Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГТУ»)**

**Институт компьютерных систем и информационной безопасности**

**Кафедра информационных систем и программирования**

**ЛАБОРАТОРНАЯ РАБОТА №9**

**Дисциплина: тестирование и отладка программного обеспечения**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Я.С Давков

Направление подготовки: 09.03.04 Программная инженерия

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Г. Волик

**1 Цель работы**

Изучить подход к автоматизации создания тестовых двойников (test doubles) при помощи библиотеки NSubstitute.

**2 Краткая теория**

Существует категория классов, которые тестировать весьма просто. Если класс зависит только от примитивных типов данных и не имеет никаких связей с другими сущностями, то достаточно создать экземпляр этого класса, проверить его путем вызовов методов и проверки ожидаемого состояния. Но количество классов, которые соответствуют данным требованиям ограничено. При проведении автоматизированного тестирования кода, часто приходится сталкиваться с проблемой необходимости тестирования кода, часть которого еще не дописана, но какая-то функциональность есть и ее нужно протестировать. В этом случае нужно создавать заглушки для классов, интерфейсов, методов, которые используются в тестируемом методе. Делать это вручную очень утомительно, т.к. нужно писать много повторяющегося кода. Для автоматизации этого процесса существуют множество наборов библиотек (фреймворков), называемых isolation framework или mocking framework, в том числе и для языка C#. Одним из них является NSubstitute. Создаваемые вручную макеты имеют не только преимущества, но и недостатки, один из которых заключается в растущем потоке кода, вызываемом предоставлением основанной на делегатах реализации для каждого модульного теста – особенно если при этом еще и записывать вызовы, на манер тестового шпиона. Крупнейшим же недостатком, вероятно, является то, что работа по написанию делегатов как реализаций быстро становится утомительной и подверженной ошибкам.

**3 Задание**

1) На основе класса из предыдущей лабораторной работы (вариант задания оставить прежним) создать тестирующий проект, создающий тестовых двойников при помощи NSubstitute 2) Протестировать класс. 3) Оформить отчёт

**4 Выполнение работы**

Мой вариант – 4, реализуемый алгоритм - Проверка на наличие циклов в графе, Входной формат - Матрица смежности, Внутреннее представление - Матрица смежности, Выходной формат - Матрица инцидентности

Листинг кода предыдущей работы:

public class MyGraph  
{  
 public int[,] AdjacencyMatrix { get; }  
  
 public int VertexCount => AdjacencyMatrix.GetLength(0);  
  
 public MyGraph(int[,] adjacencyMatrix)  
 {  
 AdjacencyMatrix = adjacencyMatrix ?? throw new ArgumentNullException(nameof(adjacencyMatrix));  
 }  
  
 public bool HasCycle()  
 {  
 bool[] visited = new bool[VertexCount];  
 bool[] stack = new bool[VertexCount];  
  
 for (int i = 0; i < VertexCount; i++)  
 if (DetectCycleDFS(i, visited, stack))  
 return true;  
  
 return false;  
 }  
  
 private bool DetectCycleDFS(int v, bool[] visited, bool[] stack)  
 {  
 if (!visited[v])  
 {  
 visited[v] = true;  
 stack[v] = true;  
  
 for (int j = 0; j < VertexCount; j++)  
 {  
 if (AdjacencyMatrix[v, j] == 1)  
 {  
 if (!visited[j] && DetectCycleDFS(j, visited, stack))  
 return true;  
 else if (stack[j])  
 return true;  
 }  
 }  
 }  
  
 stack[v] = false;  
 return false;  
 }  
  
 public int[,] ToIncidenceMatrix()  
 {  
 var edges = new List<(int from, int to)>();  
  
 for (int i = 0; i < VertexCount; i++)  
 for (int j = 0; j < VertexCount; j++)  
 if (AdjacencyMatrix[i, j] == 1)  
 edges.Add((i, j));  
  
 int[,] incidence = new int[VertexCount, edges.Count];  
  
 for (int k = 0; k < edges.Count; k++)  
 {  
 incidence[edges[k].from, k] = 1;  
 incidence[edges[k].to, k] = -1;  
 }  
  
 return incidence;  
 }  
}

public class MatrixGraphWriter : IGraphWriter  
{  
 public void Write(TextWriter writer, int[,] matrix)  
 {  
 int rows = matrix.GetLength(0);  
 int cols = matrix.GetLength(1);  
  
