Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Институт повышения квалификации
и переподготовки

Кафедра «Информатика»

В. С. Мурашко

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

ПРАКТИКУМ
по одноименной дисциплине
для слушателей специальности переподготовки
1-40 01 73 «Программное обеспечение
информационных систем»
заочной формы обучения

Гомель 2019

УДК 004.921(075.8)
ББК 32.973-018я73
М91

Рекомендовано кафедрой «Информатика» ГГТУ им. П. О. Сухого
(*протокол № 6 от 28.11.2018 г.*)

Рецензент: доц. каф. «Информационные технологии» ГГТУ им. П. О. Сухого
канд. техн. наук, доц. *В. И. Токочаков*

Мурашко, В. С.

М91 Тестирование программного обеспечения : практикум по одноим. дисциплине для слушателей специальности переподготовки 1-40 01 73 «Программное обеспечение информационных систем» заоч. формы обучения / В. С. Мурашко. – Гомель : ГГТУ им. П. О. Сухого, 2019. – 89 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://elib.gstu.by. – Загл. с титул. экрана.

В практикуме представлено шесть работ по дисциплине «Тестирование программного обеспечения». Даны варианты заданий и порядок выполнения работ с необходимыми методическими указаниями.

Издание адресовано слушателям ИПКиП специальности 1-40 01 73 «Программное обеспечение информационных систем».

УДК 004.921(075.8)
ББК 32.973-018я73

Содержание

ЛАБОРАТОРНАЯ РАБОТА №1 «ТЕСТИРОВАНИЕ ТРЕБОВАНИЙ»

Постановка задачи

Вы – системный аналитик. Ваша задача написать техническое задание на предмет, маркетинговые требования к которому Вам были переданы в соответствии с вариантом (в рамках лабораторной работы делать этого не надо, нужно только решить, достаточно ли для этого информации). Для этого Вы должны раскрыть требования маркетинга на уровень пользователей. В процессе этого у Вас могут возникать различные проблемы.

В лабораторной работе необходимо проанализировать требования заказчика, указать на проблемы в требованиях (каждую ошибку отнести к соответствующей категории) и скорректировать требования таким образом, чтобы в результате получился предмет указанного наименования. Для каждого пункта требований описать, каким образом будет производиться его проверка.

Методические указания

Тестирование программного обеспечения должно начинаться с самого раннего этапа – этапа формулировки требований. Основными проблемами требований являются:

- некорректность;
- двусмысленность;
- неполнота;
- непроверяемость;
- несвязанность между требованиями разных уровней (нетрассируемость);
- непонятность.

Требования к программному обеспечению формулируются на трех уровнях. Самый верхний – уровень маркетинговых требований – содержит самые общие требования. Следующий уровень – уровень пользователей – содержит, обычно, описание бизнес-процессов. На самом низком уровне – уровне программистов – используются мелкие технические требования.

Типовой проблемой при выполнении лабораторной работы является попытка придраться к каждому слову спецификации. На самом деле, предполагается моделирование реальной ситуации – Вам прислали требования заказчика и Вы делаете по ним техническое задание, по мере выполнения которого обнаруживаете разнообразные проблемы, препятствующие этому.

Заказчик никогда сам не пишет техническое задание. Когда в ответ на требование вида «разработать двухкамерный холодильник» Вы пишете замечания вида «неизвестно какого он должен быть размера, цвета и формы» Вы, тем самым, говорите заказчику, что он не сделал Вашу работу, потому что определить размер, цвет и форму, если они не указаны заказчиком, – задача аналитика.

Еще одной важной задачей является произвести на заказчика впечатление грамотного специалиста, а не человека, который не умеет делать свою работу.

Пример выполнения

Спецификация на разработку стула.

1. Стул должен иметь четыре ножки и горизонтальную поверхность для сидения.
2. Стул должен иметь возможность регулирования высоты
3. Стул должен быть удобным.
4. Стул должен иметь высоту 60 сантиметров.
5. Стул должен весить не более 500 грамм
6. Стул должен быть легко перемещаем по помещению.
7. Стул не должен царапать паркет при перемещении
8. Стул должен использовать только нетоксичные материалы.

Рассмотрим требования по очереди.

Требование номер один интересно тем, что в нем упоминаются только ножки и поверхность для сидения. Стул, у которого нет спинки, называется «табурет». Имеет смысл уточнить, имеется ли в самом деле в виду табурет или спинка была просто забыта при описании.

Требование номер два обычно используется для стульев имеющих одну ножку, хотя принципиальных проблем с реализацией для четырехногого стула не имеется.

Требование номер три невозможно проверить, его можно скорректировать разнообразными способами, например, «дизайн стула утверждается заказчиком» в том смысле, что сперва будет утвержден дизайн и только после этого будет продолжена реализация.

Требование номер 4 явно противоречит требованию номер два, необходимо указать диапазон изменения высоты либо отказаться от требования номер два. Кроме того, возникает вопрос – является ли указанная высота высотой сиденья или спинки.

Варианты заданий

1 Спецификация на разработку холодильника

Необходимо разработать двухкамерный холодильник на базе системы андроид, отвечающий следующим требованиям:

1. Холодильник двухкамерный.
2. При захлопывании дверцы она всегда обеспечивает плотное прижатие, вне зависимости от того, с какой силой было произведено это действие.
3. Холодильник имеет интерфейс через сенсорный дисплей с локализацией, поддерживающий следующие языки: Русский, Английский.

4. Управление температурой в холодильной и морозильной камерах осуществляется с дисплея.
5. Когда дверца холодильника открыта, дисплей показывает предупреждающее сообщение и не разрешает управление температурой.
6. Когда дверца холодильника закрыта, дисплей отображает текущую температуру в холодильной и морозильной камерах.
7. При изменении температуры на N градусов фактическая температура в камере должна измениться через N минут.

2 Спецификация на разработку пылесоса

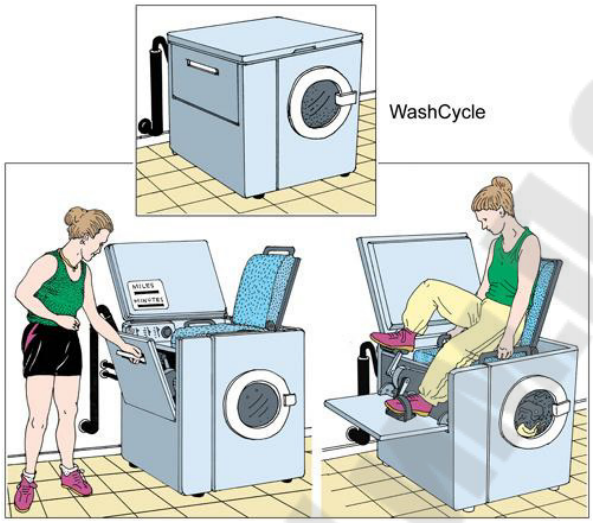
Необходимо разработать пылесос на базе системы андроид, отвечающий следующим требованиям:

1. Пылесос способен убирать пыль и мелкий мусор.
2. Пылесос обеспечивает всасывание воздуха с мощностью 1600 Ватт.
3. Масса пылесоса в процессе работы не должна превышать 5 килограмм.
4. Пылесос может быть использован для сбора пыли на любых поверхностях и под любыми предметами мебели.
5. На пылесосе должна быть предусмотрена ручка.
6. Заряда пылесос должно хватать на 1 час работы.

3 Спецификация на разработку стиральной машины

Необходимо разработать стиральную машину, отвечающую следующим требованиям:

1. Стиральная машина должна уметь стирать белье.
2. Минимальная загрузка должна составлять пять килограмм
3. Стирка должна осуществляться в двух режимах – быстрая и полная, а также машина должна уметь осуществлять полоскание.



4. Стиральная машина должна подключаться к водопроводной трубе, сама закачивать воду, нагревать ее до нужной температуры, по окончании стирки – сливать.
5. У машины должен быть дисплей, демонстрирующий пользователю полезную информацию.
6. Машина должна подключаться к WiFi.

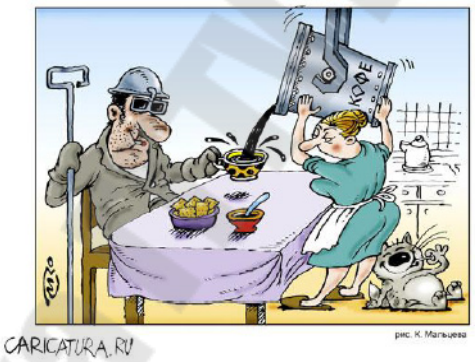
4 Спецификация на разработку микроволновой печи

Необходимо разработать микроволновую печь, отвечающую следующим требованиям.

1. Микроволновая печь должна уметь нагревать продукты, а также осуществлять разморозку.
2. Микроволновая печь должна иметь дверцу.
3. Микроволновая печь во включенном состоянии не должна производить опасное для человека излучение.
4. Нагревать продукты в микроволновой печи нужно при закрытой дверце.
5. Микроволновая печь должна позволять помещать вместе с продуктами металлические столовые приборы и посуду, при этом они не должны нагреваться или искрить.
6. Микроволновая печь должна иметь подсветку.
7. Микроволновая печь не должна сушить продукты, которые в ней готовятся.

5 Спецификация на разработку электрокофеварки

Необходимо разработать электрокофеварку, отвечающий следующим требованиям.



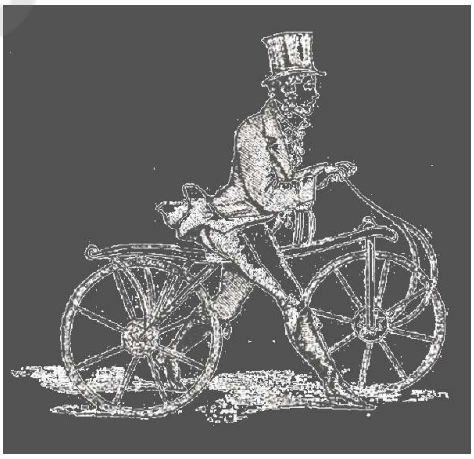
1. Кофеварка должна уметь варить кофе.
2. Кофеварка должна иметь кнопку включения, ручку и крышку над емкостью для наливания воды.
3. Кофеварка должна работать от электричества.
4. Кнопка должна включаться, только если крышка закрыта.
5. Воду в кофеварку можно наливать, только если крышка закрыта
6. Кофеварка должен поддерживать протокол HTCPCP (RFC 2324). Заказчик особенно настаивает на этом требовании и отказывается его убирать.
7. Кофеварка должна быть красивой.

6 Спецификация на разработку газовой плиты

Необходимо разработать газовую плиту, отвечающую следующим требованиям.

1. Плита должна иметь четыре конфорки и духовку со стеклянной дверцей.
2. Когда духовка открыта, газ подаваться не должен.
3. На передней панели плиты должен быть расположен датчик температуры.
4. Духовка должна быть оборудована таймером, выключающим газ, и зуммером с настраиваемой мелодией, срабатывающим по завершении.
5. На плите можно готовить весь набор популярных блюд.
6. Плита должна исключать возможность возникновения пожара.

7 Спецификация на разработку велосипеда



Необходимо разработать велосипед, отвечающий следующим требованиям.

1. У велосипеда должно быть удобное, регулирующееся по высоте сиденье.
2. Велосипед должен поддерживать 16 скоростей.
3. Велосипед не должен сильно подпрыгивать на небольших бугорках.
4. Велосипед должен позволять перевозить двух пассажиров.

8 Спецификация на разработку дырокола

Необходимо разработать дырокол, отвечающий следующим требованиям.

1. Дырокол должен делать два отверстия в листе бумаги на стандартном для папок расстоянии.
2. Дырокол должен позволять вставить не менее 20 листов бумаги.
3. Габариты дырокола не должны превышать 20 сантиметров в ширину и 10 сантиметров в высоту.
4. Вес дырокола со вставленной бумагой не должен быть более 200 грамм.
5. Дырокол не должен иметь острых или выступающих краев, которыми можно пораниться или поранить другого.
6. Дырокол должен иметь информационное табло, отображающее актуальную информацию.

9 Спецификация на разработку шкафа-купе

Необходимо разработать шкаф-купе, отвечающий следующим требованиям.

1. Шкаф имеет три вертикальные секции и три двери.
2. Двери шкафа должны крепиться на горизонтальных направляющих.
3. Высота шкафа должна составлять 2 метра.
4. Двери шкафа должны быть созданы с таким расчетом, чтобы ими ничего нельзя было прищемить.
5. Глубина шкафа должна быть выбрана таким образом, чтобы в него помещался велосипед.

ЛАБОРАТОРНАЯ РАБОТА №2 «МОДУЛЬНОЕ ТЕСТИРОВАНИЕ»

Цель: *Познакомиться с методикой компонентного тестирования, научиться составлять план тестирования класса и разрабатывать тест-драйвы.*

Постановка задачи

Выполнение лабораторной работы разбивается на две части – одна выполняется разработчиком ПО, а другая – тестировщиком.

Роль разработчика

1. Разработать класс Массив (Матрица) согласно варианту.
2. Составить спецификацию на класс Массив (Матрица).
3. Сохранить класс как проект Class Library.
4. Передать проект Class Library и спецификацию для тестирования.

Роль тестировщика

1. Получить для тестирования проект Class Library.
2. Составить план тестирования класса, описанного в требованиях и спецификации на все тестовые случаи.
3. Создать тестовые драйверы на все тестовые случаи. Предусмотреть запись результатов теста в текстовый log-файл.
4. Провести тестирование и описать выявленные дефекты.

Варианты заданий

Вариант 1

Создать класс «Массив», в котором описать следующие элементы:

- закрытое поле – массив целых чисел,
- свойство для определения длины массива,
- индексатор для доступа к элементам поля-массива,
- конструктор с одним параметром – длина массива,
- конструктор с одним параметром – массив целых чисел,
- перегруженные методы для поиска максимального элемента во всем массиве и для поиска максимального элемента в части массива, ограниченной начальным и конечным значениями индекса, передаваемых в метод в качестве параметров.

Вариант 2

Создать класс «Матрица», в котором описать следующие элементы:

- закрытое поле – матрица целых чисел,
- свойства для определения количества строк и столбцов массива,
- индексатор для доступа к элементам поля-массива,
- конструктор с двумя параметрами – количество строк, количество столбцов матрицы,
- конструктор с одним параметром – матрица целых чисел,
- перегруженные методы для вычисления произведения отрицательных элементов матрицы и вычисление произведения отрицательных элементов, в четных строках.

Вариант 3

Создать класс «Массив», в котором описать следующие элементы:

- закрытое поле – массив целых чисел,
- свойство для определения длины массива,
- индексатор для доступа к элементам поля-массива,
- конструктор с одним параметром – длина массива,
- конструктор с одним параметром – массив целых чисел,
- статический метод с переменным числом параметров для вычисления общей суммы отрицательных элементов в нескольких массивах.

Вариант 4

Создать класс «Матрица», в котором описать следующие элементы:

- закрытое поле – матрица вещественных чисел,
- свойства для определения количества строк и столбцов массива,
- индексатор для доступа к элементам поля-массива,
- конструктор с двумя параметрами – количество строк, количество столбцов матрицы,
- конструктор с одним параметром – матрица вещественных чисел,
- перегруженные методы для вычисления суммы квадратов элементов матрицы, больших заданного числа (параметр – заданное число), и вычисления суммы квадратов элементов матрицы, расположенных после определенного элемента (параметры – номер строки и номер столбца).

Вариант 5

Создать класс «Массив», в котором описать следующие элементы:

- закрытое поле – массив вещественных чисел,
- свойство для определения длины массива,
- индексатор для доступа к элементам поля-массива,
- конструктор с одним параметром – длина массива,
- конструктор с одним параметром – массив вещественных чисел,
- метод с переменным числом параметров для вычисления суммы элементов массива с заданными номерами, если номера не указаны, вычисляется сумма всех элементов.

Вариант 6

Создать класс «Квадратная матрица», в котором описать следующие элементы:

- закрытое поле – матрица вещественных чисел,
- свойство для определения размерности матрицы,
- индексатор для доступа к элементам поля-массива,
- конструктор с одним параметром – размерность матрицы,
- конструктор с одним параметром – матрица вещественных чисел,
- метод с переменным числом параметров для вычисления произведения элементов диагоналей матрицы, параллельных главной (если параметров нет, вычисляется сумма элементов всех диагоналей, параметры – номера диагоналей: 0 – главная диагональ, 1, 2 и т.д. – выше главной, -1, -2 и т.д. – ниже главной).

Вариант 7

Создать класс «Массив», в котором описать следующие элементы:

- закрытое поле – массив вещественных чисел,
- свойство для определения длины массива,
- индексатор для доступа к элементам поля-массива,
- конструктор с одним параметром – длина массива,
- конструктор с одним параметром – массив вещественных чисел,
- перегруженные методы для определения количества отрицательных элементов во всем массиве, определения количества элементов расположенных после элемента с

заданным номером, а также для определения количества отрицательных элементов, больших заданного числа.

Вариант 8

Создать класс «Матрица», в котором описать следующие элементы:

- закрытое поле – матрица целых чисел,
- свойства для определения количества строк и столбцов массива,
- индексатор для доступа к элементам поля-массива,
- конструктор с двумя параметрами – количество строк, количество столбцов матрицы,
- конструктор с одним параметром – матрица целых чисел,
- перегруженные методы для нахождения минимального элемента среди всех элементов матрицы, для нахождения минимального среди элементов в четных или нечетных строках (параметр равен 1 для нечетных строк, 2 – для четных) и для нахождения минимального среди элементов, не превышающих заданного значения (параметр – заданное число).

Вариант 9

Создать класс «Массив», в котором описать следующие элементы:

- закрытое поле – массив вещественных чисел,
- свойство для определения длины массива,
- индексатор для доступа к элементам поля-массива,
- конструктор с одним параметром – длина массива,
- конструктор с одним параметром – массив вещественных чисел,
- методы ввода и вывода массива,
- перегруженные методы для определения произведения всех элементов массива, для определения произведения элементов массива с номерами кратными заданному числу, а также для определения произведения элементов массива, расположенных до элемента с заданным номером.

Вариант 10

Создать класс «Матрица», в котором описать следующие элементы:

- закрытое поле – матрица вещественных чисел,
- свойства для определения количества строк и столбцов массива,

- индексатор для доступа к элементам поля-массива,
- конструктор с двумя параметрами – количество строк, количество столбцов матрицы,
- конструктор с одним параметром – матрица вещественных чисел,
- перегруженные методы для вычисления количества элементов матрицы, больших заданного числа (параметр – заданное число), и вычисления количества элементов матрицы, больших заданного числа и расположенных в столбцах с номерами кратными заданному целому числу.

