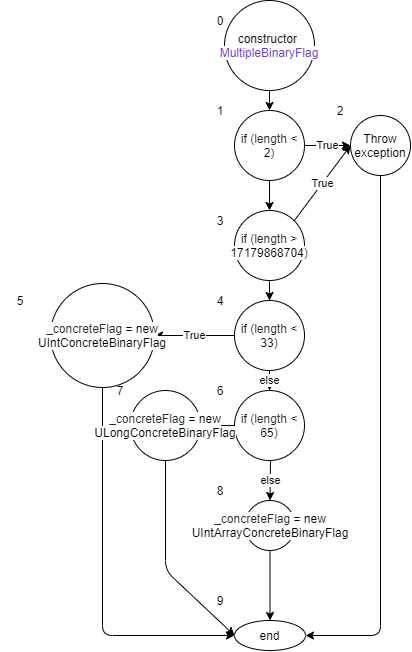
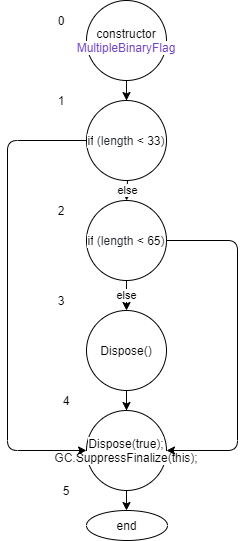
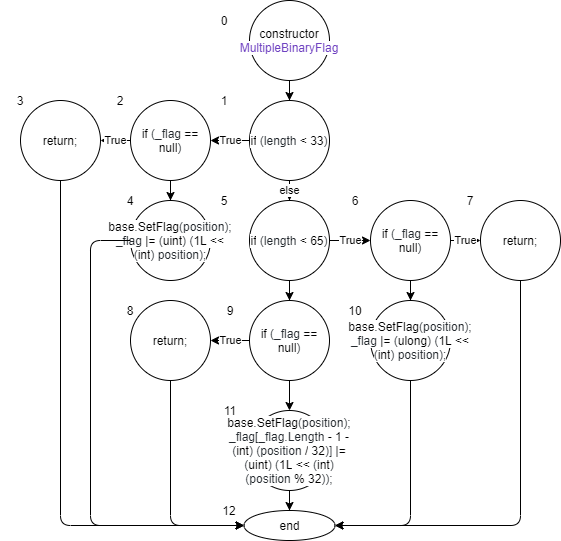
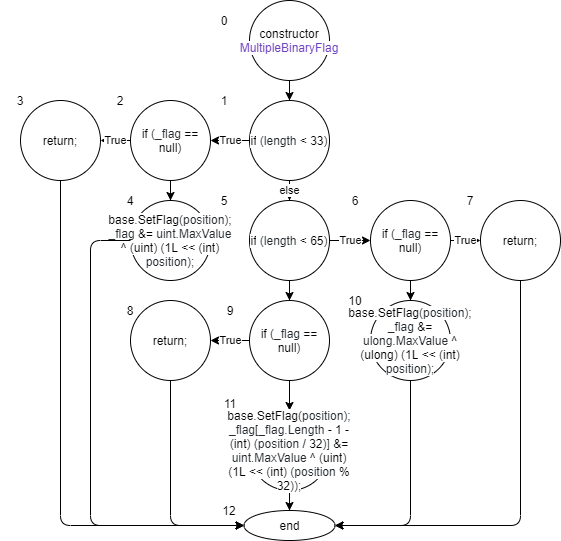
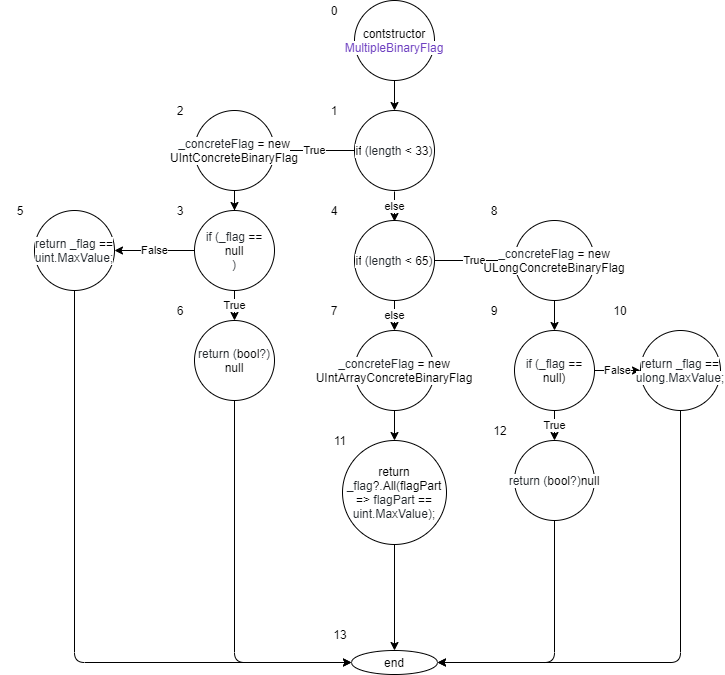
Буду я працювати із платформою **.NET 5** та бібліотекою для тестування **xUnit**.

