Министерство науки И ВЫСШЕГО ОБРАЗОВАНИЯ Российской федерации

федеральное государственное бюджетное образование учреждение высшего образования **«**КУРГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**»**

Кафедра «Программное обеспечение автоматизированных систем»

КОНТРОЛЬНАЯ РАБОТА

**по дисциплине**

**«УПРАВЛЕНИЕ КАЧЕСТВОМ И ТЕСТИРОВАНИЕ ПО»**

**Вариант 20**

**РФ КГУ 09.03.04КР23 010007**

Выполнил студент гр. ИТ-0940320 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Третьяков Н.Ю. /

Проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Семахин А.М /

Курган 2023

СОДЕРЖАНИЕ

[КОНТРОЛЬНАЯ РАБОТА 1](#_Toc155723715)

[СОДЕРЖАНИЕ 2](#_Toc155723716)

[ВВЕДЕНИЕ 3](#_Toc155723717)

[ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ 4](#_Toc155723718)

[ПОСТАНОВКА ЗАДАЧИ 5](#_Toc155723719)

[ВЫБОР ПРОГРАММНЫХ СРЕДСТВ 6](#_Toc155723720)

[РЕШЕНИЕ ЗАДАЧИ 7](#_Toc155723721)

[АВТОМАТИЗИРОВАННОЕ ЮНИТ-ТЕСТИРОВАНИЕ 11](#_Toc155723722)

[РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ РАБОТЫ 13](#_Toc155723723)

ВВЕДЕНИЕ

В эпоху информационных технологий качество и надежность программного обеспечения становятся критически важными. Управление качеством и тестирование ПО являются ключевыми элементами процесса разработки, обеспечивающими соответствие продукта требованиям и ожиданиям пользователя.

Управление качеством ПО представляет собой систематический подход к контролю и улучшению процесса разработки и самого продукта. Это включает планирование, контроль, управление и улучшение качества.

Тестирование ПО – это процесс исследования продукта для обнаружения и исправления ошибок и дефектов. Оно играет важную роль в обеспечении корректной работы продукта и его соответствия требованиям пользователя.

Своевременное обнаружение и устранение дефектов на этапе разработки критически важно для обеспечения качества ПО. Ошибки, обнаруженные на ранних стадиях, обычно проще и дешевле исправить, что помогает предотвратить возникновение серьезных проблем в будущем.

В целом, управление качеством и тестирование ПО играют важную роль в создании надежного и высококачественного продукта.

ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

Существуют два метода сборки модулей:

* монолитный, характеризующийся одновременным объединением модулей в тестируемый комплекс;
* инкрементальный, характеризующийся пошаговым (помодульным) наращиванием комплекса программ с пошаговым тестированием собираемого комплекса.

В инкрементальном методе выделяют две стратегии добавления модулей:

* восходящее тестирование;
* нисходящее тестирование.

При восходящем тестировании сначала тестируются терминальные модули, независящие от других модулей, затем тестируются модули, которые зависят от проверенных модулей. Для модуля разрабатывают драйвер.

Драйвер – программный модуль, эмулирующий окружение модуля при тестировании. Драйвер вызывает тестируемый модуль, передает тестовые данные, проверяет результаты работы модуля и корректность реализации его интерфейса. При восходящем тестировании упрощается локализация ошибок.

При нисходящем тестировании сначала тестируют верхний управляющий модуль программной системы без модулей низкого уровня, вместо которых используют заглушки, затем тестируют модули более низкого уровня.

Заглушка – программный модуль, обладающий тем же интерфейсом, что и замещаемый модуль нижележащего уровня. Заглушка не реализует функциональность замещаемого модуля, а возвращает значения, позволяющие проверить функционирование тестируемого модуля.

ПОСТАНОВКА ЗАДАЧИ

Дан одномерный динамический массив целых чисел. Необходимо вычислить:

1) сумму всех элементов массива;

2) количество элементов массива;

3) средние арифметическое значение всех элементов.