Вариант 11

Создать класс «Массив», в котором описать следующие элементы:

- закрытое поле – массив целых чисел,
- свойство для определения длины массива,
- индексатор для доступа к элементам поля-массива,
- конструктор с одним параметром – длина массива,
- конструктор с одним параметром – массив целых чисел,
- статический метод с переменным числом параметров для вычисления общего количества положительных элементов в нескольких массивах.

Вариант 12

Создать класс «Матрица», в котором описать следующие элементы:

- закрытое поле – матрица целых чисел,
- свойства для определения количества строк и столбцов массива,
- индексатор для доступа к элементам поля-массива,
- конструктор с двумя параметрами – количество строк, количество столбцов матрицы,
- конструктор с одним параметром – матрица целых чисел,
- перегруженные методы для вычисления среднего арифметического отрицательных элементов матрицы, которые повторяются более n раз и вычисление среднего арифметического отрицательных элементов матрицы.

Вариант 13

Создать класс «Массив», в котором описать следующие элементы:

- закрытое поле – массив вещественных чисел,
- открытое поле с именем массива,
- свойство для определения длины массива,
- индексатор для доступа к элементам поля-массива,
- конструктор с одним параметром – длина массива,
- конструктор с одним параметром – массив вещественных чисел,
- метод с переменным числом параметров для вычисления произведения элементов массива с заданными номерами, если номера не указаны, вычисляется сумма всех элементов.

Вариант 14

Создать класс «Матрица», в котором описать следующие элементы:

- закрытое поле – матрица целых чисел,
- свойства для определения количества строк и столбцов матрицы,
- индексатор для доступа к элементам поля-массива,
- конструктор с двумя параметрами – количество строк, количество столбцов матрицы,
- конструктор с одним параметром – матрица целых чисел,
- перегруженные методы для вычисления суммы квадратов положительных элементов матрицы, расположенных ниже минимального среди элементов строк с номерами кратными n (параметр – n>=0), и вычисления суммы квадратов отрицательных элементов матрицы (без параметров).

Вариант 15

Создать класс «Массив», в котором описать следующие элементы:

- закрытое поле – массив вещественных чисел,
- свойство для определения длины массива,
- индексатор для доступа к элементам поля-массива,
- конструктор с одним параметром – длина массива,
- конструктор с одним параметром – массив вещественных чисел,
- перегруженные методы для определения суммы всех элементов массива, для определения суммы элементов массива с

номерами кратными заданному числу, а также для определения произведения элементов массива, расположенных после элемента с заданным номером.

Методические указания

Построение тестового драйвера

Тестирование классов обычно выполняется путем разработки *тестового драйвера*, который создает экземпляры классов и окружает эти экземпляры соответствующей средой, чтобы стал возможен прогон соответствующего тестового случая.

В обязанности тестового драйвера обычно входит удаление любого созданного им экземпляра, если в языке программирования имеет место управляемое программистом распределение памяти (например, C++).

Тестовый драйвер представляет собой элемент программы, который осуществляет прогон тестовых случаев и сбор полученных при этом результатов.

Способы реализации тестового драйвера.

1. *В виде метода тестируемого класса*, который может быть вызван для выполнения и сбора результатов прогона каждого тестового случая.

Плюсы:

- программный код драйвера находится в непосредственной близости к программному коду класса;
- упрощается многократное использование кода драйвера (в силу наследования) для тестирования подклассов.

Минусы:

- необходимость соблюдения осторожности при отделении программного кода драйвера от поставляемого программного обеспечения.

2. *В виде отдельного класса*, в обязанности которого входит выполнение и сбор результатов для каждого тестового случая.

Для тестирования нужно создать экземпляры этого класса. Прогон тестовых случаев может осуществляться вызовом специально созданных для этого методов от имени объектов класса или непосредственно при создании экземпляров.

Плюсы:

- легко осуществляется многократное использование драйвера при тестировании подклассов;
- достигается максимально компактный рабочий код;
- достигается максимальное быстродействие рабочего кода.

Минусы:

- необходимость построения нового класса;
- необходимость соблюдения осторожности при отображения изменений в классе на изменения в тестах.

3. Тестовый драйвер можно также реализовать *в виде класса-наследника тестируемого класса*.

Такому тестовому драйверу будет доступна не только *public*, но и *protected* часть класса.

Тестовое окружение также может использоваться для отчуждения отдельных модулей системы от всей системы. Разделение модулей системы на ранних этапах тестирования позволяет более точно локализовать проблемы, возникающие в их программном коде. Для поддержки работы модуля в отрыве от системы тестовое окружение должно моделировать поведение всех модулей, к функциям или данным которых обращается тестируемый модуль.

Поскольку тестовое окружение само является программой (причем зачастую реализованной не на том языке программирования, на котором написана система), оно само должно быть протестировано. Целью тестирования тестового окружения является доказательство того, что тестовое окружение никаким образом не искажает выполнение тестируемого модуля и адекватно моделирует поведение системы.

Требования, предъявляемые к тестовым классам

Тестовый драйвер должен иметь сравнительно простую структуру, чтобы не пришлось создавать для него тестовый драйвер и осуществлять прогон тестовых случаев.

Тестовый драйвер должен быть удобным в сопровождении и легко приспосабливаемым в ответ на изменения в спецификации класса, для тестирования которого он и предназначен.

Тестовый драйвер должен содержать:

- конструктор (в качестве параметра в конструктор можно передавать имя файла для записи результатов тестирования);
- функциональные элементы, позволяющие выполнять прогоны различных тестовых наборов, или даже все из них.

Для упрощения сопровождения тестов тестовые случаи могут быть сгруппированы по категориям:

- функциональный набор (строится на базе спецификации);
- структурный набор (строится на базе структурного кода);
- набор для тестирования взаимодействий (тестируют корректность функционирования последовательности событий, происходящих на объекте).

Пример выполнения

Создать класс «Массив», в котором описать следующие элементы:

- закрытое поле – массив вещественных чисел,
- свойство для определения длины массива,
- индексатор для доступа к элементам поля-массива,
- конструктор с одним параметром – длина массива,
- конструктор с одним параметром – массив вещественных чисел,
- методы ввода и вывода массива,
- перегруженные методы для определения произведения элементов массива с четными номерами, для определения произведения элементов массива в интервале значений номеров элементов массива.

Реализация роли разработчика ПО.

1. Создать в Visual Studio, выбрав язык C#, проект Class Library (Библиотека классов) (см. рисунок 2.1).

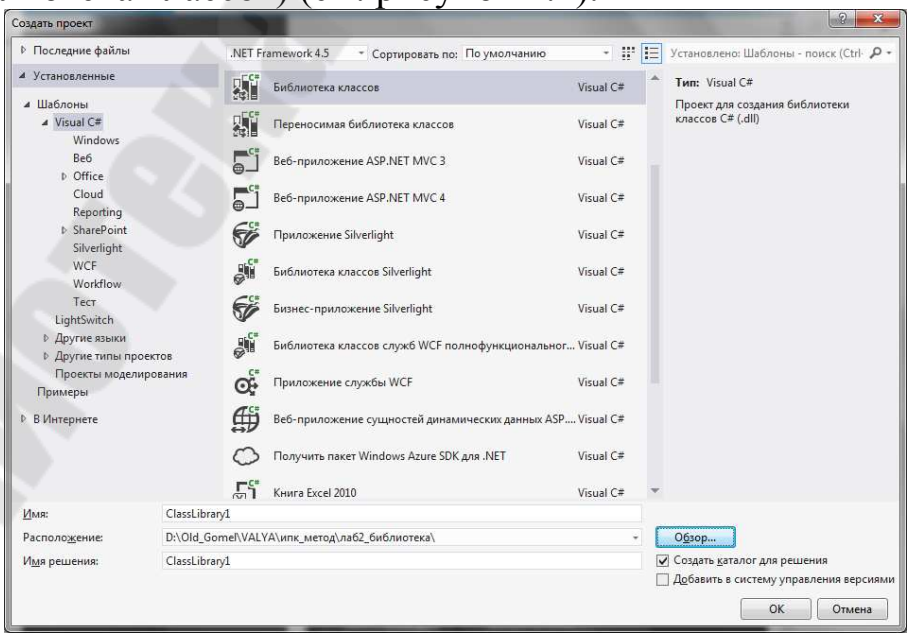


Рисунок 2.1 – Создание проекта Class Library

2. Разработать класс CMass (см. рисунок 2.2).

ClassLibrary1.CMass

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ClassLibrary1
{
    public class CMass
    {
        private int[] mass;
    // конструкторы
        public CMass(int lengh)
        {
            mass = new int[lengh];
        }
    public CMass (int [] massValue)
        {
            mass = new int[massValue.Length];
            mass = massValue;
        }
    // свойство определения длины массива
    public int getLenghMass
    {
        get { return mass.Length; }
    }
    //индексатор
    public int this[int i]
    {
        get { return mass[i]; }
        set { mass[i] = value; }
    }
    // вычисление произведения массива с четными номерами
    public int compositionMass()
    {
        int composit = 1;
        bool isComposit = false;

        for (int i = 0; i < mass.Length; i++)
        {
            if ((i +1)% 2 == 0)
            {
                composit = composit * mass[i];
                isComposit = true;
            }
        }
        if (isComposit) return composit; else return 0;
    }

    // вычисление произведения массива в интервале значений номеров массива
    public int compositionMass(int primary, int lastThing)
    {
        int composit = 1;
        bool isComposit = false;

        for (int i = primary-1; i <= lastThing-1; i++)
        {
            composit = composit * mass[i];
            isComposit = true;
        }
        if (isComposit) return composit; else return 0;
    }


    }
}
```

Рисунок 2.2 – Создание класса CMass

3. Составить спецификацию на класс «CMass».

Класс CMass

```
public class CMass
  {
     private int[] mass; //ссылка на массив целых чисел
  // конструкторы
     public CMass(int lengh)
     public CMass (int [] massValue)
// свойство определения длины массива
  public int getLenghMass
//индексатор
  public int this[int i]
// метод вычисление произведения массива с четными номерами
  public int compositionMass()
// метод вычисление произведения массива в интервале значений номеров массива
  public int compositionMass(int primary, int lastThing)
}
```

Класс реализует поток целых чисел**.**

Операции:

Конструктор **CMass**(int lengh) принимает на вход параметр целого типа int lengh и выделяет для закрытого поле int[] mass память для lengh элементов, которые инициализируются нулями.

Конструктор **CMass** (int [] massValue) принимает на вход одномерный массив целых чисел и инициличирует им закрытое поле int[] mass.

Свойство **int getLenghMass** определяет длину закрытого поля int[] mass.

Индексатор **int this[**int i**]** позволяет как получить доступ к i-му элементу закрытого поля int [] mass, так и установит значение i-му элементу.

Метод **int compositionMass()** вычисляет произведение элементов закрытого поля массив целых чисел с четными номерами. Если длина массива равна 1, то произведение равно 0.

Метод **int compositionMass(**int primary, int lastThing**)** вычисляет произведение элементов закрытого поля массив целых чисел в интервале значений номеров массива. Если значение параметра primary больше параметра lastThing, то произведение равно 0.

4. Сохранить класс как проект Class Library.
5. Передать проект Class Library и спецификацию для тестирования.

Реализация роли тестировщика

1. Получить для тестирования проект Class Library.
2. Составить план тестирования класса, описанного в требованиях и спецификации на все тестовые случаи.

Спецификации тестового случая для тестирования метода compositionMass

Название класса: **CMass**	Название тестового случая: MassTest1
Описание тестового случая: Тест проверяет правильность работы метода **compositionMass –** вычисления произведения элементов закрытого поля- массив целых чисел с четными номерами Для следующего потока целых чисел 5 6 84 2 7 9	
Ожидаемый результат: 108	

Название класса: **CMass**	Название тестового случая: MassTest2
Описание тестового случая: Тест проверяет правильность работы метода **compositionMass –** вычисления произведения элементов закрытого поля- массив целых чисел с четными номерами Для следующего потока целых чисел 5	
Ожидаемый результат: 0	

Название класса: **CMass**	Название тестового случая: massTestComposInterval1
Описание тестового случая: Тест проверяет правильность работы метода **compositionMass –** вычисления произведения элементов закрытого поля- массив целых чисел в интервале значений номеров массива. Для следующего потока целых чисел 5 6 84 2 7 9 На вход подается следующий интервал номеров элементов массива: 1 3	
Ожидаемый результат: 2520	

Название класса: **CMass**	Название тестового случая: massTestComposInterval1
Описание тестового случая: Тест проверяет правильность работы метода **compositionMass** – вычисления произведения элементов закрытого поля- массив целых чисел в интервале значений номеров массива. Для следующего потока целых чисел 5 6 84 2 7 9 На вход подается следующий интервал номеров элементов массива: 3 1	
Ожидаемый результат: 0	

3. Создать тестовые драйверы на все тестовые случаи. Предусмотреть запись результатов теста в текстовый log-файл.

Для выполнения этого пункта создадим консольное приложение (см. рисунок 2.3) и добавим новый класс CTest_Driver (см рисунок 2.4).

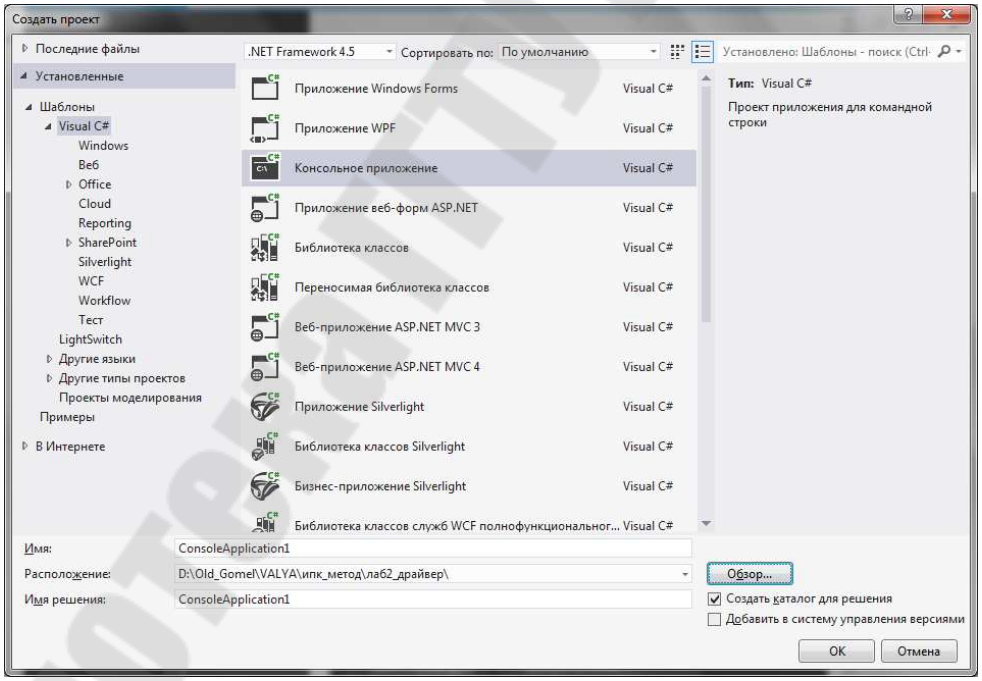


Рисунок 2.3 – Создание консольного приложения

Для того чтобы можно было записывать результаты теста в файл, необходимо подключить библиотеку:

