МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

ОТЧЕТ

по лабораторной работе № 6
на тему: «Разработка собственного тестового драйвера»
по дисциплине: «Качество и тестирование программного обеспечения»
Вариант 18

Выполнил: Шорин В.Д. Шифр: 171406
Институт приборостроения, автоматизации и информационных технологий
Направление: 09.03.04 «Программная инженерия»
Группа: 71ПГ
Проверили: Ужаринский А.Ю., Конюхова О.В.

Отметка о зачете:

Дата: «____» _____ 2020 г.

Орел, 2020 г.

Задание на лабораторную работу

Необходимо разработать программу тестового драйвера для тестирования программных модулей. На вход данная программа получает путь к исходному коду тестируемого модуля. Также драйвер на вход получает файл с описанием тестов, которые нужно выполнить. Описание теста включает в себя название функции, параметры, которые должны быть переданы в функцию и ожидаемый результат. Структура файла с тестами может быть произвольной. Результатом работы тестового драйвера должен быть отчёт о результатах тестирования. Отчёт должен содержать следующую информацию: номер теста, полученный результат, ожидаемый результат, результат прохождения теста (success или faile). Если результат прохождения теста faile, то нужно вывести причину ошибки.

Выполнение

Имена модуля для тестов и файла с тестами вводятся в консоли Драйвер загружает все тесты и тестируемый модуль, затем поочередно запускает каждый из тестов. При выполнении теста могут возникнуть следующие ошибки:

- ожидаемый и полученный результаты не совпадают
- отсутствуют требуемые для теста данные (нет какого-то поля в тесте)
- ошибка при выполнении тестируемый функции

Получение каждой ошибки сопровождается описанием этой ошибки.

В качестве формата файла с тестами был выбран XML.

Результат выполнения тестов

При выполнении первых двух тестов ошибки не возникло. При выполнении теста №3 произошла ошибка отсутствия какого-либо параметра, о чем сообщается в описание ошибки теста (рисунок 1).