Далі я створив проект Lab3. До проекту я під’єднав файл **.dll** бібліотеки, що тестую.

Оскільки це WhiteBox тестування, то я у мене була змога подивитись код бібліотеки. У ньому я побачив один публічний клас з декількома методами – MultipleBinaryFlag. Тому тестувати буду його.

Далі необхідно обрати техніку **WBT** і відповідно до неї написати тести. Я обрав **Тестування потоку виконання**. Коли програма тестується даним видом, то відповідні тест-кейси створюються таким чином, щоб перевірити правильність виконання максимально можливої кількість шляхів виконання ПЗ. Отже, моїм завданням стало створення таких шляхів, і найкраще це можна представити графічно.

Я створив діаграми для кожного методу, що тестую, щоб побачити усі можливі «сценарії»:

* public MultipleBinaryFlag()
* public void Dispose()
* public void SetFlag()
* public void ResetFlag()
* public bool? GetFlag()

Усі ці діаграми знаходяться [тут](https://github.com/mezgoodle/TestLabs-kpi/tree/master/docs).

Тепер по заданим маршрутам ми створюємо тест-кейси:

using IIG.BinaryFlag;

using System;

using Xunit;

namespace Lab3

{

    public class MultipleBinaryFlagTests

    {

        /// <summary>

        /// Test for constructor method

        /// </summary>

        public class ConstructorTests

        {

            /// <summary>

            /// Test UIntArrayConcreteBinaryFlag object creation

            /// </summary>

            [Fact]

            public void Route\_0\_1\_3\_4\_6\_8\_9true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 + 1, true);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test UIntArrayConcreteBinaryFlag object creation

            /// </summary>

            [Fact]

            public void Route\_0\_1\_3\_4\_6\_8\_9false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 + 1, false);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test ULongConcreteBinaryFlag object creation

            /// </summary>

            [Fact]

            public void Route\_0\_1\_3\_4\_6\_7\_9true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(64, true);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test ULongConcreteBinaryFlag object creation

            /// </summary>

            [Fact]

            public void Route\_0\_1\_3\_4\_6\_7\_9false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(64, false);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test UIntConcreteBinaryFlag object creation

            /// </summary>

            [Fact]

            public void Route\_0\_1\_3\_4\_5\_9true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(32, true);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test UIntConcreteBinaryFlag object creation

            /// </summary>

            [Fact]

            public void Route\_0\_1\_3\_4\_5\_9false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(32, false);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test ArgumentOutOfRangeException exception

            /// </summary>

            [Fact]

            public void Route\_0\_1\_3\_2\_9true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(17179868704 + 1, true);

                    Assert.False(true);

                }

                catch (ArgumentOutOfRangeException)

                {

                    Assert.True(true);

                }

            }

            /// <summary>

            /// Test ArgumentOutOfRangeException exception

            /// </summary>

            [Fact]

            public void Route\_0\_1\_3\_2\_9false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(17179868704 + 1, false);

                    Assert.False(true);

                }

                catch (ArgumentOutOfRangeException)

                {

                    Assert.True(true);

                }

            }

            /// <summary>

            /// Test ArgumentOutOfRangeException exception

            /// </summary>

            [Fact]

            public void Route\_0\_1\_2\_9true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(2 - 1, true);

                    Assert.False(true);

                }

                catch (ArgumentOutOfRangeException)

                {

                    Assert.True(true);

                }

            }

            /// <summary>

            /// Test ArgumentOutOfRangeException exception

            /// </summary>

            [Fact]

            public void Route\_0\_1\_2\_9false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(2 - 1, false);

                    Assert.False(true);

                }

                catch (ArgumentOutOfRangeException)

                {

                    Assert.True(true);

                }

            }

        }

        public class GetFlagMethodTests

        {

            /// <summary>

            /// Test GetFlag method with UIntArrayConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_4\_7\_11\_13true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 + 1, true);

                    bool? expected = obj.GetFlag();

                    Assert.NotNull(expected);

                    Assert.True(expected);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test GetFlag method with UIntArrayConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_4\_7\_11\_13false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 + 1, false);

                    bool? expected = obj.GetFlag();

                    Assert.NotNull(expected);

                    Assert.False(expected);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test GetFlag method with ULongConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_4\_8\_9\_10\_13true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 - 1, true);

                    bool? expected = obj.GetFlag();

                    Assert.NotNull(expected);

                    Assert.True(expected);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test GetFlag method with ULongConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_4\_8\_9\_10\_13false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 - 1, false);

                    bool? expected = obj.GetFlag();

                    Assert.NotNull(expected);

                    Assert.False(expected);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test GetFlag method with UIntConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_2\_3\_5\_13true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(33 - 1, true);

                    bool? expected = obj.GetFlag();

                    Assert.NotNull(expected);

                    Assert.True(expected);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test GetFlag method with UIntConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_2\_3\_5\_13false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(33 - 1, false);

                    bool? expected = obj.GetFlag();