```
using System.IO;
```

Переданный разработчиком проект Class Library добавим в консольное приложение. В Обозревателе решений на Решении

`ConsoleApplication1` нажать правой кнопкой мыши, в появившемся контекстном меню выбрать Добавить-Существующий проект, а затем указать местоположение этого проекта (см. рисунок 2.5)

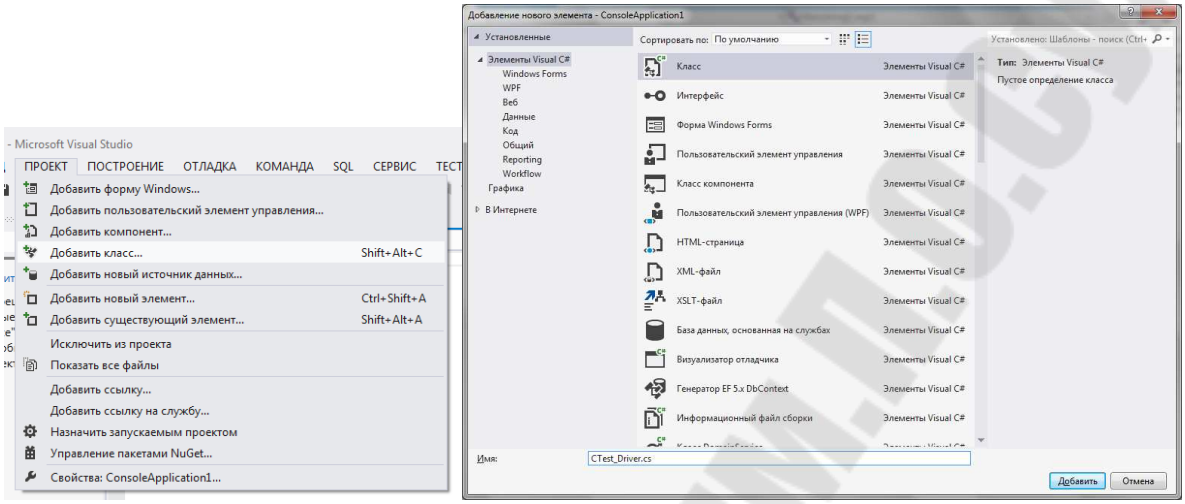


Рисунок 2.4 – Добавление класса в проект

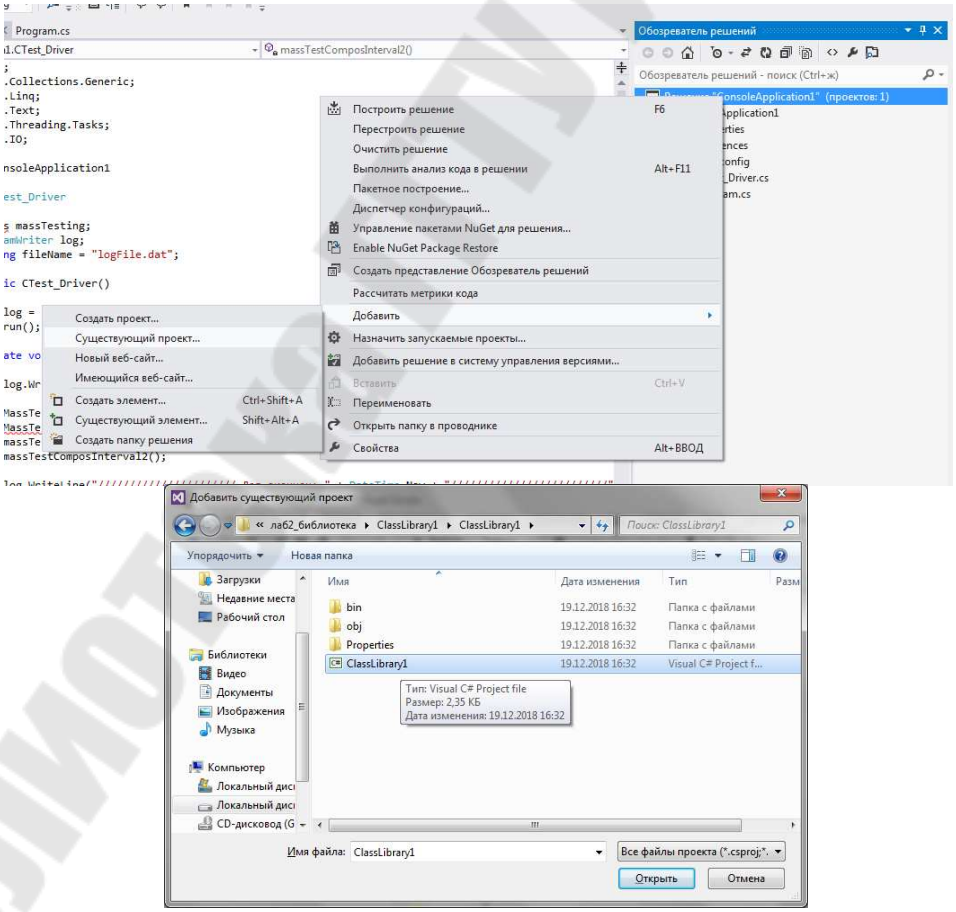


Рисунок 2.5 – Добавление существующего проекта

Следующий шаг – сделать ссылку на пространство имен, где находится класс CMass. Для этого в Обозревателе решений в Решении `ConsoleApplication1` на References (ссылки) нажать правую кнопку мыши и выбрать Добавить ссылку… (см. рисунок 2.6). Также в коде необходимо подключить с помощью директивы using следующее пространство имён:`using ClassLibrary1;`

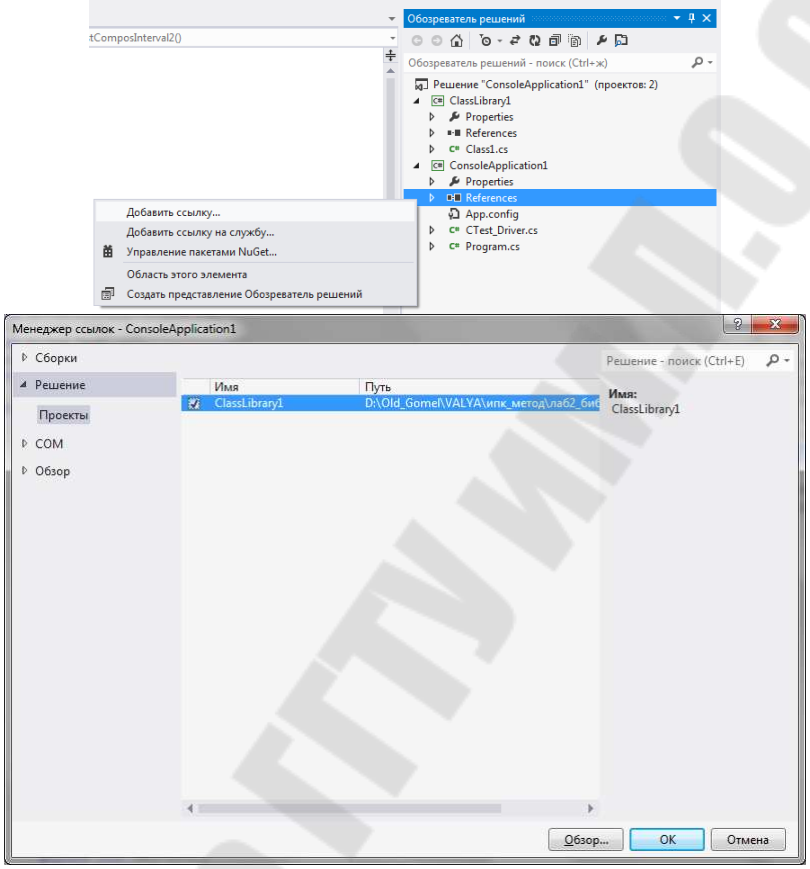


Рисунок 2.6 – Добавление ссылки

На рисунке 2.7 представлен тестовый драйвер CTest_Driver.

4. Провести тестирование и описать выявленные дефекты.
   Для этого необходимо в методе Main() создать объект класса CTest_Driver и запустить проект (см. рисунок 2.8).

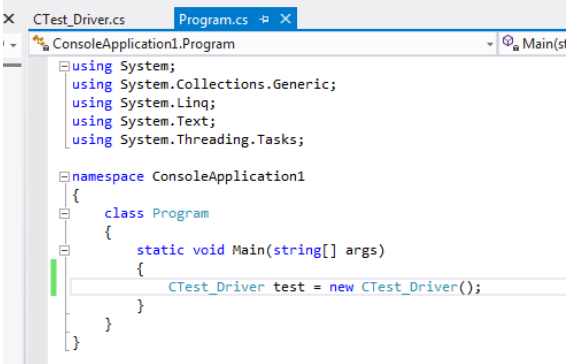


Рисунок 2.8 – Создание объекта класса CTest_Driver

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using ClassLibrary1;
namespace ConsoleApplication1
{
    class CTest_Driver
    {
        CMass massTesting;
        StreamWriter log;
        string fileName = "logFile.dat";
        public CTest_Driver()
        {
            log = new StreamWriter(fileName, true);
            run();
        }
        private void run()
        {
            log.WriteLine("////////////////////// Лог начат: " + DateTime.Now + "//////////////////////");
            MassTest1();
            MassTest2();
            massTestComposInterval1();
            massTestComposInterval2();
          log.WriteLine("////////////////////// Лог окончен: " + DateTime.Now + "//////////////////////");
            log.Close();
        }
        private void MassTest1()
        {
            int[] massTest1;
            int rezult = 0;
            massTest1 = new int[] { 5, 6, 84, 2, 7, 9 };
            massTesting = new CMass(massTest1);
            rezult = massTesting.compositionMass();
            log.WriteLine(DateTime.Now + ": Метод: " + System.Reflection.MethodInfo.GetCurrentMethod() +
                ": Результат:" + rezult);
        }
        private void MassTest2()
        {
            int[] massTest1;
            int rezult = 0;
            massTest1 = new int[] { 5, 6 };
            massTesting = new CMass(massTest1);
            rezult = massTesting.compositionMass();
            log.WriteLine(DateTime.Now + ": Метод: " + System.Reflection.MethodInfo.GetCurrentMethod() +
                ": Результат:" + rezult);
        }
        private void massTestComposInterval1()
        {
            int[] massTest1;
            int rezult = 0;
            int intervalOt = 1;
            int intervalDo = 3;
            massTest1 = new int[] { 5, 6, 84, 2, 7, 9 };
            massTesting = new CMass(massTest1);
            rezult = massTesting.compositionMass(intervalOt, intervalDo);
            log.WriteLine(DateTime.Now + ": Метод: " + System.Reflection.MethodInfo.GetCurrentMethod() +
                ": Результат:" + rezult);
        }
        private void massTestComposInterval2()
        {
            int[] massTest1;
            int rezult = 0;
            int intervalOt = 3;
            int intervalDo = 1;
            massTest1 = new int[] { 5, 6, 84, 2, 7, 9 };
            massTesting = new CMass(massTest1);
            rezult = massTesting.compositionMass(intervalOt, intervalDo);
            log.WriteLine(DateTime.Now + ": Метод: " + System.Reflection.MethodInfo.GetCurrentMethod() +
                ": Результат:" + rezult);
        }
    }
}
```

Рисунок 2.7– Тестовый драйвер

Результаты тестирования будут сохранены в log-файле (см. рисунок 2.9) по умолчанию в папке Debug\
… \ConsoleApplication1\ConsoleApplication1\bin\Debug\

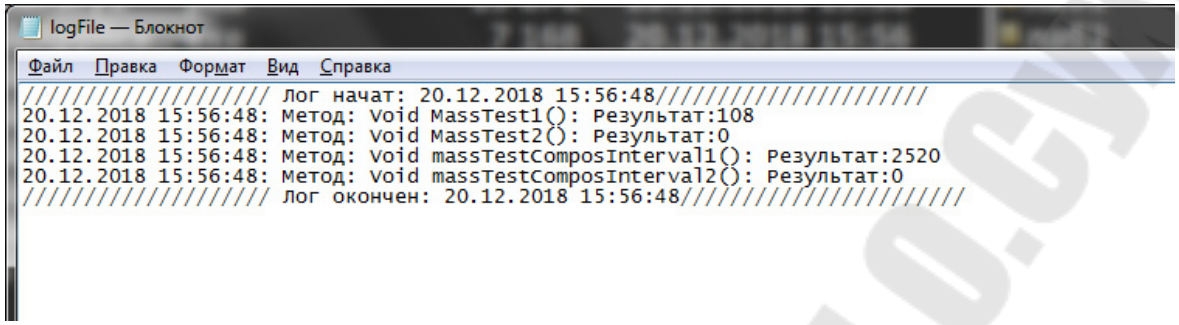


Рисунок 2.9 – Результаты тестирования

На рисунке 2.10 представлен возможный вариант тестового отчета.

Название тестового случая:
Тестировщик:
Тест пройден: Да/Нет (PASS/FAIL)
Степень важности ошибки: **Фатальная (3 уровень – crash)** **Серьезная (2 уровень – расхождение в спецификации)** **Незначительная (1 уровень – незначительная ошибка)**
Описание проблемы:
Как воспроизвести ошибку:
Предлагаемое исправление (необязательно):
Комментарий тестировщика (необязательно):

Рисунок 2.10 – Тестовый отчет

ЛАБОРАТОРНАЯ РАБОТА №3 «МОДУЛЬНЫЕ ТЕСТЫ В VISUAL STUDIO»

Цель: *Познакомиться с методикой создания модульных тестов в Visual Studio .*

Постановка задачи

Для методов класса Массив (Матрица), разработанного во второй лабораторной работе, создать модульные тесты в Visual Studio и провести анализ покрытия кода.

Варианты заданий

Варианты заданий те же, что и в лабораторной работе №2.

Методические указания

Посмотрим на определения, что же такое модульные тесты.

Модульное тестирование, или юнит-тестирование (англ. unit testing) – процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы (wikipedia.org).

Модульный тест – это автоматизированный фрагмент кода, который вызывает тестируемый метод или класс, а затем проверяет несколько предположений относительно логического поведения метода или класса (Джефф Левинсон).

Модульный тест – это код, который обеспечивает выполнение части вашего рабочего кода с ожиданием результата.

Модульный тест – это метод, который тестирует метод, класс или более крупный компонент вашего приложения изолированно от других частей, внешних систем и ресурсов (Ларри Брейдер, Алан Кэмерон Уиллс).

Как видно, даже в определениях можно найти такое вот разнообразие. Например, последнее определение четко и однозначно определяет, что модульные тесты, это изолированные тесты. Когда тестируемая сущность не взаимодействует с привычным окружением, которое тоже может быть причиной появления ошибок.

Пример выполнения

В качестве примера возьмем пример из лабораторной работы №2 и создадим для перегруженных методов класса Массив модульные тесты.

1. Создать в Visual Studio, выбрав язык C#, проект Class Library (Библиотека классов) (см. рисунок 2.1).
2. Разработать класс CMass (см. рисунок 2.2).

3. Также в коде необходимо подключить с помощь Создание проекта для модульного тестирования в Visual Studio

Чтобы выполнить unit-тестирование, необходимо в рамках того же самого решения создать ещё один проект соответствующего типа. Правой кнопкой щёлкните по решению, выберите «Добавить» и затем «Создать проект…» (см. рисунок 3.1).

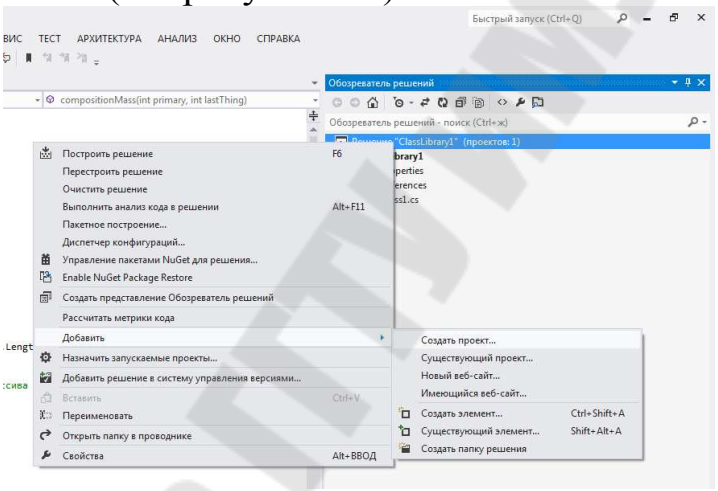


Рисунок 3.1 – Добавить новый проект

В открывшемся окне в группе Visual C# щёлкните «Тест», а затем выберите «Проект модульного теста» и нажмите «ОК». Таким образом, проект будет создан (см. рисунок 3.2).

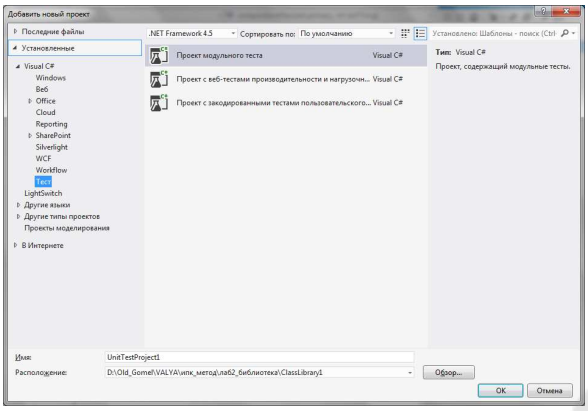


Рисунок 3.2 – Выбор проекта модульного теста

Появится следующий код (см. рисунок 3.3).

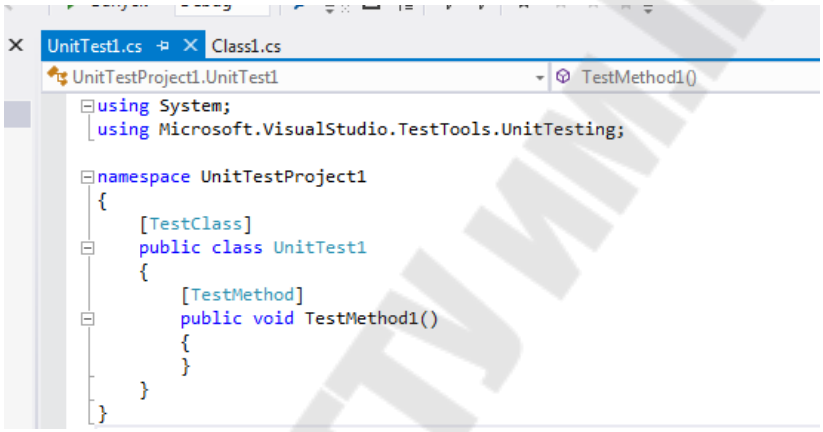


Рисунок 3.3 – Код нового проекта

Директива [TestMethod] обозначает, что далее идёт метод, содержащий модульный (unit) тест. А [TestClass] в свою очередь говорит о том, что далее идёт класс, содержащий методы, в которых присутствуют unit-тесты.

Затем в References (Ссылки) проекта необходимо добавить ссылку на проект, код которого будем тестировать. Правой кнопкой щёлкаем на References, а затем выбираем «Добавить ссылку…» . В появившемся окне раскрываем группу «Решение», выбираем «Проекты» и ставим галочку напротив проекта **ClassLibrary1**. Затем нажать «ОК» (см. рисунок 3.4).

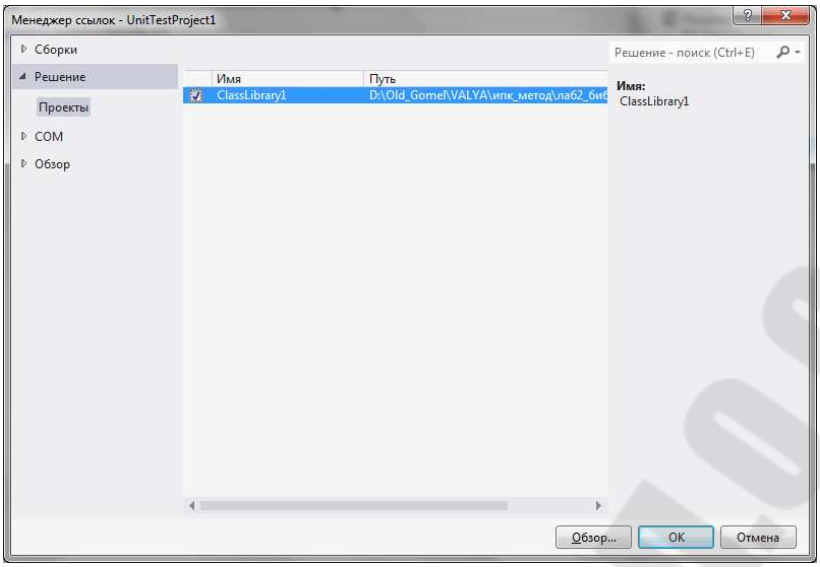


Рисунок 3.4 – Добавление ссылки

Для доступа к элементам классу CMass с помощью директивы using подключить следующее пространство имён:

```
using ClassLibrary1;
```

4. Написание теста.

Проверим правильно ли вычисляет метод произведение элементов массива с четными номерами. Ожидаемый результат (правильное решение) в данном случае это число 108.

Тестирующий метод обычно содержит три необходимых компонента:

1. исходные данные: входные значения и ожидаемый результат;
2. код, вычисляющий значение с помощью тестируемого метода;
3. код, сравнивающий ожидаемый результат с полученным.

Соответственно тестирующий код представлен на рисунке 3.5.