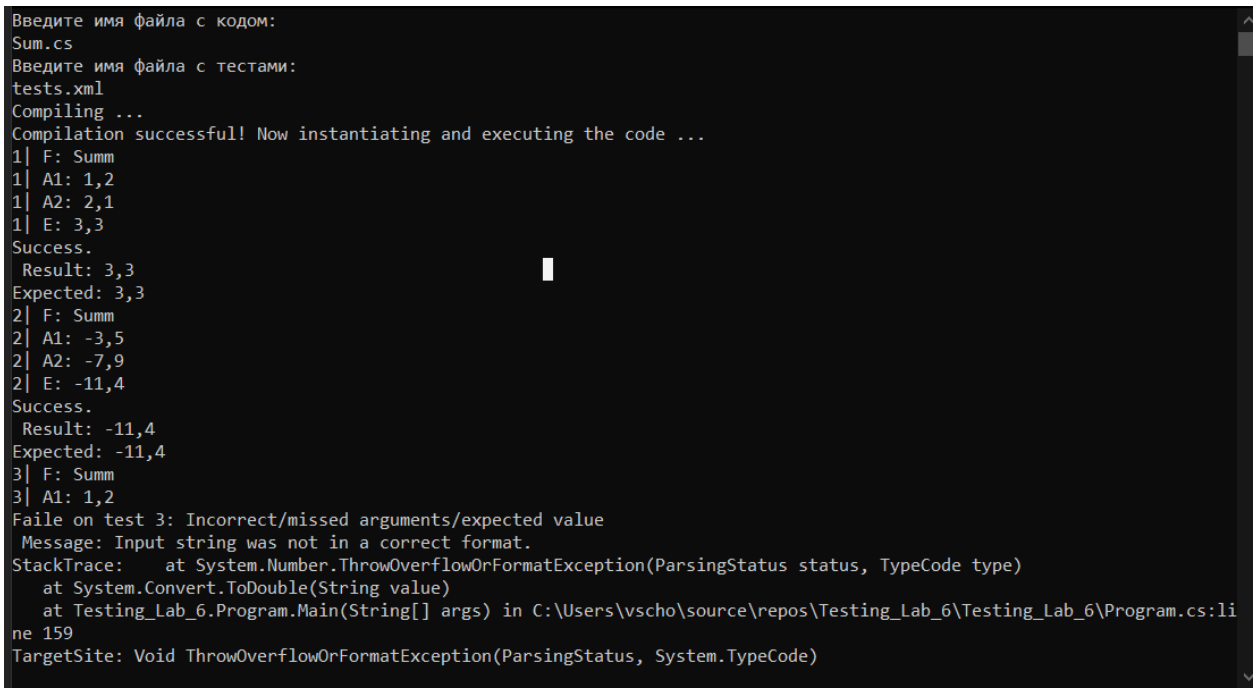

Рисунок 1 – Результат выполнения тестов

Код

«Program.cs»

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Runtime.Loader;
using System.Xml;
using Microsoft.CodeAnalysis;
using Microsoft.CodeAnalysis.CSharp;
using Microsoft.CodeAnalysis.Emit;

namespace Testing_Lab_6
{
    class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Введите имя файла с кодом:");
            string fileCode = Console.ReadLine();
            Console.WriteLine(fileCode);
```

```
            Console.WriteLine("Введите имя файла с тестами:");
            string fileTests = Console.ReadLine();
            Console.WriteLine(fileTests);

            string pathCode = "../../../" + fileCode;
            string pathTests = "../../../" + fileTests;

            string codeToCompile = "";

            try
            {
                using (StreamReader sr = new StreamReader(pathCode))
                {
                    codeToCompile = sr.ReadToEnd();
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                return;
            }
            SyntaxTree syntaxTree = CSharpSyntaxTree.ParseText(codeToCompile);

            string assemblyName = Path.GetRandomFileName();
            var refPaths = new[] {
                typeof(object).GetTypeInfo().Assembly.Location,
                typeof(Console).GetTypeInfo().Assembly.Location,

Path.Combine(Path.GetDirectoryName(typeof(System.Runtime.GCSettings).GetTypeInfo().Assembly.Location),
"System.Runtime.dll")
            };
            MetadataReference[] references = refPaths.Select(r =>
MetadataReference.CreateFromFile(r)).ToArray();

            Console.WriteLine("Compiling ...");
            CSharpCompilation compilation = CSharpCompilation.Create(
                assemblyName,
                syntaxTrees: new[] { syntaxTree },
                references: references,
                options: new CSharpCompilationOptions(OutputKind.DynamicallyLinkedLibrary));

            using (var ms = new MemoryStream())
```

```
{
    EmitResult result = compilation.Emit(ms);

    if (!result.Success)
    {
        Console.WriteLine("Compilation failed!");
        IEnumerable<Diagnostic> failures = result.Diagnostics.Where(diagnostic =>
            diagnostic.IsWarningAsError ||
            diagnostic.Severity == DiagnosticSeverity.Error);

        foreach (Diagnostic diagnostic in failures)
        {
            Console.Error.WriteLine("\t{0}: {1}", diagnostic.Id, diagnostic.GetMessage());
        }
    }
    else
    {
        Console.WriteLine("Compilation successful! Now instantiating and executing the code ...");
        ms.Seek(0, SeekOrigin.Begin);

        Assembly assembly = AssemblyLoadContext.Default.LoadFromStream(ms);
        var type = assembly.GetType("RoslynCompileSample.Sum");
        var instance = assembly.CreateInstance("RoslynCompileSample.Sum");

        XmlDocument xmlDocument = new XmlDocument();
        xmlDocument.Load(pathTests);
        XmlElement xRoot = xmlDocument.DocumentElement;

        string functionName = "";
        List<double> arguments = new List<double>() { 0, 0 };
        double expectedValue = 0;
        int testNumber = 1;

        foreach(XmlNode test in xRoot)
        {
            foreach (XmlNode childNode in test)
            {
                if (childNode.Name == "function")
                {
                    try
                    {
                        functionName = childNode.InnerText;
```