 for (int i = 0; i < rows; i++)  
 {  
 for (int j = 0; j < cols; j++)  
 {  
 writer.Write(matrix[i, j]);  
 if (j < cols - 1) writer.Write(' ');  
 }  
 writer.WriteLine();  
 }  
 }  
}

public class MatrixGraphReader : IGraphReader  
{  
 public int[,] Read(TextReader reader)  
 {  
 var lines = new List<string>();  
 string? line;  
 while ((line = reader.ReadLine()) != null)  
 {  
 if (!string.IsNullOrWhiteSpace(line))  
 lines.Add(line.Trim());  
 }  
  
 int n = lines.Count;  
 int[,] matrix = new int[n, n];  
  
 for (int i = 0; i < n; i++)  
 {  
 var parts = lines[i].Split(' ', StringSplitOptions.*RemoveEmptyEntries*);  
 for (int j = 0; j < n; j++)  
 matrix[i, j] = int.Parse(parts[j]);  
 }  
  
 return matrix;  
 }  
}

public interface IGraphWriter  
{  
 void Write(TextWriter writer, int[,] matrix);  
}

public interface IGraphReader  
{  
 int[,] Read(TextReader reader);  
}

public class GraphSaver  
{  
 private readonly IGraphWriter writer;  
  
 public GraphSaver(IGraphWriter writer)  
 {  
 this.writer = writer ?? throw new ArgumentNullException(nameof(writer));  
 }  
  
 public void Save(string path, int[,] matrix)  
 {  
 string dir = Path.GetDirectoryName(path);  
 if (!string.IsNullOrEmpty(dir) && !Directory.Exists(dir))  
 Directory.CreateDirectory(dir);  
  
 using var sw = new StreamWriter(path);  
 writer.Write(sw, matrix);  
 }  
}

public class GraphFileProcessor  
{  
 private readonly IGraphReader reader;  
 private readonly IGraphWriter writer;  
  
 public GraphFileProcessor(IGraphReader reader, IGraphWriter writer)  
 {  
 this.reader = reader ?? throw new ArgumentNullException(nameof(reader));  
 this.writer = writer ?? throw new ArgumentNullException(nameof(writer));  
 }  
  
 public int[,] Load(string path)  
 {  
 using var sr = new StreamReader(path);  
 return reader.Read(sr);  
 }  
  
 public void Save(string path, int[,] matrix)  
 {  
 using var sw = new StreamWriter(path);  
 writer.Write(sw, matrix);  
 }  
}

Листинг тестов с использованием тестовых двойников:

[TestFixture]  
public class WithNSubstituteTests  
{  
 private IGraphReader readerMock;  
 private IGraphWriter writerMock;  
 private GraphFileProcessor processor;  
 private string tempFilePath;  
  
 [SetUp]  
 public void Setup()  
 {  
 readerMock = Substitute.For<IGraphReader>();  
 writerMock = Substitute.For<IGraphWriter>();  
 processor = new GraphFileProcessor(readerMock, writerMock);  
  
 tempFilePath = Path.GetTempFileName();  
 File.WriteAllText(tempFilePath, "0 1\n1 0");  
 }  
  
 [TearDown]  
 public void Cleanup()  
 {  
 if (File.Exists(tempFilePath))  
 File.Delete(tempFilePath);  
 }  
  
 [Test]  
 public void Load\_UsesReaderReadMethod()  
 {  
 readerMock.Read(Arg.Any<TextReader>())  
 .Returns(new int[,] { { 0, 1 }, { 1, 0 } });  
  
 int[,] result = processor.Load(tempFilePath);  
  
 readerMock.Received(1).Read(Arg.Any<TextReader>());  
  
 Assert.That(result.GetLength(0), Is.EqualTo(2));  
 Assert.That(result.GetLength(1), Is.EqualTo(2));  
 }  
  
 [Test]  
 public void Save\_UsesWriterWriteMethod()  
 {  
 string dummyPath = Path.Combine(Path.GetTempPath(), "out.txt");  
 int[,] matrix = new int[,] { { 1, 0 }, { 0, 1 } };  
  
 processor.Save(dummyPath, matrix);  
  
 writerMock.Received(1).Write(Arg.Any<TextWriter>(), matrix);  
  
 if (File.Exists(dummyPath))  
 File.Delete(dummyPath);  
 }  
}

Запустим тесты и проверим что все выполняется корректно

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 1 – результат запуска тестов