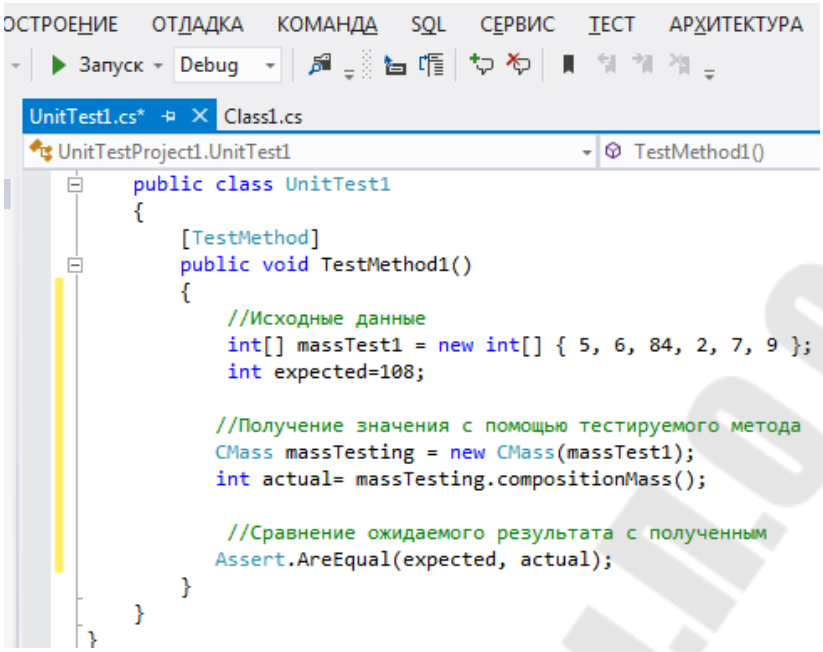


Рисунок 3.5 – Тестирующий код

Для сравнения ожидаемого результата с полученным используется метод **AreEqual** класса **Assert**. Данный класс всегда используется при написании unit тестов в Visual Studio.

5. Просмотреть все тесты.

Чтобы просмотреть все тесты, доступные для выполнения, необходимо открыть окно «Обозреватель тестов». Для этого в меню Visual Studio щёлкните на кнопку «ТЕСТ», выберите «Окна», а затем нажмите на пункт «Обозреватель тестов» (см. рисунок 3.6).

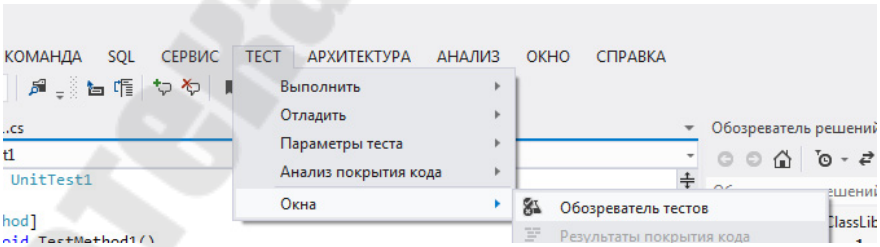


Рисунок 3.6 – Открыть окно «Обозреватель тестов»

В студии появится следующее окно (см. рисунок 3.7).

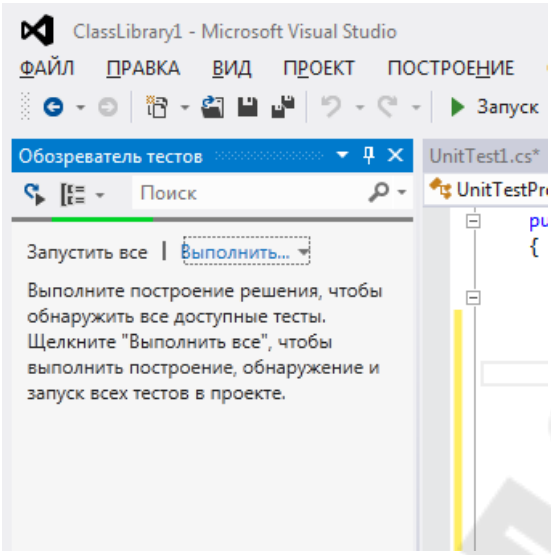


Рисунок 3.7 – Окно «Обозреватель тестов»

В данный момент список тестов пуст, поскольку решение ещё ни разу не было собрано. Выполним сборку нажатием клавиш <Ctrl><Shift><B>. После её завершения в «Обозревателе тестов» появится наш тест (см. рисунок 3.8).

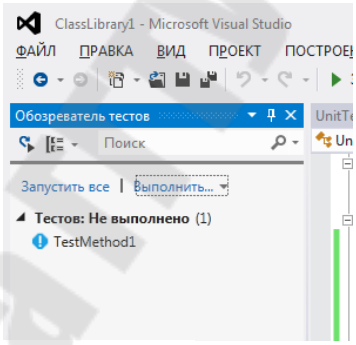


Рисунок 3.8 – Тест TestMethod1в окне «Обозреватель тестов»

Синяя табличка с восклицательным знаком означает, что указанный тест никогда не выполнялся. Выполним его. Для этого нажмём правой кнопкой мыши на его имени и выберем «Выполнить выбранные тесты» (см. рисунок 3.9).

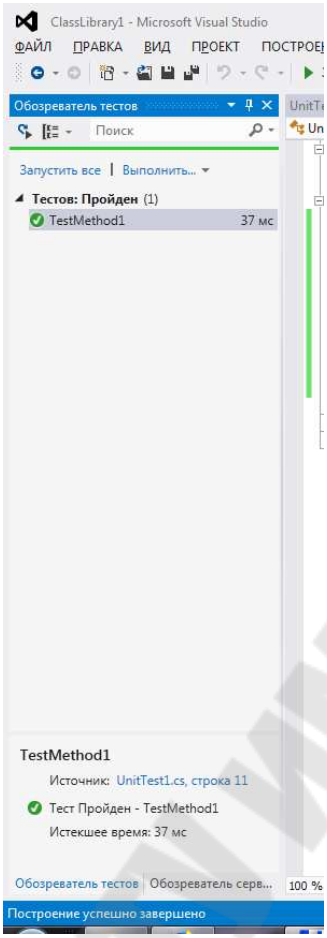


Рисунок 3.9 – Тест успешно пройден

Зелёный кружок с галочкой означает, что модульный тест успешно пройден: ожидаемый и полученный результаты равны.

Изменим код метода **compositionMass**, вычисляющего произведение элементов массива на четных местах, чтобы сымитировать провал теста и посмотреть, как поведёт себя Visual Studio. Прибавим к возвращаемому значению 10. Запустим unit-тест.

Как видно, красный круг с крестиком показывает провал модульного теста, а ниже указано, что при проверке ожидалось значение 108, а по факту оно равно 118 (см. рисунок 3.10).

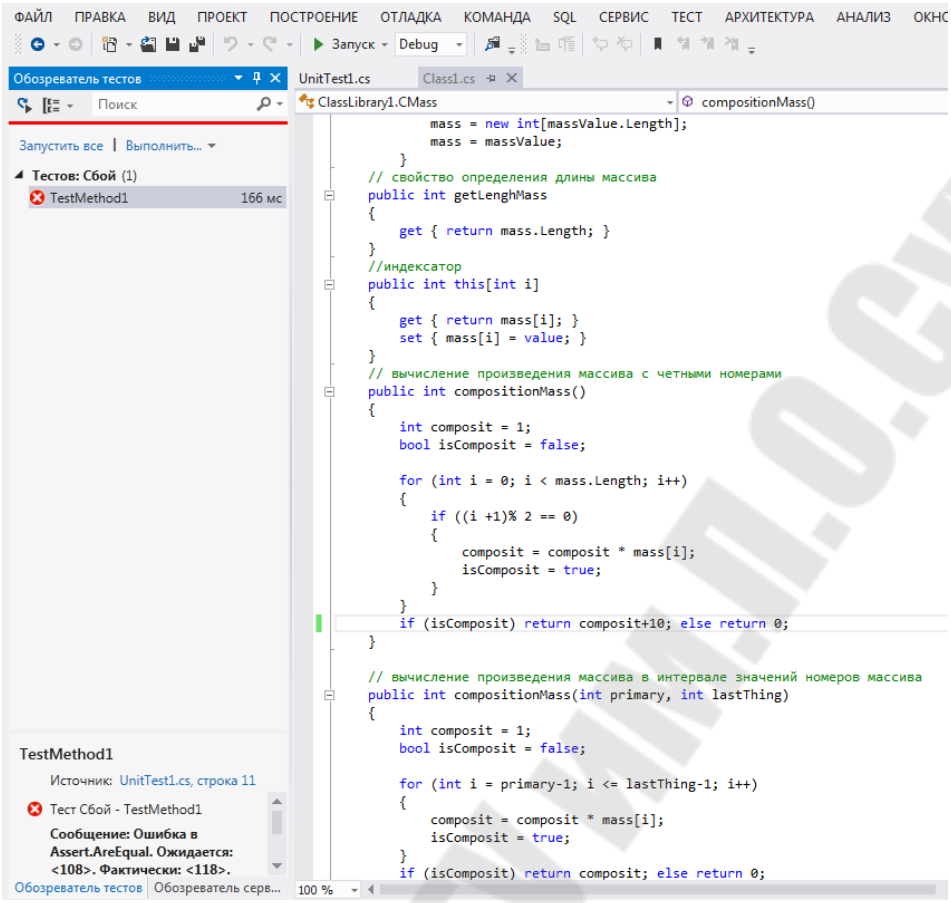


Рисунок 3.10 – Провал теста

6. Просмотр покрытия кода

Выбрать вкладка «ТЕСТ»-> Анализ покрытия кода -> Выбранные тесты/ Все тесты.

На рисунке 3.11 представлены результаты покрытия кода, где видно, что у тестируемого метода **compositionMass** не протестировано 11,11% блоков.

Результаты покрытия кода

MIA_MIA_HOME 2018-12-21 10_51_58.cover

Иерархия	Не протестировано (блоков)	Не протестировано (% блоков)	Протестировано (блоков)	Протестировано (% блоков)
MIA_MIA_HOME 2018-12-21 10_51_58.coverage	15	50,00%	15	50,00%
classlibrary1.dll	15	60,00%	10	40,00%
ClassLibrary1	15	60,00%	10	40,00%
CMass	15	60,00%	10	40,00%
CMass(int)	2	100,00%	0	0,00%
CMass(int[])	0	0,00%	2	100,00%
compositionMass()	1	11,11%	8	88,89%
compositionMass(int, int)	7	100,00%	0	0,00%
get_Item(int)	2	100,00%	0	0,00%
get_getLenghMass()	2	100,00%	0	0,00%
set_Item(int, int)	1	100,00%	0	0,00%
unittestproject1.dll	0	0,00%	5	100,00%
UnitTestProject1	0	0,00%	5	100,00%
UnitTest1	0	0,00%	5	100,00%

Рисунок 3.11 – Результаты покрытия кода

Нажав на кнопку «Цвета отображения покрытия кода» можно посмотреть, какие куски кода не покрыты тестами (см. рисунок 3.12).

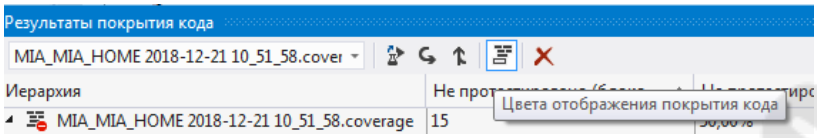


Рисунок 3.12 – Цвета отображения покрытия кода

Синим цветом, отображаются строчки, которые были пройдены тестом. Розовым цветом, отображаются строчки, не покрытые тестом (см. рисунок 3.13).

```
public int compositionMass()
{
    int composit = 1;
    bool isComposit = false;

    for (int i = 0; i < mass.Length; i++)
    {
        if ((i +1)% 2 == 0)
        {
            composit = composit * mass[i];
            isComposit = true;
        }
    }
    if (isComposit) return composit; else return 0;
}
```

Рисунок 3.13 – Покрытие кода (раскраска)

Из рисунка 3.13 видно, что надо добавить еще один модульный тест для покрытия кода

```
else return 0;
```

На рисунке 3.14 показано, что добавлен еще один тестовый метод, а на рисунке 3.15 – 100% покрытие кода двумя модульными тестами.

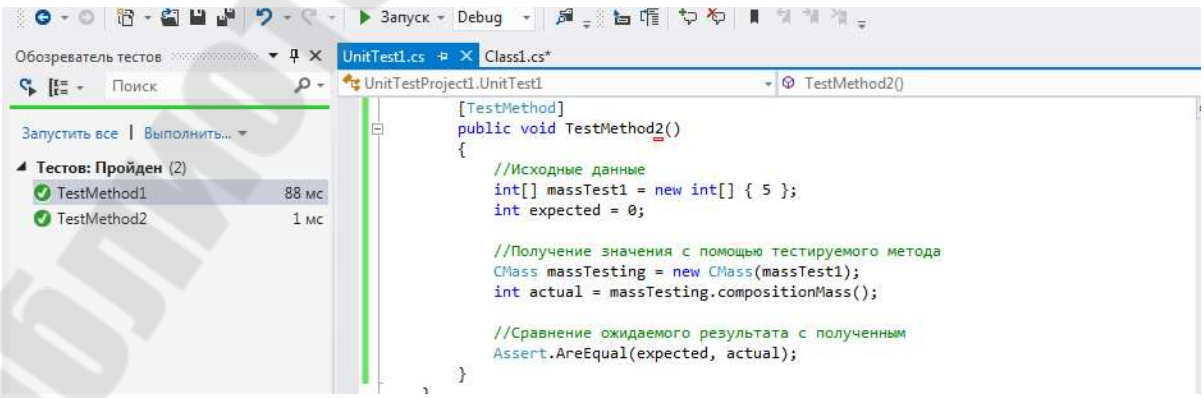
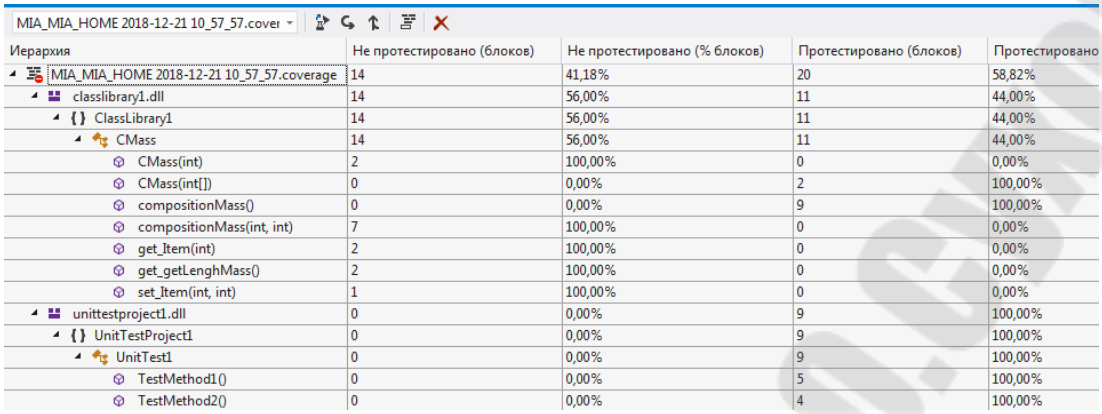


Рисунок 3.14 – TestMethod2



MIA_MIA_HOME 2018-12-21 10_57_57.cover

Иерархия	Не протестировано (блоков)	Не протестировано (% блоков)	Протестировано (блоков)	Протестировано
MIA_MIA_HOME 2018-12-21 10_57_57.coverage	14	41,18%	20	58,82%
classlibrary1.dll	14	56,00%	11	44,00%
ClassLibrary1	14	56,00%	11	44,00%
CMass	14	56,00%	11	44,00%
CMass(int)	2	100,00%	0	0,00%
CMass(int[])	0	0,00%	2	100,00%
compositionMass()	0	0,00%	9	100,00%
compositionMass(int, int)	7	100,00%	0	0,00%
get_Item(int)	2	100,00%	0	0,00%
get_getLenghMass()	2	100,00%	0	0,00%
set_Item(int, int)	1	100,00%	0	0,00%
unittestproject1.dll	0	0,00%	9	100,00%
UnitTestProject1	0	0,00%	9	100,00%
UnitTest1	0	0,00%	9	100,00%
TestMethod1()	0	0,00%	5	100,00%
TestMethod2()	0	0,00%	4	100,00%

Рисунок 3.15 – Результаты покрытия кода всеми тестами

Приведём правило, которым следует руководствоваться при написании и проведении тестов для оценки правильного функционирования программ.

Удобнее всего будет рассмотреть пример основанный на математике.

Так или иначе, тестируемый метод или функция (или вся программа в целом) имеет свою область *допустимых* входных значений. Для проверки правильности работы метода *достаточно* провести тестирование метода на входных значениях начала и конца области допустимых значений (ОДЗ), одного значения из внутренней части области, а также -1 от левой и +1 от правой границы области**.**

Например, если ОЗД функции F – это отрезок [0; 100], то для проверки корректности работы функции достаточно протестировать следующие варианты: F(0), F(50) [не обязательно 50, можно взять любое число из внутренней части ОДЗ], F(100), F(-1), F(101).

Использование классов Assert

В пространстве имен Microsoft.VisualStudio.TestTools.UnitTesting имеется несколько типов классов Assert.

– Assert;

В методе теста можно вызывать любое число методов класса Assert, таких как Assert.AreEqual(). Класс Assert содержит много методов для выбора, и многие из этих методов имеют несколько перегрузок.

– CollectionAssert;

Класс CollectionAssert служит для сравнения коллекций объектов и проверки состояния одной или нескольких коллекций.

– StringAssert;

Класс StringAssert служит для сравнения строк. Этот класс содержит различные полезные методы, такие как StringAssert.Contains, StringAssert.Matches и StringAssert.StartsWith.

– AssertFailedException;

Исключение AssertFailedException возникает в случае невыполнения теста. Причиной невыполнения теста может быть истечение времени ожидания, непредвиденное исключение или оператор Assert, создающий результат «Ошибка».

– AssertInconclusiveException;

Исключение AssertInconclusiveException возникает при каждом тесте с неопределенным результатом. Как правило, оператор Assert.Inconclusive добавляется к тесту, над которым еще ведется работа, для обозначения его неготовности к выполнению.

Альтернативным вариантом может быть обозначение теста, который еще не готов к выполнению, атрибутом Ignore. Однако недостатком в этом случае является невозможность просто создать отчет по числу тестов, которые еще необходимо реализовать.

– UnitTestAssertException;

При написании нового класса исключения Assert наследование этого класса от базового класса UnitTestAssertException упрощает определение исключения как ошибки подтверждения, а не непредвиденного исключения, выдаваемого тестом или рабочим кодом.

– ExpectedExceptionAttribute.

Если необходимо, чтобы метод теста проверял, является ли исключение, возникающее в этом методе, на самом деле требуемым исключением, включите в метод теста атрибут ExpectedExceptionAttribute.

Это утверждение, которое будет выполнено только в том случае, если модульный тест выбросит исключение типа, указанного параметром ExceptionType. Это аккуратный способ ловить исключения без необходимости возиться с блоками try...catch в юнит тесте.