```
                    Console.WriteLine($"{testNumber}| F: {functionName}");
                }
                catch (Exception e)
                {
                    Console.WriteLine($"Faile on test {testNumber}: Incorrect/missed
arguments/expected value \n " +
                                $"Message: {e.Message} \n" +
                                $"StackTrace: {e.StackTrace}\n" +
                                $"TargetSite: {e.TargetSite}");
                    return;
                }
            }
            if (childNode.Name == "arg1")
            {
                try
                {
                    arguments[0] = Convert.ToDouble(childNode.InnerText);
                    Console.WriteLine($"{testNumber}| A1: {arguments[0]}");
                }
                catch (Exception e)
                {
                    Console.WriteLine($"Faile on test {testNumber}: Incorrect/missed
arguments/expected value \n " +
                                $"Message: {e.Message} \n" +
                                $"StackTrace: {e.StackTrace}\n" +
                                $"TargetSite: {e.TargetSite}");
                    return;
                }
            }
            if (childNode.Name == "arg2")
            {
                try
                {
                    arguments[1] = Convert.ToDouble(childNode.InnerText);
                    Console.WriteLine($"{testNumber}| A2: {arguments[1]}");
                }
                catch (Exception e)
                {
                    Console.WriteLine($"Faile on test {testNumber}: Incorrect/missed
arguments/expected value \n " +
                                $"Message: {e.Message} \n" +
                                $"StackTrace: {e.StackTrace}\n" +
```

```
                        $"TargetSite: {e.TargetSite}");
                    return;
                }
            }
            if (childNode.Name == "expected")
            {
                try
                {
                    expectedValue = Convert.ToDouble(childNode.InnerText);
                    Console.WriteLine($"{testNumber}| E: {expectedValue}");
                }
                catch (Exception e)
                {
                    Console.WriteLine($"Faile on test {testNumber}: Incorrect/missed
arguments/expected value \n " +
                        $"Message: {e.Message} \n" +
                        $"StackTrace: {e.StackTrace}\n" +
                        $"TargetSite: {e.TargetSite}");
                    return;
                }
            }
        }

        var meth = type.GetMember(functionName).First() as MethodInfo;

        object methodResult = 0;
        try
        {
            methodResult = meth.Invoke(instance, new object[] { arguments[0], arguments[1] });
        }
        catch (Exception e)
        {
            Console.WriteLine($"Faile on test {testNumber}: Error in executing the function
{functionName} \n " +
                        $"Message: {e.Message} \n" +
                        $"StackTrace: {e.StackTrace}\n" +
                        $"TargetSite: {e.TargetSite}");
        }

        if (expectedValue.CompareTo(Convert.ToDouble(methodResult)) == 0)
        {
            Console.WriteLine($"Success. \n Result: {methodResult}\n" +
```

```
                                $"Expected: {expectedValue}");
                        }
                        else
                        {
                            Console.WriteLine($"Faile on test {testNumber}: Error in executing the function
{functionName}\n" +
                                $"Result: {methodResult}\n" +
                                $"Expected: {expectedValue}");

                        }

                        testNumber++;
                    }
                }
            }
        }
    }
```

«Sum.cs»

```
using System;

namespace RoslynCompileSample
{
    public class Sum
    {
        public static double Summ(double a, double b)
        {
            return a + b;
        }
    }
}
```

«tests.xml»

```
<?xml version="1.0" encoding="utf-8"?>
<tests>
  <test>
    <function>Summ</function>
    <arg1>1,2</arg1>
    <arg2>2,1</arg2>
    <expected>3,3</expected>
  </test>
  <test>
```

```
    <function>Summ</function>
    <arg1>-3,5</arg1>
    <arg2>-7,9</arg2>
    <expected>-11,4</expected>
  </test>
  <test>
    <function>Summ</function>
    <arg1>1,2</arg1>
    <arg2></arg2>
    <expected>0</expected>
  </test>
</tests>
```