                    Assert.NotNull(expected);

                    Assert.False(expected);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

        }

        public class SetFlagMethodTests

        {

            /// <summary>

            /// Test SetFlag method with UIntArrayConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_5\_9\_11\_12true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 + 123, true);

                    obj.SetFlag(100);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test SetFlag method with UIntArrayConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_5\_9\_11\_12false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 + 123, false);

                    obj.SetFlag(100);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test SetFlag method with ULongConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_5\_6\_10\_12true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 - 1, true);

                    obj.SetFlag(50);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test SetFlag method with ULongConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_5\_6\_10\_12false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 - 1, false);

                    obj.SetFlag(50);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test SetFlag method with UIntConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_2\_4\_12true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(33 - 1, true);

                    obj.SetFlag(20);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test SetFlag method with UIntConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_2\_4\_12false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(33 - 1, false);

                    obj.SetFlag(20);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

        }

        public class ResetFlagMethodTests

        {

            /// <summary>

            /// Test ResetFlag method with UIntArrayConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_5\_9\_11\_12true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 + 123, true);

                    obj.ResetFlag(100);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test ResetFlag method with UIntArrayConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_5\_9\_11\_12false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 + 123, false);

                    obj.ResetFlag(100);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test ResetFlag method with ULongConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_5\_6\_10\_12true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 - 1, true);

                    obj.ResetFlag(50);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test ResetFlag method with ULongConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_5\_6\_10\_12false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 - 1, false);

                    obj.ResetFlag(50);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test ResetFlag method with UIntConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_2\_4\_12true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(33 - 1, true);

                    obj.ResetFlag(20);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test ResetFlag method with UIntConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_2\_4\_12false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(33 - 1, false);

                    obj.ResetFlag(20);

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

        }

        public class DisposeMethodTests

        {

            /// <summary>

            /// Test Dispose method with UIntConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_4\_5true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(33 - 1, true);

                    obj.Dispose();

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test Dispose method with UIntConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_4\_5false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(33 - 1, false);

                    obj.Dispose();

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test Dispose method with ULongConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_2\_4\_5true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 - 1, true);

                    obj.Dispose();

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test Dispose method with ULongConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_2\_4\_5false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 - 1, false);

                    obj.Dispose();

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test Dispose method with UIntArrayConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_2\_3\_4\_5true()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 + 1, true);

                    obj.Dispose();

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test Dispose method with UIntArrayConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void Route\_0\_1\_2\_3\_4\_5false()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 + 1, false);

                    obj.Dispose();

                    Assert.True(true);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

        }

        public class ToStringMethodTests

        {

            /// <summary>

            /// Test ToString method with UIntConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void ToString\_UIntConcreteBinaryFlag\_NotNulltrue()

            {

                try

                {

                    MultipleBinaryFlag obj = new(33 - 1, true);

                    string expected = obj.ToString();

                    Assert.NotNull(expected);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test ToString method with UIntConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void ToString\_UIntConcreteBinaryFlag\_NotNullfalse()

            {

                try

                {

                    MultipleBinaryFlag obj = new(33 - 1, false);

                    string expected = obj.ToString();

                    Assert.NotNull(expected);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test ToString method with ULongConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void ToString\_ULongConcreteBinaryFlag\_NotNulltrue()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 - 1, true);

                    string expected = obj.ToString();

                    Assert.NotNull(expected);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test ToString method with ULongConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void ToString\_ULongConcreteBinaryFlag\_NotNullfalse()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 - 1, false);

                    string expected = obj.ToString();

                    Assert.NotNull(expected);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test ToString method with UIntArrayConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void ToString\_UIntArrayConcreteBinaryFlag\_NotNulltrue()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 + 1, true);

                    string expected = obj.ToString();

                    Assert.NotNull(expected);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

            /// <summary>

            /// Test ToString method with UIntArrayConcreteBinaryFlag object

            /// </summary>

            [Fact]

            public void ToString\_UIntArrayConcreteBinaryFlag\_NotNullfalse()

            {

                try

                {

                    MultipleBinaryFlag obj = new(65 + 1, false);

                    string expected = obj.ToString();

                    Assert.NotNull(expected);

                }

                catch (Exception)

                {

                    Assert.False(true);

                }

            }

        }

    }

}

У назвах методів вказано, який сценарій я тестую, і перевірити його можна по діаграмі. Тільки при тестуванні методу **ToString** немає сценаріїв, оскільки вони там недоречні.

## Висновок

Під час виконання роботи я познайомився із технологією **WhiteBox** тестування. Вона істотно відрізняється від **BBT.** Головним завданням є правильне складання усіх тест-кейсів, тому це не легше, ніж саме написання тестів.

## Джерела

1. [Папка](https://github.com/mezgoodle/TestLabs-kpi/tree/master/Lab3) з програмою.
2. [Папка](https://github.com/mezgoodle/TestLabs-kpi/tree/master/docs) з діаграмами.
3. [Результати](https://github.com/mezgoodle/TestLabs-kpi/actions/workflows/dotnet.yml) CI.