Класс Assert из пространства имен Microsoft.VisualStudio.TestTools.UnitTesting с помощью своих статических методов позволяет верифицировать результат

выполнения некоторого действия. При тестировании нам доступен еще ряд методов:

- AreEqual(object expected, object actual): проверяет, равны ли оба объекта. Имеет различные перегруженные версии, позволяющие сравнивать различные типы объектов;
- AreEqual<T>(T expected, T actual): обобщенная версия предыдущего метода. Например, Assert.AreEqual<string>("Index", result.MasterName);
- AreNotEqual(object expected, object actual): проверяет, не равны ли оба объекта. Тест проходит успешно, если объекты не равны;
- AreNotEqual<T>(T expected, T actual): обобщенная версия предыдущего метода;
- AreSame(object expected, object actual): проверяет, указывают ли оба объекта на один и тот же объект в памяти;
- AreNotSame(object expected, object actual): проверяет, указывают ли оба объекта на разные объекты в памяти. Если они указывают на один и тот же объект, то тест заканчивается неудачно;
- Equals(object objA, object objB): проверяет на равенство оба объекта;
- IsFalse(bool condition): проверяет, равно ли условие condition значению false;
- IsTrue(bool condition): проверяет, равно ли условие condition значению true;
- IsNull(object value): проверяет, имеет ли объект value значение null;
- IsNotNull(object), IsNotNull(object, string): Утверждает, что переменной присвоена ссылка на объект;
- Fail(), Fail(string): Утверждение не выполнилось: условия не проверены;
- IsInstanceOfType(object value, Type expectedType): проверяет, представляет ли объект value тип expectedType;
- IsNotInstanceOfType(object, Type), IsNotInstanceOfType(object, Type, string): Утверждает, что этот объект не является объектом указанного типа.

ЛАБОРАТОРНАЯ РАБОТА №4 «КОНСОЛЬНОЕ C# ПРИЛОЖЕНИЕ С NUNIT ТЕСТИРОВАНИЕМ»

Цель: *Познакомиться с методикой создания консольного приложения на языке C# с Nunit тестированием.*

Постановка задачи

1. Создать консольное приложение C#.
2. В классе Programm создать статический метод, согласно указанному варианту.
3. Добавить библиотеку NUnit, используя NuGet
4. Написать приложение с NUnit тестирование
5. Провести NUnit тестирование

Варианты заданий

Варианты заданий представлены в таблице 4.1.

Таблица 4.1 – Варианты заданий

Вариант	Задание
1	Найти сумму элементов массива, стоящих на местах кратных 3 и больших, чем среднее арифметическое всех элементов массива
2	Найти сумму положительных элементов массива, стоящих за максимальным элементом массива
3	Найти количество нулей среди элементов массива, расположенных между минимальным и максимальным элементами массива
4	Найти максимальный элемент среди элементов, расположенных за минимальным элементом массива
5	Найти значение второго положительного элемента массива среди элементов, расположенных за минимальным элементом массива
6	Найти произведение элементов массива, находящихся между первым отрицательным и максимальным элементами

Вариант	Задание
	массива, не включая их
7	Найти произведение отрицательных чисел, которые находятся между вторым и третьим положительными элементами массива, не включая их
8	Найти минимальное значение из чисел, встречающихся в массиве только один раз
9	Найти количество нулей среди первых 3 и последних 2 элементов массива
10	Найти максимальное значение среди отрицательных элементов массива
11	Найти максимальное значение среди элементов массива, расположенных до второго отрицательного числа
12	Найти сумму ненулевых чисел, стоящих на четных местах
13	Найти сумму квадратов отрицательных чисел, стоящих на местах кратных 3
14	Найти среднее арифметическое чисел, непопадающих в промежуток [*A*, *B*]
15	Найти среднее арифметическое положительных чисел, стоящих на нечетных местах

Методические указания

NUnit – это фреймворк для unit-тестирования приложений на **C#**. NUnit позволяет создавать автоматические тесты. Писать тесты, с использованием NUnit, можно непосредственно в среде разработки, например, MS Visual Studio.

Пример выполнения

Постановка задачи.

1. Создать консольное приложение C#.
2. В классе Programm создать статический метод, вычисления количества положительных целых чисел.
3. Добавить библиотеку NUnit, используя NuGet
4. Написать приложение с NUnit тестированием.

5. Провести NUnit тестирование

Порядок выполнения.

1. Создание консольного приложения в C# (см. рисунок 4.1).

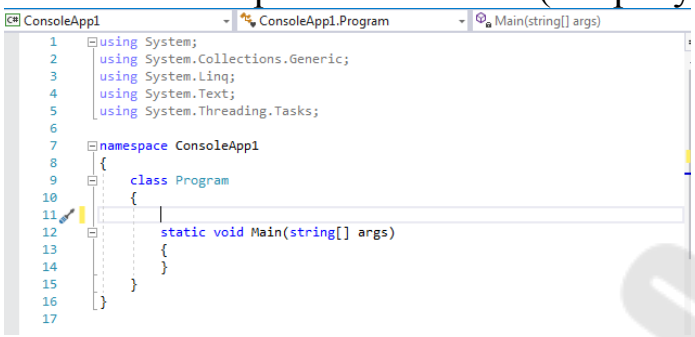


Рисунок 4.1 – Создание консольного приложения

2. Создание статического метода вычисления положительных целых чисел (см. рисунок 4.2).

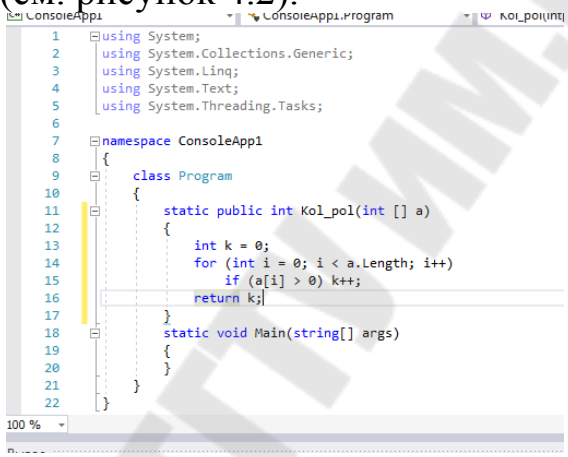


Рисунок 4.2 – Создание статического метода

3. Добавить библиотеку NUnit, используя NuGet. В Обозревателе решений нажать правую кнопку мыши и выбрать из контекстного меню Управление пакетами NuGet для решения (см. рисунок 4.3). Пишем **nunit** и нажимаем **Install** (см. рисунок 4.4). Добавить **NUnit Test Adapter** (см. рисунок 4.5).

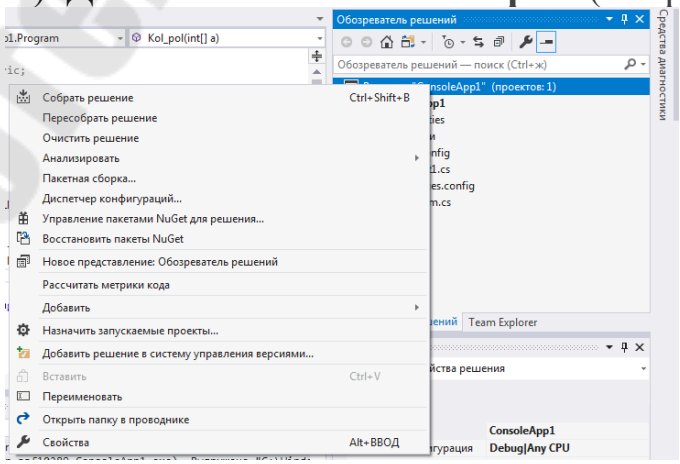


Рисунок 4.3 – Выбор Управление пакетами NuGet

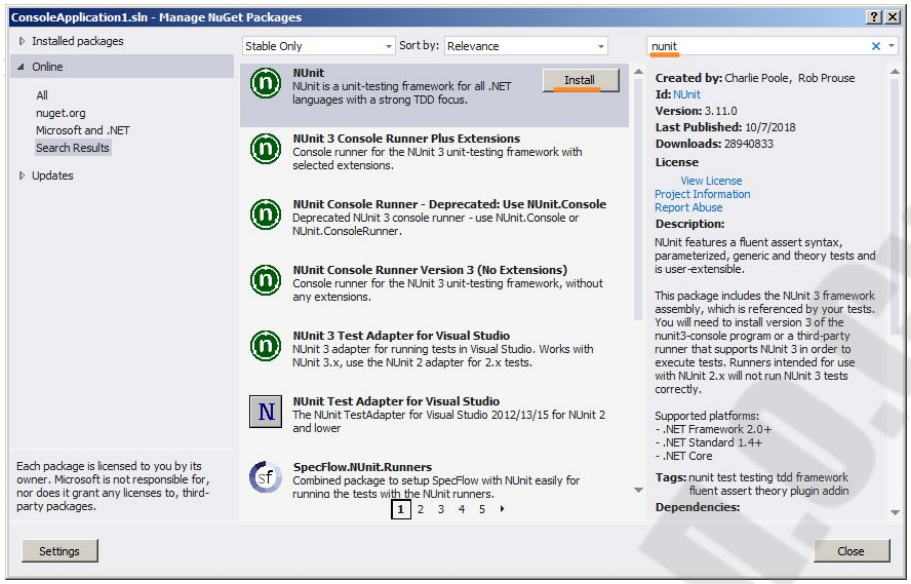


Рисунок 4.4 – Установка библиотеки Nunit

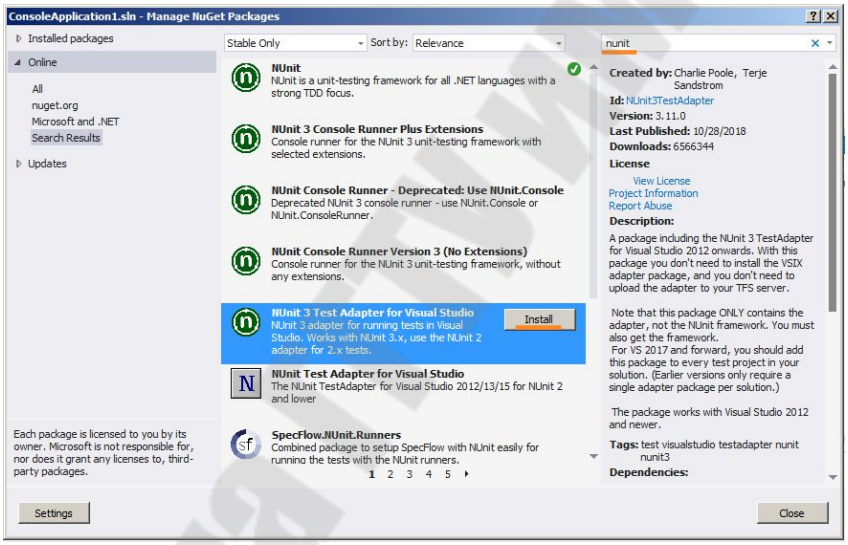


Рисунок 4.5 – Установка NUnit 3 Test Adapter

4. Пишем приложение с NUnit тестированием.

Добавить класс MyTest1 (выбрать Проект/Добавить класс) с атрибутом [TestFixture], который означает, что класс для автоматического тестирования.

Добавить ссылку на пространство имен:

```
using NUnit.Framework;
```

Пишем метод MyCol_pol с атрибутом[Test], который означает, что метод для автоматического тестирования (см. рисунок 4.6).

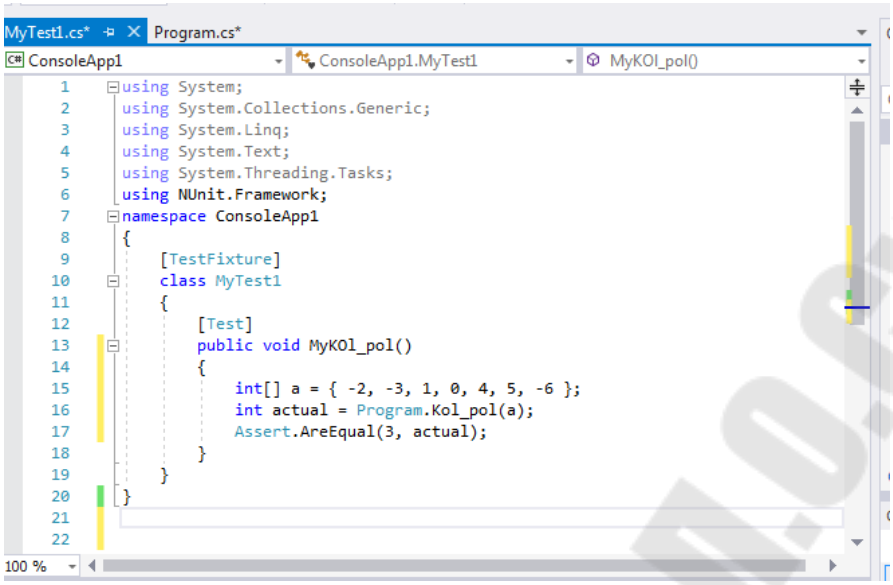


Рисунок 4.6 – Класс MyTest1

5. Смотрим результаты тестирования. Выбрать вкладку Тест/Выполнить/Все тесты (см. рисунок 4.7). Из рисунка 4.8 видно, что тест прошел.

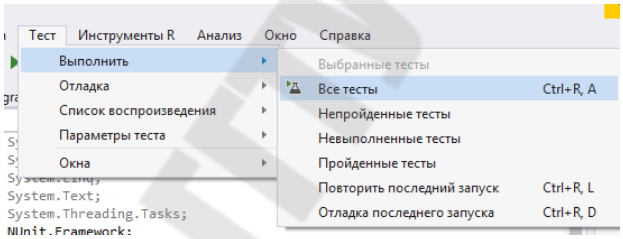


Рисунок 4.7 – Выполнить все тесты

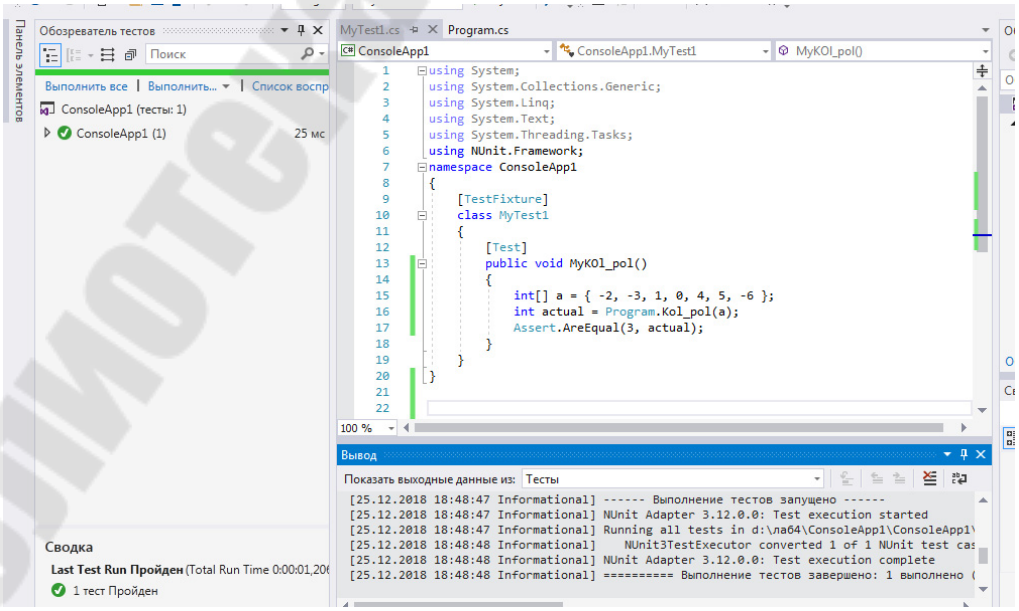


Рисунок 4.8 – Тест прошел

Теперь, когда тест проходит успешно, пора создать дополнительные тесты. Можно добавить новые тесты с помощью атрибута [Test], но это скоро станет утомительным. Есть другие атрибуты NUnit, которые позволяют создавать наборы похожих тестов. Атрибут [TestCase] используется для создания набора тестов, которые выполняют один и тот же код, но имеют разные входные аргументы. С помощью атрибута [TestCase] можно указать значения для этих входных аргументов.

В нашем случае входными аргументами будут ожидаемый результат – переменная int exep и массив целых чисел int [] a переменной длины. Итак, вместо трех тестовых методов, напишем один, но с тремя разными наборами входных аргументов (см. рисунок 4.9).

```
}
[TestCase(0,-1, -2, -3)]
[TestCase(0, 0, 0)]
[TestCase(2, 1, -2, 3,-7)]
public void MyKOl_pol1(int exep, params int [] a)
{

    int actual = Program.Kol_pol(a);
    Assert.AreEqual(exep, actual);
}
```

Рисунок 4.9 – Атрибут [TestCase] с разными наборами входных аргументов

Запустим тесты на выполнение, все четыре прошли (см. рисунок 4.10).

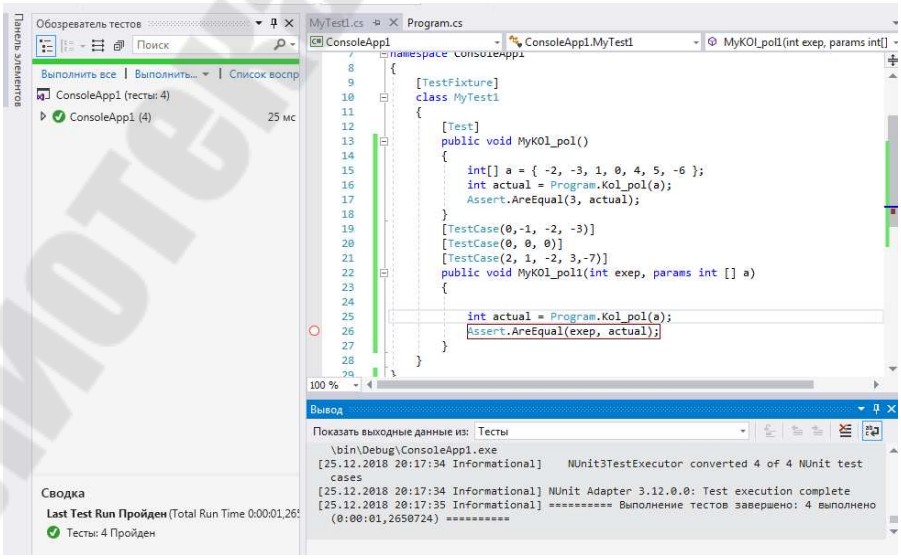


Рисунок 4.10 – Все тесты пройдены

ЛАБОРАТОРНАЯ РАБОТА №5 «РАЗРАБОТКА ЧЕРЕЗ ТЕСТИРОВАНИЕ»

Цель: *Познакомиться с методикой разработки приложения через тестирование (TDD), научиться составлять план тестирования подсистемы модулей и разрабатывать тестовый проект средствами с использованием Microsoft Visual Studio Team Edition (MVSTE) for Software Testers до создания тестируемого класса.*

Постановка задачи

1. Для усвоения методики разработки приложения через тестирования выполнить учебный пример – разработать метод решения квадратного уравнения через тестирование.
2. Все элементы класса «Одномерный массив» или «Матрица» разработать через тестирование, используя модульные тесты в Visual Studio на языке C#. Сохранить класс как проект Class Library.