Решение каждой конкретной задачи должно быть вынесено в отдельный модуль программы.

Тестирование каждого модуля осуществляется с помощью драйвера.

Тестирование является интеграционным, восходящим.

Тестирование необходимо осуществить с помощью фреймворка xUnit.

ВЫБОР ПРОГРАММНЫХ СРЕДСТВ

Для выполнения контрольной работы, направленной на тестирование программного кода, был определён набор следующих инструментов.

1. Язык программирования: C# 10

2. Интегрированная среда разработки (IDE): Visual Studio 2022 Community Edition

3. Платформа разработки: .NET 7.0

4. Библиотека для автоматизированного тестирования: xUnit

РЕШЕНИЕ ЗАДАЧИ

Для создания модулей понадобятся следующие интерфейсы:

1. ICountModule

public interface ICountModule

{

string Sequence { get; set; }

public int Count();

}

Описание:

* Sequence – свойство, которое содержит последовательность чисел в формате строки.
* Метод Count возвращает количество элементов последовательности.

2. ISumModule

public interface ISumModule

{

string Sequence { get; set; }

public int Sum();

}

Описание:

* Sequence – свойство, которое содержит последовательность чисел в формате строки.
* Метод Sum возвращает сумму элементов последовательности.

Реализация интерфейсов осуществляется с помощью следующих классов:

1. SumModule

public class SumModule : ISumModule

{

public string Sequence { get; set; }

public SumModule(string sequence) {

Sequence = sequence;

}

public int Sum()

{

List<int> sequenceOfNumbers =

SequenceConverts.ConvertToList(Sequence);

return sequenceOfNumbers.Sum();

}

}

Описание:

* Sequence – свойство, которое содержит последовательность чисел в формате строки.
* SumModule – конструктор данного класса.
* Метод Sum возвращает сумму элементов последовательности.

2. CountModule

public class CountModule : ICountModule

{

public string Sequence { get; set; }

public CountModule(string sequence)

{

Sequence = sequence;

}

public int Count()

{

List<int> sequenceOfNumbers =

SequenceConverts.ConvertToList(Sequence);

return sequenceOfNumbers.Count;

}

}

Описание:

* Sequence – свойство, которое содержит последовательность чисел в формате строки.
* CountModule – конструктор данного класса.
* Метод Count возвращает количество элементов последовательности.

3. SumPlug (модуль-заглушка)

public class SumPlug : ISumModule

{

public string Sequence { get; set; }

public SumPlug(string sequence)

{

sequence = sequence;

}

public int Sum()

{

Random random = new Random();

return random.Next(1, 10000);

}

}

Описание:

* Sequence – свойство, которое содержит последовательность чисел в формате строки.
* SumPlug – конструктор данного класса.
* Метод Sum возвращает фиктивную сумму.

4. CountPlug (модуль-заглушка)

public class CountPlug : ICountModule

{

public string Sequence { get; set; }

public CountPlug(string sequence)

{

sequence = sequence;

}

public int Count()

{

Random random = new Random();

return random.Next(1, 100);

}

}

Описание:

* Sequence – свойство, которое содержит последовательность чисел в формате строки.
* CountPlug – конструктор данного класса.
* Метод Count возвращает фиктивное количество.

5. ArithmeticMeanModule

public class ArithmeticMeanModule

{

ISumModule SumModule;

ICountModule CountModule;

public ArithmeticMeanModule(ISumModule sumModule, ICountModule countModule)

{

SumModule = sumModule;

CountModule = countModule;

}

public double ArithmeticMean()

{

double sum = SumModule.Sum();

int count = CountModule.Count();

double result = sum / count;

return Math.Round(result,2);

}

}

Описание:

* SumModule – свойство, которое содержит экземпляр модуля для подсчёта суммы значений элементов последовательности.
* CountModule – свойство, которое содержит экземпляр модуля для подсчёта количества элементов последовательности.
* ArithmeticMeanModule – конструктор данного класса. Принимает на вход интерфейсы модуля суммы и количества.
* ArithmeticMean – метод для подсчёта среднего арифметического значения.