Варианты заданий

Вариант 1

Для решения задачи создать класс «Одномерный массив», в котором описать следующие элементы:

- закрытое поле – массив целых чисел;
- свойство для определения длины массива;
- индексатор для доступа к элементам поля-массива;
- конструктор с одним параметром – длина массива;
- конструктор с одним параметром – массив целых чисел;
- метод для вычисления суммы положительных элементов, стоящих после второго максимума.

Вариант 2

Для решения задачи создать класс «Матрица», в котором описать следующие элементы:

- закрытое поле – матрица целых чисел;
- свойства для определения количества строк и столбцов матрицы;
- индексатор для доступа к элементам поля-массива;

– конструктор с двумя параметрами – количество строк, количество столбцов матрицы;
– конструктор с одним параметром – матрица целых чисел;
– метод для поиска количества нулей в матрице между минимумом и максимумом.

Вариант 3

Для решения задачи создать класс «Одномерный массив», в котором описать следующие элементы:

– закрытое поле – массив целых чисел;
– свойство для определения длины массива;
– индексатор для доступа к элементам поля-массива;
– конструктор с одним параметром – длина массива;
– конструктор с одним параметром – массив целых чисел;
– метод для вычисления суммы положительных элементов стоящих после 2-го нуля.

Вариант 4

Для решения задачи создать класс «Матрица», в котором описать следующие элементы:

– закрытое поле – матрица целых чисел;
– свойства для определения количества строк и столбцов матрицы;
– индексатор для доступа к элементам поля-массива;
– конструктор с двумя параметрами – количество строк, количество столбцов матрицы;
– конструктор с одним параметром – матрица целых чисел;
– метод для вычисления количества тех чисел, которые равны сумме элементов, стоящей между максимумом и минимумом.

Вариант 5

Для решения задачи создать класс «Одномерный массив», в котором описать следующие элементы:

– закрытое поле – массив целых чисел;
– свойство для определения длины массива;
– индексатор для доступа к элементам поля-массива;
– конструктор с одним параметром – длина массива;
– конструктор с одним параметром – массив целых чисел;

– метод для вычисления суммы модулей отрицательных элементов массива, расположенных после первого элемента, равного нулю.

Вариант 6

Для решения задачи создать класс «Матрица», в котором описать следующие элементы:

– закрытое поле – матрица целых чисел;
– свойства для определения количества строк и столбцов матрицы;
– индексатор для доступа к элементам поля-массива;
– конструктор с двумя параметрами – количество строк, количество столбцов матрицы;
– конструктор с одним параметром – матрица целых чисел;
– метод вычисления суммы положительных чисел между первым и вторым 0.

Вариант 7

Для решения задачи создать класс «Одномерный массив», в котором описать следующие элементы:

– закрытое поле – массив целых чисел; свойство для определения длины массива;
– индексатор для доступа к элементам поля-массива;
– конструктор с одним параметром – длина массива;
– конструктор с одним параметром – массив целых чисел;
– метод вычисления произведения отрицательных элементов, стоящих после второго минимума.

Вариант 8

Для решения задачи создать класс «Матрица», в котором описать следующие элементы:

– закрытое поле – матрица целых чисел;
– свойства для определения количества строк и столбцов матрицы;
– индексатор для доступа к элементам поля-массива;
– конструктор с двумя параметрами – количество строк, количество столбцов матрицы;
– конструктор с одним параметром – матрица целых чисел;

– метод вычисления суммы чисел между вторым положительным и первым отрицательным элементами.

Вариант 9

Для решения задачи создать класс «Одномерный массив», в котором описать следующие элементы:

– закрытое поле – массив целых чисел;
– свойство для определения длины массива;
– индексатор для доступа к элементам поля-массива;
– конструктор с одним параметром – длина массива;
– конструктор с одним параметром – массив целых чисел;
– метод для вычисления произведения отрицательных элементов, стоящих после второго максимума.

Вариант 10

Для решения задачи создать класс «Матрица», в котором описать следующие элементы:

– закрытое поле – матрица целых чисел;
– свойства для определения количества строк и столбцов матрицы;
– индексатор для доступа к элементам поля-массива;
– конструктор с двумя параметрами – количество строк, количество столбцов матрицы;
– конструктор с одним параметром – матрица целых чисел;
– метод для поиска количества отрицательных элементов стоящих после 1го нуля.

Вариант 11

Для решения задачи создать класс «Одномерный массив», в котором описать следующие элементы:

– закрытое поле – массив целых чисел;
– свойство для определения длины массива;
– индексатор для доступа к элементам поля-массива;
– конструктор с одним параметром – длина массива;
– конструктор с одним параметром – массив целых чисел;

– метод для вычисления суммы положительных элементов после последнего отрицательного.

Вариант 12

Для решения задачи создать класс «Матрица», в котором описать следующие элементы:

– закрытое поле – матрица целых чисел;
– свойства для определения количества строк и столбцов матрицы;
– индексатор для доступа к элементам поля-массива;
– конструктор с двумя параметрами – количество строк, количество столбцов матрицы;
– конструктор с одним параметром – матрица целых чисел;
– метод для вычисления количества чисел из данного интервала между первым и последним положительным.

Вариант 13

Для решения задачи создать класс «Одномерный массив», в котором описать следующие элементы:

– закрытое поле – массив целых чисел;
– свойство для определения длины массива;
– индексатор для доступа к элементам поля-массива;
– конструктор с одним параметром – длина массива;
– конструктор с одним параметром – массив целых чисел;
– метод для вычисления суммы отрицательных элементов на четных местах после второго 0.

Вариант 14

Для решения задачи создать класс «Матрица», в котором описать следующие элементы:

– закрытое поле – матрица целых чисел;
– свойства для определения количества строк и столбцов матрицы;
– индексатор для доступа к элементам поля-массива;
– конструктор с двумя параметрами – количество строк, количество столбцов матрицы;
– конструктор с одним параметром – матрица целых чисел;

– метод для вычисления суммы отрицательных чисел в нечетных столбцах после первого положительного.

Вариант 15

Для решения задачи создать класс «Одномерный массив», в котором описать следующие элементы:

– закрытое поле – массив целых чисел;
– свойство для определения длины массива;
– индексатор для доступа к элементам поля-массива;
– конструктор с одним параметром – длина массива;
– конструктор с одним параметром – массив целых чисел;
– метод вычисления произведения положительных элементов после последнего отрицательного.

Методические указания

Разработка через тестирование (англ. test-driven development, TDD) – техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода к соответствующим стандартам (wikipedia.org).

TDD (сокр. от англ. test-driven development – «разработка через тестирование») – это специальная методика разработки ПО, которая основывается на коротких циклах работы, где сначала создаётся тест, а потом функционал.

Все определения, сводятся к цикличности и первичности тестов. Визуально это можно представить вот так рисунок 5.1.

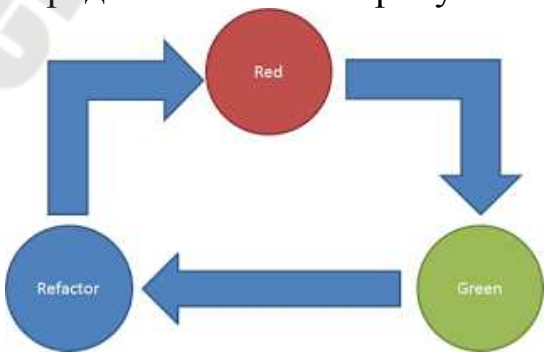


Рисунок 5.1 – Цикличность и первичность тестов

Красная зона, это зона написания нового теста, который проверяет функционал отсутствующий в приложении (тест не проходит, он красный). Затем идет зеленая зона, в рамках которой необходимо написать минимальное количества кода, чтобы тест стал «зеленым». Ну и если получился не очень красивый код, то проводим его рефакторинг. Так как весь ранее написанный функционал покрыт тестами, то при написании нового и при рефакторинге, если что-то и сломается, то сразу об этом узнаем.

В TDD применяются все те же принципы, что были описаны в модульном тестировании: *написание тестов для небольших участков кода, тестирование в изоляции, автоматизация тестов.*

Пример выполнения

В качестве примера, напишем метод решения квадратного уравнения через тестирование.

Порядок выполнения

1. Создать в Visual Studio, выбрав язык C#, проект Class Library (Библиотека классов) (см. рисунок 2.1).
2. Класс Class1 переименовать в класс SqrtRoots.
3. В классе SqrtRoots определить статический метод Calc с результатом возврата null.

```
public class SqrtRoots
    {
public static double[] Calc(double a, double b,
double c)
{
    return null;
}
}
```

4. Добавить проект для тестирования.

Чтобы выполнить unit-тестирование, необходимо в рамках того же самого решения создать ещё один проект соответствующего типа. Правой кнопкой щёлкните по решению, выберите «Добавить» и затем «Создать проект…» (см. рисунок 5.2).

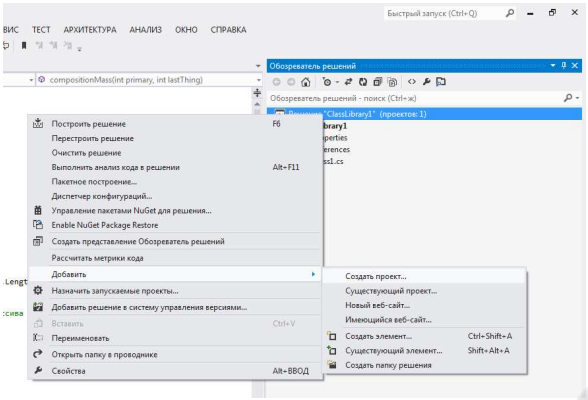


Рисунок 5.2 – Добавить новый проект

В открывшемся окне в группе Visual C# щёлкните «Тест», а затем выберите «Проект модульного теста» и нажмите «ОК». Таким образом, проект будет создан (см. рисунок 5.3).

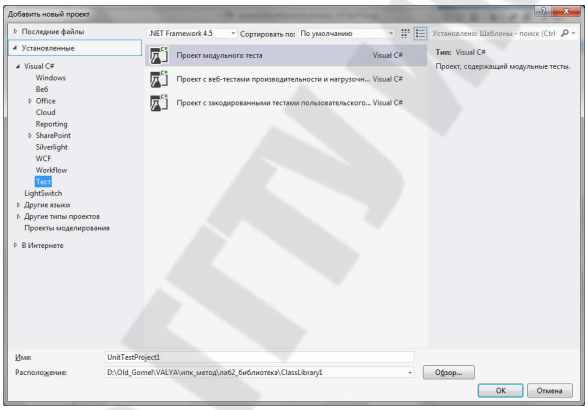


Рисунок 5.3 – Выбор проекта модульного теста

Появится следующий код (см. рисунок 5.4).

```
UnitTest1.cs | Class1.cs
UnitTestProject1.UnitTest1 | TestMethod1()
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace UnitTestProject1
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}
```

Рисунок 5.4 – Код нового проекта

Затем в References (Ссылки) проекта необходимо добавить ссылку на проект, код которого будем тестировать. Правой кнопкой щёлкаем на References, а затем выбираем «Добавить ссылку…» . В появившемся окне раскрываем группу «Решение», выбираем «Проекты» и ставим галочку напротив проекта **ClassLibrary1**. Затем нажать «ОК» (см. рисунок 5.5).

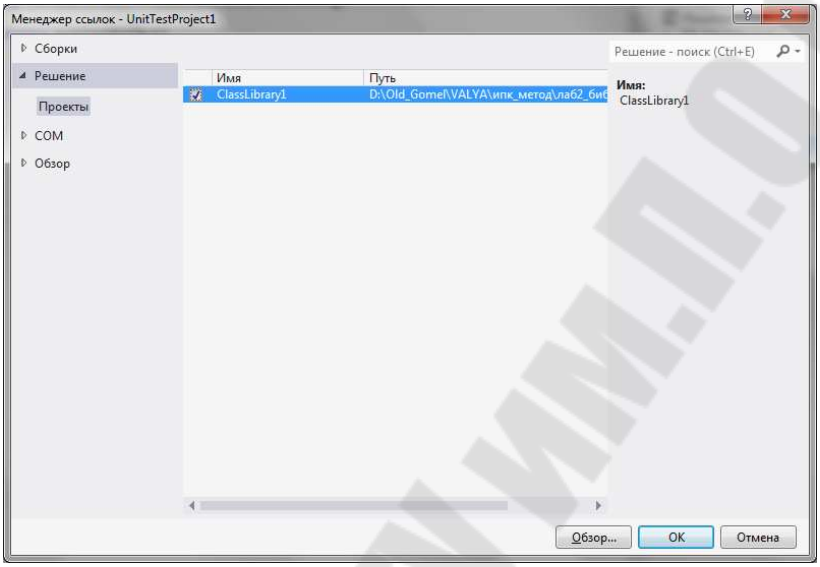


Рисунок 5.5 – Добавление ссылки

Подключить с помощью директивы using следующее пространство имён:

```
using ClassLibrary1;
```

5. Пишем первый тестовый случай –нет корней:

```
[TestMethod]
public void NotRootsTest()
{
    double a = 1;
    double b = 4;
    double c = 5;
    double[] actual = SqrtRoots.Calc(a, b, c);
    Assert.IsNotNull(actual);
    Assert.IsTrue(actual.GetLength(0) == 0);
}
```

Так как только что написали этот тест, то построив решение, должны оказаться в красной зоне, с не прошедшим тестом (см. рисунок 5.6).

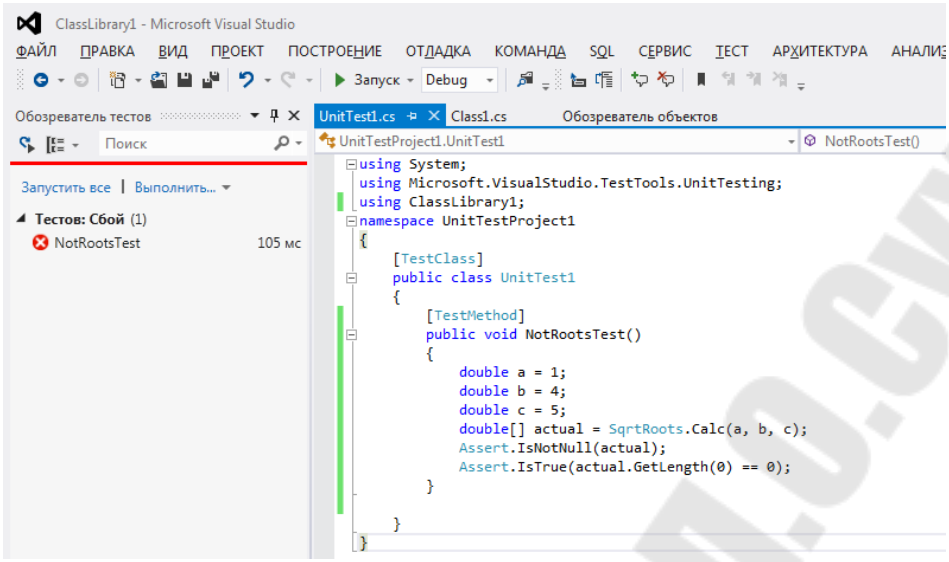


Рисунок 5.6 – Провал первого теста

Правим код:

```
public static double[] Calc(double a, double b,
double c)
{
    return new double[0];
}
```

Обратите внимание, что написано минимально необходимое количество кода, чтобы тест прошел (см. рисунок 5.7).

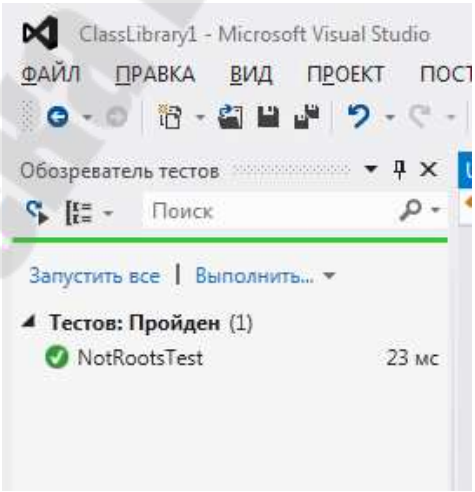


Рисунок 5.7 –Первый тест прошел

Подвергать рефакторингу здесь пока нечего, поэтому переходим в следующую итерацию.

6. Пишем второй тестовый случай – один корень:

```
[TestMethod]
public void OneRootTest()
{
    double a = 1;
    double b = -4;
    double c = 4;
    double[] expected = { 2 };
    double[] actual = SqrtRoots.Calc(a, b, c);
    CollectionAssert.AreEqual(expected, actual);
}
```

Опять попали в красную зону, написав новый тест (см. рисунок 5.8).

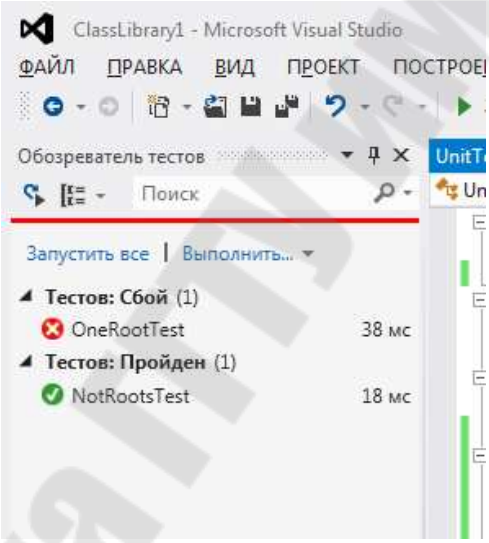


Рисунок 5.8 – Провал второго теста

Вносим изменения в код. Сразу, правильно обрабатываем такую ситуацию.

```
public static double[] Calc(double a, double b,
double c)
{
    double d = Math.Pow(b, 2) - 4 * a * c;
    if (d < 0)
    {
        return new double[0];
    }
    else
```

```
        {
            return new double[] { -b / (2 * a) };
        }
    }
```

Запускаем: выполнить все тесты. Оба теста прошли. Так как в таком, достаточно простом методе две точки выхода, проведем рефакторинг и перепишем его:

```
public static double[] Calc(double a, double b,
double c)
{
    double[] result = new double[0];
    double d = Math.Pow(b, 2) - 4 * a * c;
    if (d >= 0)
    {
        result = new double[] { -b / (2 * a) };
    }
    return result;
}
```

Пересобираем проект, удостоверяемся, что оба теста проходят.

7. Пишем третий тестовый случай – два корня:

```
 [TestMethod]
public void DoubleRootsTest()
{
    double a = 1;
    double b = -5;
    double c = 4;
    double[] expected = { 4, 1 };
    double[] actual = SqrtRoots.Calc(a, b, c);
    CollectionAssert.AreEqual(expected, actual);
}
```

И опять, в красной зоне (см рисунок 5.9).

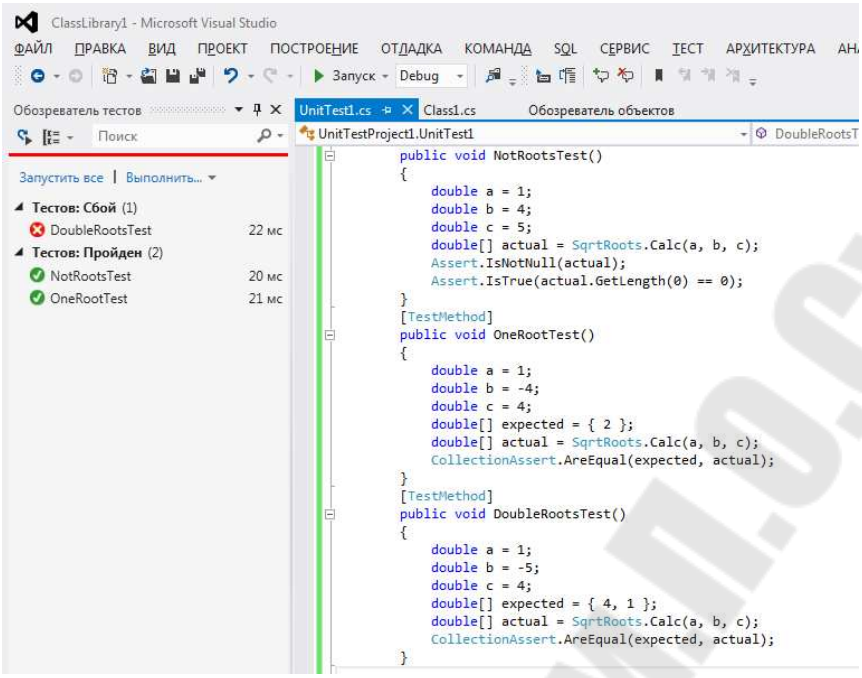


Рисунок 5.9 – Провал третьего теста

Правим код:

```
public static double[] Calc(double a, double b,
double c)
{
    double[] result = new double[0];
    double d = Math.Pow(b, 2) - 4 * a * c;
    if (d == 0)
    {
        result = new double[] { -b / (2 * a) };
    }
    else
    {
        result = new double[] { (-b + Math.Sqrt(d))
/ (2 * a), (-b - Math.Sqrt(d)) / (2 * a) };
    }
    return result;
}
```

После запуска тестов: новый тест проходит, но не проходит первый тест (см. рисунок 5.10).

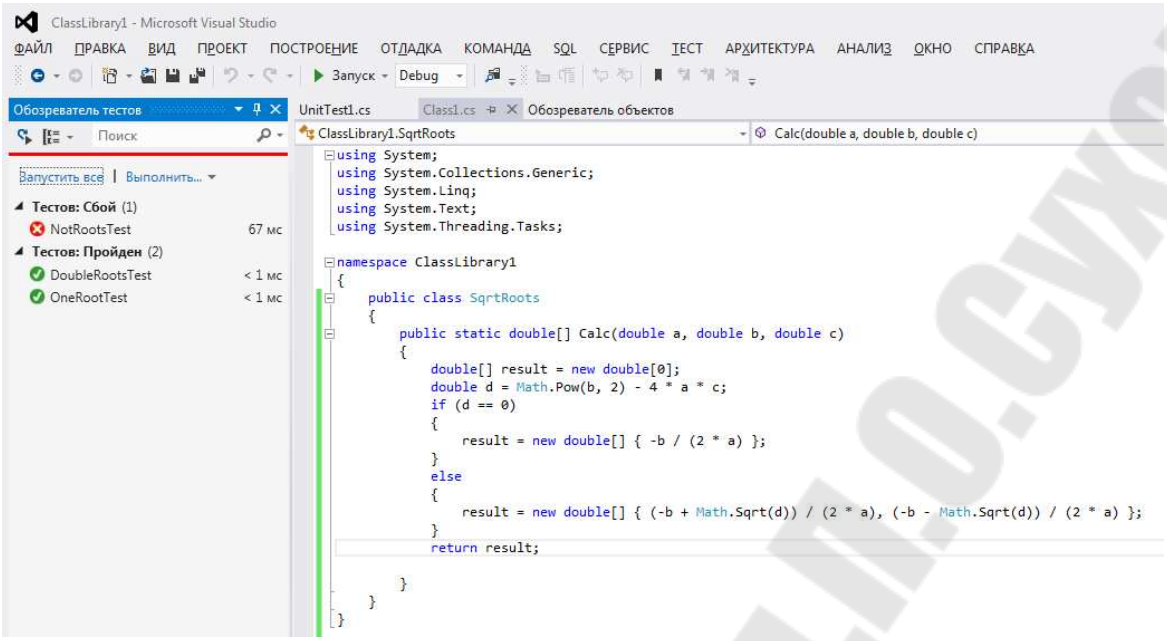


Рисунок 5.10 – Провал опять первого теста

Исправляем код:

```
public static double[] Calc(double a, double b,
double c)
{
    double[] result = new double[0];
    double d = Math.Pow(b, 2) - 4 * a * c;
    if (d == 0)
    {
        result = new double[] { -b / (2 * a) };
    }
    else if (d > 0)
    {
        result = new double[] { (-b + Math.Sqrt(d))
/ (2 * a), (-b - Math.Sqrt(d)) / (2 * a) };
    }
    return result;
}
```

Перестраиваем проект, удостоверяемся, что находимся в зеленой зоне (см. рисунок 5.11).

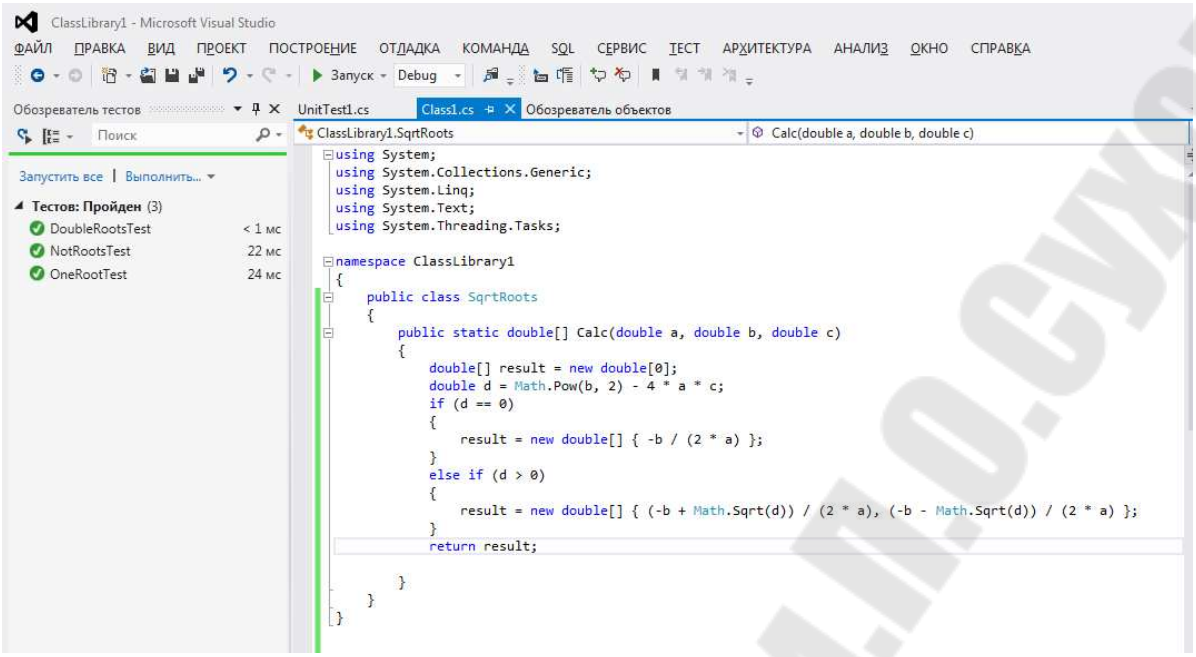


Рисунок 5.11 – Три теста проходят

Рефакторинг не требуется, можно уходить на следующую итерацию.

8. Пишем четвертый тестовый случай – уравнение не является квадратным.

Исправим код:

```
public static double[] Calc(double a, double b, double
c)
        {
            if (a == 0) throw new
InvalidOperationException("Уравнение не является
квадратным");
            else
            {
            double[] result = new double[0];
            double d = Math.Pow(b, 2) - 4 * a * c;
            if (d == 0)
            {
                result = new double[] { -b / (2 * a)
};
            }
            else if (d > 0)
            {
```

```
                result = new double[] { (-b +
Math.Sqrt(d)) / (2 * a), (-b - Math.Sqrt(d)) / (2 * a)
};
            }
            return result;
            }
    }
```

Для того необходимо, чтобы метод теста проверял, является ли исключение, возникающее в этом методе, на самом деле требуемым исключением, включим в метод теста атрибут ExpectedException:

```
        [TestMethod]

[ExpectedException(typeof(InvalidOperationException))]
        public void ArgZeroTest()
        {
            double a = 0;
            double b = -5;
            double c = 4;
            double[] actual = SqrtRoots.Calc(a, b, c);

        }
```

Перестраиваем проект, удостоверяемся, что находимся в зеленой зоне (см. рисунок 5.12).

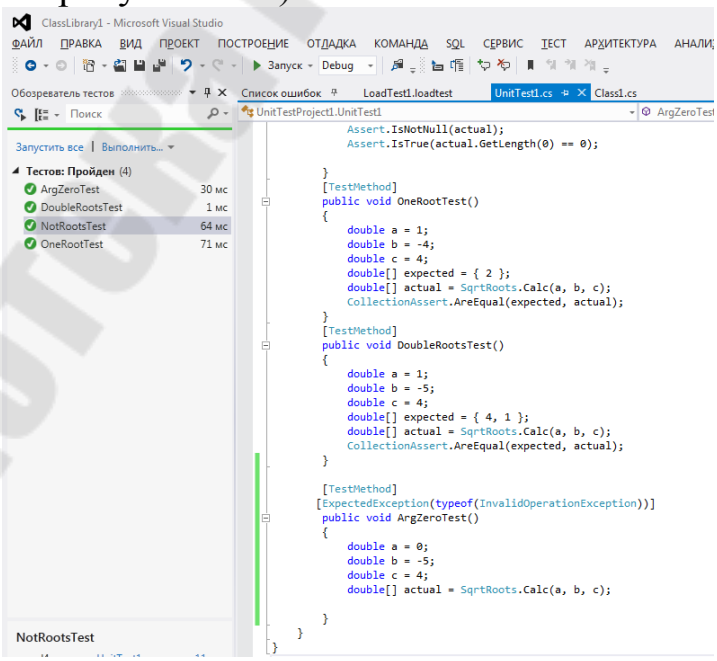


Рисунок 5.12 – Четыре теста проходят

Как видим на рисунке 5.13, для достаточно тривиального метода, количество строк кода необходимых, чтобы его протестировать больше в почти в три раза самого метода.

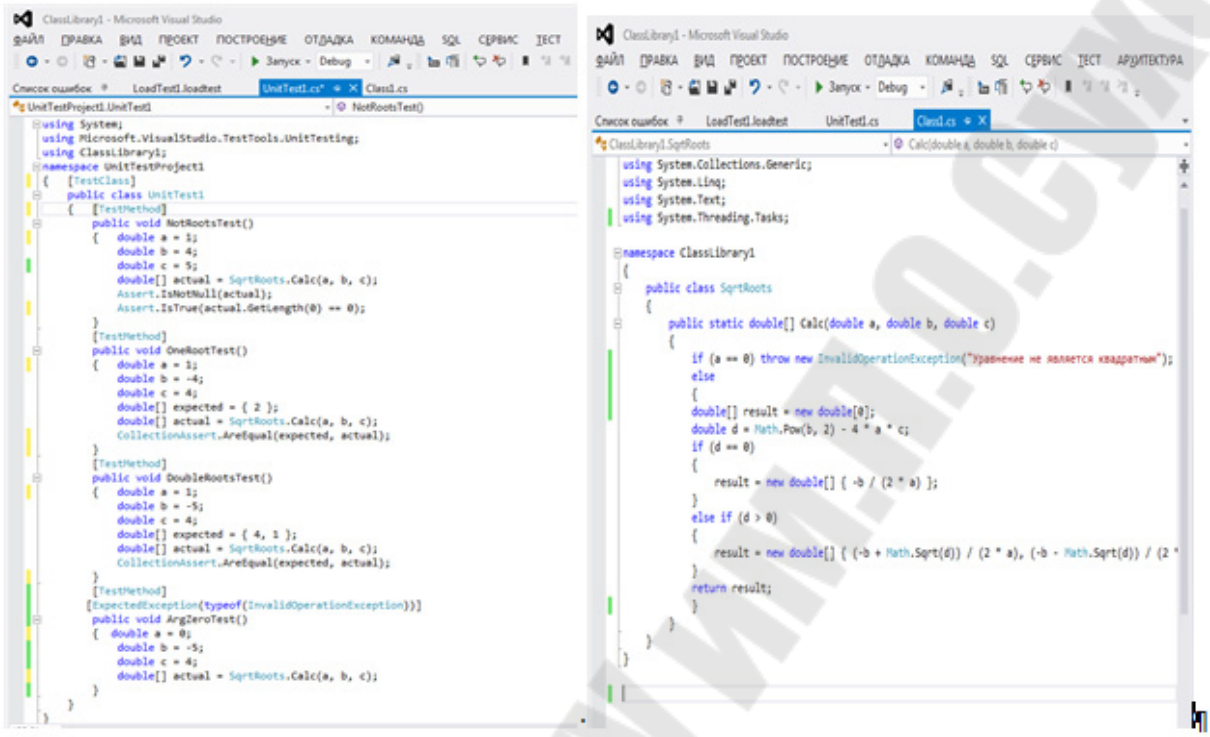


Рисунок 5.13 – Разработанный метод через тестирование

И это основная причина, почему программисты не любят модульные тесты. Кода приходится писать в 3-4 раза больше, чем необходимо для решения задачи. И это не единственный недостаток TDD. Можно привести еще несколько:

1. Так как тесты и код пишет один и тот же человек, то если он неправильно понял требования, то он написал неправильный тест, который показывает, что корректен неправильный код. Все тесты проходят, а ничего не работает.
2. При изменении требований, нам придется разбираться, какие тестовые методы перестали работать не потому, что мы «сломали код», а потому, что тесты не удовлетворяют новым требованиям.
3. Не все ситуации, можно протестировать. Например, взаимодействие между процессами (нет, мы конечно посмотрим пример, как в некоторых случаях, можно изолированно тестировать код зависящий от внешней среды, но это работает не всегда и не со всем).

У TDD есть и достоинства.

1. Программисты, использующие TDD на новых проектах, отмечают, что они реже ощущают необходимость использовать отладчик. Как только тест перестал проходить, вы сразу можете понять, где это. А если после изменения, которое вроде ничего не должно поломать, все тесты упали, то проще не отлаживать, а откатиться к стабильно работающей версии и попробовать обдумать проблему заново.
2. Разработка через тестирование также влияет на дизайн программы. Изначально сфокусировавшись на тестах, проще представить, какая функциональность необходима пользователю. Таким образом, разработчик продумывает детали интерфейса до реализации
3. Несмотря на то, что при разработке через тестирование требуется написать большее количество кода, общее время, затраченное на разработку, обычно оказывается меньше. Тесты защищают от ошибок. Поэтому время, затрачиваемое на отладку, снижается многократно.
4. Тесты позволяют производить рефакторинг кода без риска его испортить. При внесении изменений в хорошо протестированный код риск появления новых ошибок значительно ниже.
5. Разработка через тестирование способствует более модульному, гибкому и расширяемому коду. Это связано с тем, что при этой методологии разработчику необходимо думать о программе как о множестве небольших модулей, которые написаны и протестированы независимо и лишь, потом соединены вместе.
6. Поскольку пишется лишь тот код, что необходим для прохождения теста, автоматизированные тесты покрывают все пути исполнения.

ЛАБОРАТОРНАЯ РАБОТА №6 «ИНТЕГРАЦИОННОЕ ТЕСТИРОВАНИЕ»

Цель: *Познакомиться с методикой интеграционного тестирования, научиться составлять план тестирования подсистемы модулей и разрабатывать модульные тесты с использованием MVSTE.*

Постановка задачи

1. Согласно варианту задания разработать проект ClassLibrary, в состав которого входит класс **A**, содержащий поле-массив объектов класса **B**, поле название организации, поле руководитель. Фамилию, имя, отчество в классе **B** и и поле руководитель в классе **A** представлять как объект класса **Fio**.
2. Составить спецификации на все тестовые случаи для проверки взаимодействия классов **A** и **B**.
3. Создать модульные тесты на все тестовые случаи с использованием MVSTE.
4. Провести тестирование и описать выявленные дефекты

Варианты заданий

Вариант 1

Класс **Firma** реализует поток объектов класса **Sotrudnik** и содержит следующие элементы:

- закрытое поле-массив объектов класса **Sotrudnik;**
- закрытое поле director как объект класса Fio;
- закрытое поле строка fname (название фирмы);
- конструктор public **Firma**(string fileName,string direktor,string fname) принимает на вход имя файла с данными о сотрудниках и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Sotrudnik**. Второй параметр – это фамилия, имя и отчество директора, представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название фирмы;
- метод public float Sr_St()возвращает средний стаж сотрудников фирмы;
- метод public Fio Max_Staj(string otdel) возвращает фамилию, имя и отчество сотрудника заданного отдела с максимальным стажем в виде объекта класса Fio.

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:

- закрытые поля: строки фамилия, имя, отчество;
- конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;
- Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Sotrudnik** описывает объект – сотрудника фирмы и содержит следующие элементы:

- закрытые поля: Фамилия, имя, отчество (тип Fio), название отдела (string), стаж (float) и дата рождения(DateTime);
- конструктор public Sotrudnik(Fio FIO,string otdel,float staj, string DR) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также отдел, стаж и дату рождения;
- свойства: public Fio FIO, public string Otdel, public float Staj, public int Vozrast предназначены только для чтения.

Вариант 2

Класс **Firma** реализует поток объектов класса **Sotrudnik** и содержит следующие элементы:

- закрытое поле-массив объектов класса **Sotrudnik;**
- закрытое поле director как объект класса Fio;
- закрытое поле строка fname (название фирмы);
- конструктор public **Firma**(string fileName,string direktor,string fname) принимает на вход имя файла с данными о сотрудниках и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Sotrudnik**. Второй параметр – это фамилия, имя и отчество директора, представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название фирмы;
- метод public float Sr_Sarp()возвращает среднюю зарплату сотрудников фирмы;
- метод public Fio Molod(string otdel) возвращает фамилию, имя и отчество самого молодого сотрудника заданного отдела в виде объекта класса Fio.