6. SumDriver

public class SumDriver

{

public ISumModule SumModule;

public SumDriver(ISumModule sumModule) {

SumModule = sumModule;

}

public int Sum() {

return SumModule.Sum();

}

}

Описание:

* SumModule – свойство, которое содержит экземпляр модуля для подсчёта суммы значений элементов последовательности.
* SumDriver – конструктор данного класса.
* Sum – метод, который запускает модуль суммы.

7. CountDriver

public class CountDriver

{

public ICountModule CountModule;

public CountDriver(ICountModule countModule)

{

CountModule = countModule;

CountModule.Sequence = CountModule.Sequence;

}

public int Count() { return CountModule.Count(); }

}

Описание:

* CountModule – свойство, которое содержит экземпляр модуля для подсчёта количества элементов последовательности.
* CountDriver – конструктор данного класса.
* Count – метод, который запускает модуль количества.

АВТОМАТИЗИРОВАННОЕ ЮНИТ-ТЕСТИРОВАНИЕ

Код содержит юнит-тесты для функций подсчёта суммы и количества элементов, а также – функций запуска модулей и их корректной работы.

1. Тестирование SumModule.Sum():

[Fact]

public void SumModuleTest()

{

//Создаём экземпляр модуля суммы

var sequence = "3 4 5";

var sumModule = new SumModule(sequence);

//Ожидаемый результат

var expectedResult = 12;

//Фактический результат

var actualResult = sumModule.Sum();

//Test

Assert.Equal(expectedResult, actualResult);

}

2. Тестирование CountModule.Count():

[Fact]

public void CountModuleTest()

{

//Создаём экземпляр модуля счётчика

var sequence = "3 4 5";

var countModule = new CountModule(sequence);

//Ожидаемый результат

var expectedResult = 3;

//Фактический результат

var actualResult = countModule.Count();

//Test

Assert.Equal(expectedResult, actualResult);

}

3. Тестирование ArithmeticMeanModule.ArithmeticMean():

[Fact]

public void ArithmeticMeanModuleTest()

{

//Создаём экземпляр модулей счётчика и суммы

var sequence = "3 4 5";

var countModule = new CountModule(sequence);

var sumModule = new SumModule(sequence);

//Создаём экземпляр модуля среднего арифметического

var aritmeticMeanModule = new ArithmeticMeanModule(sumModule, countModule);

//Ожидаемый результат

var expectedResult = 4;

//Фактический результат

var actualResult = aritmeticMeanModule.ArithmeticMean();

//Test

Assert.Equal(expectedResult, actualResult);

}

4. Тестирование SumDriver.Sum():

[Fact]

public void SumDriverTest()

{

//Создаём экземпляр модуля суммы и драйвера модуля суммы

var sequence = "3 4 5";

var sumModule = new SumModule(sequence);

var sumDriver = new SumDriver(sumModule);

//Ожидаемый результат

var expectedResult = sumModule.Sum();

//Фактический результат

var actualResult = sumDriver.Sum();

//Test

Assert.Equal(expectedResult, actualResult);

}

2. Тестирование CountDriver.Count():

[Fact]

public void CountDriverTest()

{

//Создаём экземпляр модуля счётчика

var sequence = "3 4 5";

var countModule = new CountModule(sequence);

var countDriver = new CountDriver(countModule);

//Ожидаемый результат

var expectedResult = countModule.Count();

//Фактический результат

var actualResult = countDriver.Count();

//Test

Assert.Equal(expectedResult, actualResult);

}

РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ РАБОТЫ

Результаты выполнения тестирования приведены на изображении ниже (рисунок 1).

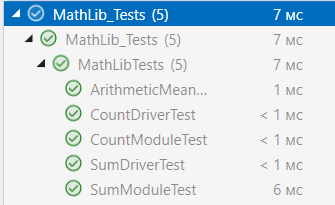


Рисунок 1 – Результаты выполнения юнит-тестирования