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:

- закрытые поля: строки фамилия, имя, отчество;
- конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;
- Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Sotrudnik** описывает объект – сотрудника фирмы и содержит следующие элементы:

- закрытые поля: Фамилия, имя, отчество (тип Fio), название отдела (string), средняя зарплата (float) и дата рождения(DateTime);
- конструктор public Sotrudnik(Fio FIO,string otdel,float sr_zarp, string DR) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также отдел, среднюю зарплату и дату рождения;
- свойства: public Fio FIO, public string Otdel, public float Sr_zarp, public int Vozrast предназначены только для чтения.

Вариант 3

Класс **Firma** реализует поток объектов класса **Sotrudnik** и содержит следующие элементы:

- закрытое поле-массив объектов класса **Sotrudnik;**
- закрытое поле director как объект класса Fio;
- закрытое поле строка fname (название фирмы);
- конструктор public **Firma**(string fileName,string direktor,string fname) принимает на вход имя файла с данными о сотрудниках и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Sotrudnik**. Второй параметр – это фамилия, имя и отчество директора, представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название фирмы;
- метод public float Max_Zarp(string otdel) возвращает максимальную зарплату сотрудников фирмы заданного отдела;
- метод public Fio [] Sr_zarp_na_5 () возвращает список сотрудников, средняя зарплата которых превышает среднюю по предприятию больше чем на 5% в виде массива объектов класса **Fio**.

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:

- закрытые поля: строки фамилия, имя, отчество;
- конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;
- Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Sotrudnik** описывает объект – сотрудника фирмы и содержит следующие элементы:

- закрытые поля: Фамилия, имя, отчество (тип Fio), название отдела (string), средняя зарплата (float) и дата рождения(DateTime);
- конструктор public Sotrudnik(Fio FIO,string otdel,float sr_zarp, string DR) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также отдел, среднюю зарплату и дату рождения;
- свойства: public Fio FIO, public string Otdel, public float Sr_zarp, public int Vozrast предназначены только для чтения.

Вариант 4

Класс **Firma** реализует поток объектов класса **Sotrudnik** и содержит следующие элементы:

- закрытое поле-массив объектов класса **Sotrudnik;**
- закрытое поле director как объект класса Fio;
- закрытое поле строка fname (название фирмы);
- конструктор public **Firma**(string fileName,string direktor,string fname) принимает на вход имя файла с данными о сотрудниках и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Sotrudnik**. Второй параметр – это фамилия, имя и отчество директора, представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название фирмы;
- метод public float Sr_Vozrast() возвращает средний возраст сотрудников фирмы;
- метод public Fio [] Sr_zarp_menjche(float zarplata) возвращает список сотрудников, у которых средняя зарплата меньше заданной в виде массива объектов класса **Fio**.

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:

- закрытые поля: строки фамилия, имя, отчество;
- конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;
- Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Sotrudnik** описывает объект – сотрудника фирмы и содержит следующие элементы:

- закрытые поля: Фамилия, имя, отчество (тип Fio), название отдела (string), средняя зарплата (float) и дата рождения(DateTime);
- конструктор public Sotrudnik(Fio FIO,string otdel,float sr_zarp, string DR) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также отдел, среднюю зарплату и дату рождения;
- свойства: public Fio FIO, public string Otdel, public float Sr_zarp, public int Vozrast предназначены только для чтения.

Вариант 5

Класс **Firma** реализует поток объектов класса **Sotrudnik** и содержит следующие элементы:

- закрытое поле-массив объектов класса **Sotrudnik;**
- закрытое поле director как объект класса Fio;
- закрытое поле строка fname (название фирмы);
- конструктор public **Firma**(string fileName,string direktor,string fname) принимает на вход имя файла с данными о сотрудниках и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Sotrudnik**. Второй параметр – это фамилия, имя и отчество директора, представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название фирмы;
- метод public int Kol_Sotr() возвращает количество сотрудников, средняя зарплата которых больше средней зарплаты по всему предприятию.
- метод public Fio Min_zarp(float zarp) возвращает фамилию, имя , отчества сотрудника с минимальной средней зарплатой по предприятию и не превышающей заданную зарплату в виде объектакласса **Fio**.

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:

- закрытые поля: строки фамилия, имя, отчество;
- конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;
- Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Sotrudnik** описывает объект – сотрудника фирмы и содержит следующие элементы:

- закрытые поля: Фамилия, имя, отчество (тип Fio), название отдела (string), средняя зарплата (float) и дата рождения(DateTime);
- конструктор public Sotrudnik(Fio FIO,string otdel,float sr_zarp, string DR) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также отдел, среднюю зарплату и дату рождения;
- свойства: public Fio FIO, public string Otdel, public float Sr_zarp, public int Vozrast предназначены только для чтения.

Вариант 6

Класс **Fakultet** реализует поток объектов класса **Prepodavatel** и содержит следующие элементы:

- закрытое поле-массив объектов класса **Prepodavatel;**
- закрытое поле dekan как объект класса Fio;
- закрытое поле строка fname (название факультета);
- конструктор public **Fakultet** (string fileName,string dekan,string fname) принимает на вход имя файла с данными о преподавателях и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Prepodavatel**. Второй параметр – это фамилия, имя и отчество декана, представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название факультета;
- метод public float Max_nagruzka() возвращает максимальную нагрузку по факультету;
- метод public Fio [] Sr_nagr_boljche(float nagr) возвращает список преподавателей, у которых средняя нагрузка больше заданной в виде массива объектов класса **Fio.**

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:

- закрытые поля: строки фамилия, имя, отчество;
- конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;
- Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Prepodavatel** описывает объект – преподаватель факультета и содержит следующие элементы:

- закрытые поля: Фамилия, имя, отчество (тип Fio), название кафедры (string), средняя нагрузка в часах за 10 месяцев (float) и дата рождения(DateTime);
- конструктор public **Prepodavatel** (Fio FIO,string kaf, float nagruzka, string DR) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также кафедру, среднюю нагрузку и дату рождения;
- свойства: public Fio FIO, public string Kaf, public float Nagruzka, public int Vozrast предназначены только для чтения.

Вариант 7

Класс **Fakultet** реализует поток объектов класса **Prepodavatel** и содержит следующие элементы:

- закрытое поле-массив объектов класса **Prepodavatel;**
- закрытое поле dekan как объект класса Fio;
- закрытое поле строка fname (название факультета);
- конструктор public **Fakultet** (string fileName,string dekan,string fname) принимает на вход имя файла с данными о преподавателях и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Prepodavatel**. Второй параметр – это фамилия, имя и отчество декана, представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название факультета;
- метод public float Min_nagruzka() возвращает максимальную нагрузку по факультету;
- метод public Fio Old_Prepod(string kaf) возвращает фамилию, имя и отчество самого старого преподавателя заданной кафедры в виде объекта класса Fio.

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:

- закрытые поля: строки фамилия, имя, отчество;
- конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;
- Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Prepodavatel** описывает объект – преподаватель факультета и содержит следующие элементы:

- закрытые поля: Фамилия, имя, отчество (тип Fio), название кафедры (string), средняя нагрузка в часах за 10 месяцев (float) и дата рождения(DateTime);
- конструктор public **Prepodavatel** (Fio FIO,string kaf, float nagruzka, string DR) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также кафедру, среднюю нагрузку и дату рождения;
- свойства: public Fio FIO, public string Kaf, public float Nagruzka, public int Vozrast предназначены только для чтения.

Вариант 8

Класс **Fakultet** реализует поток объектов класса **Prepodavatel** и содержит следующие элементы:

- закрытое поле-массив объектов класса **Prepodavatel;**
- закрытое поле dekan как объект класса Fio;
- закрытое поле строка fname (название факультета);
- конструктор public **Fakultet** (string fileName,string dekan,string fname) принимает на вход имя файла с данными о преподавателях и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Prepodavatel**. Второй параметр – это фамилия, имя и отчество декана, представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название факультета;
- метод public float Sr_nagruzka(string kaf) возвращает среднюю нагрузку для заданной кафедры;
- метод public Fio Max_Prepod() возвращает фамилию, имя и отчество преподавателя с максимальной нагрузкой по факультету в виде объекта класса Fio.

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:

- закрытые поля: строки фамилия, имя, отчество;
- конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;
- Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Prepodavatel** описывает объект – преподаватель факультета и содержит следующие элементы:

- закрытые поля: Фамилия, имя, отчество (тип Fio), название кафедры (string), средняя нагрузка в часах за 10 месяцев (float) и дата рождения(DateTime);
- конструктор public **Prepodavatel** (Fio FIO,string kaf, float nagruzka, string DR) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также кафедру, среднюю нагрузку и дату рождения;
- свойства: public Fio FIO, public string Kaf, public float Nagruzka, public int Vozrast предназначены только для чтения.

Вариант 9

Класс **Fakultet** реализует поток объектов класса **Prepodavatel** и содержит следующие элементы:

- закрытое поле-массив объектов класса **Prepodavatel;**
- закрытое поле dekan как объект класса Fio;
- закрытое поле строка fname (название факультета);
- конструктор public **Fakultet** (string fileName,string dekan,string fname) принимает на вход имя файла с данными о преподавателях и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Prepodavatel**. Второй параметр – это фамилия, имя и отчество декана, представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название факультета;
- метод public int Sr_nagruzka(string kaf) возвращает количество преподавателей, средняя нагрузка которых для заданной кафедры больше средней нагрузки по факультету;

– метод public Fio Molod_Prepod() возвращает фамилию, имя и отчество самого молодого преподавателя на факультете в виде объекта класса Fio.

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:

– закрытые поля: строки фамилия, имя, отчество;
– конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;
– Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Prepodavatel** описывает объект – преподаватель факультета и содержит следующие элементы:

– закрытые поля: Фамилия, имя, отчество (тип Fio), название кафедры (string), средняя нагрузка в часах за 10 месяцев (float) и дата рождения(DateTime);
– конструктор public **Prepodavatel** (Fio FIO,string kaf, float nagruzka, string DR) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также кафедру, среднюю нагрузку и дату рождения;
– свойства: public Fio FIO, public string Kaf, public float Nagruzka, public int Vozrast предназначены только для чтения.

Вариант 10

Класс **Fakultet** реализует поток объектов класса **Prepodavatel** и содержит следующие элементы:

– закрытое поле-массив объектов класса **Prepodavatel;**
– закрытое поле dekan как объект класса Fio;
– закрытое поле строка fname (название факультета);
– конструктор public **Fakultet** (string fileName,string dekan,string fname) принимает на вход имя файла с данными о преподавателях и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Prepodavatel**. Второй параметр – это фамилия, имя и отчество декана, представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название факультета;
– метод public float Sr_Vozrast(string kaf) возвращает средний возраст преподавателей для заданной кафедры;

– метод public Fio [] Vozrast_Prepod(int vozrast) возвращает список преподавателей, возраст которых не превышает заданного возраста в виде массива объектов класса Fio.

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:

– закрытые поля: строки фамилия, имя, отчество;
– конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;
– Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Prepodavatel** описывает объект – преподаватель факультета и содержит следующие элементы:

– закрытые поля: Фамилия, имя, отчество (тип Fio), название кафедры (string), средняя нагрузка в часах за 10 месяцев (float) и дата рождения
– (DateTime);
– конструктор public **Prepodavatel** (Fio FIO,string kaf, float nagruzka, string DR) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также кафедру, среднюю нагрузку и дату рождения;
– свойства: public Fio FIO, public string Kaf, public float Nagruzka, public int Vozrast предназначены только для чтения.

Вариант 11

Класс **Organisazia** реализует поток объектов класса **Sportsmen** и содержит следующие элементы:

– закрытое поле-массив объектов класса **Sportsmen;**
– закрытое поле director как объект класса Fio;
– закрытое поле строка fname (название организации);
– конструктор public **Organisazia** (string fileName,string director,string fname) принимает на вход имя файла с данными о спортсменах и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Sportsmen**. Второй параметр – это фамилия, имя и отчество директора, представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название организации;

- метод public float Sr_resultat(string vid) возвращает средний результат для заданного вида спорта;
- метод public Fio Max_Sport() возвращает фамилию, имя и отчество спортмена с максимальным результатом по организации в виде объекта класса Fio.

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:

- закрытые поля: строки фамилия, имя, отчество;
- конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;
- Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Sportsmen** описывает объект – спортсмен организации (например, Спартак) и содержит следующие элементы:

- закрытые поля: Фамилия, имя, отчество (тип Fio), название вида спорта (string), среднbq результат (float) и дата рождения (DateTime);
- конструктор public **Sportsmen** (Fio FIO, string sport, float rezult, string DR) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также вид спорта, средний результат и дату рождения;
- свойства: public Fio FIO, public string Sport, public float Rezultat, public int Vozrast предназначены только для чтения.

Вариант 12

Класс **Organisazia** реализует поток объектов класса **Sportsmen** и содержит следующие элементы:

- закрытое поле-массив объектов класса **Sportsmen;**
- закрытое поле director как объект класса Fio;
- закрытое поле строка fname (название организации);
- конструктор public **Organisazia** (string fileName,string director,string fname) принимает на вход имя файла с данными о спортсменах и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Sportsmen**. Второй параметр – это фамилия, имя и отчество директора, представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название организации;

- метод public int Sr_resultat(float res) возвращает количество спортсменов средний результат которых больше заданного результата;
- метод public Fio Min_Sport(string vid) возвращает фамилию, имя и отчество спортмена с минимальным результатом для заданного вида спорта в виде объекта класса Fio.

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:

- закрытые поля: строки фамилия, имя, отчество;
- конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;
- Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Sportsmen** описывает объект – спортсмен организации (например, Спартак) и содержит следующие элементы:

- закрытые поля: Фамилия, имя, отчество (тип Fio), название вида спорта (string), среднbq результат (float) и дата рождения (DateTime);
- конструктор public **Sportsmen** (Fio FIO, string sport, float rezult, string DR) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также вид спорта, средний результат и дату рождения;
- свойства: public Fio FIO, public string Sport, public float Rezultat, public int Vozrast предназначены только для чтения.

Вариант 13

Класс **Organisazia** реализует поток объектов класса **Sportsmen** и содержит следующие элементы:

- закрытое поле-массив объектов класса **Sportsmen;**
- закрытое поле director как объект класса Fio;
- закрытое поле строка fname (название организации);
- конструктор public **Organisazia** (string fileName,string director,string fname) принимает на вход имя файла с данными о спортсменах и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Sportsmen**. Второй параметр – это фамилия, имя и отчество директора,

представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название организации;
- метод public int Sr_resultat(float res1, float res2) возвращает количество спортсменов средний результат которых находится в заданном диапозоне.
- метод public Fio Molod_Sport(string vid) возвращает фамилию, имя и отчество самого молодого спортмена для заданного вида спорта в виде объекта класса Fio.

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:
- закрытые поля: строки фамилия, имя, отчество;
- конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;
- Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Sportsmen** описывает объект – спортсмен организации (например, Спартак) и содержит следующие элементы:
- закрытые поля: Фамилия, имя, отчество (тип Fio), название вида спорта (string), среднbq результат (float) и дата рождения (DateTime);
- конструктор public **Sportsmen** (Fio FIO, string sport, float rezult, string DR) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также вид спорта, средний результат и дату рождения;
- свойства: public Fio FIO, public string Sport, public float Rezultat, public int Vozrast предназначены только для чтения.

Вариант 14

Класс **Organisazia** реализует поток объектов класса **Sportsmen** и содержит следующие элементы:
- закрытое поле-массив объектов класса **Sportsmen;**
- закрытое поле director как объект класса Fio;
- закрытое поле строка fname (название организации);
- конструктор public **Organisazia** (string fileName,string director,string fname) принимает на вход имя файла с данными о спортсменах и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Sportsmen**.

Второй параметр – это фамилия, имя и отчество директора, представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название организации;

- метод public int Sr_resultat(string vid) возвращает количество спортсменов средний результат которых находится между минимальным и максимальным результатом в заданном виде спорта.
- метод public Fio Old_Sport() возвращает фамилию, имя и отчество самого старого спортмена в организации в виде объекта класса Fio.

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:

- закрытые поля: строки фамилия, имя, отчество;
- конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;
- Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Sportsmen** описывает объект – спортсмен организации (например, Спартак) и содержит следующие элементы:

- закрытые поля: Фамилия, имя, отчество (тип Fio), название вида спорта (string), среднbq результат (float) и дата рождения (DateTime);
- конструктор public **Sportsmen** (Fio FIO, string sport, float rezult, string DR) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также вид спорта, средний результат и дату рождения;
- свойства: public Fio FIO, public string Sport, public float Rezultat, public int Vozrast предназначены только для чтения.

Вариант 15

Класс **Organisazia** реализует поток объектов класса **Sportsmen** и содержит следующие элементы:

- закрытое поле-массив объектов класса **Sportsmen;**
- закрытое поле director как объект класса Fio;
- закрытое поле строка fname (название организации);
- конструктор public **Organisazia** (string fileName,string director,string fname) принимает на вход имя файла с данными о

спортсменах и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Sportsmen**. Второй параметр – это фамилия, имя и отчество директора, представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название организации;

- метод public int Sr_resultat() возвращает количество спортсменов средний результат которых превышает максимальный результат в зпортивнной организации.
- метод public Fio [] Old_Sport(int vozr) возвращает список спортсменов моложе заданного возраста в организации в виде массива объектов класса Fio.

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:

- закрытые поля: строки фамилия, имя, отчество;
- конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;
- Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Sportsmen** описывает объект – спортсмен организации (например, Спартак) и содержит следующие элементы:

- закрытые поля: Фамилия, имя, отчество (тип Fio), название вида спорта (string), среднbq результат (float) и дата рождения (DateTime);
- конструктор public **Sportsmen** (Fio FIO, string sport, float rezult, string DR) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также вид спорта, средний результат и дату рождения;
- свойства: public Fio FIO, public string Sport, public float Rezultat, public int Vozrast предназначены только для чтения.

Методические указания

Интеграционное тестирование – одна из фаз тестирования программного обеспечения, при котором отдельные программные модули объединяются и тестируются в группе.

Обычно интеграционное тестирование проводится после модульного тестирования и предшествует системному тестированию.

Цель *интеграционного тестирования* – проверка связи между компонентами, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между различными системами).

Уровни интеграционного тестирования:

- компонентный интеграционный уровень (Component Integration testing). Проверяется взаимодействие между компонентами системы после проведения компонентного тестирования.
- системный интеграционный уровень (System Integration Testing). Проверяется взаимодействие между разными системами после проведения системного тестирования.

Пример выполнения

Постановка задачи.

1. Разработать проект StudentsLibrary1, в состав которого входит:

Класс **Fakultet** реализует поток объектов класса **Student** и содержит следующие элементы:

- закрытое поле-массив объектов класса **Student;**
- закрытое поле dekan как объект класса Fio;
- закрытое поле строка fname (название факультета);
- конструктор public **Fakultet** (string fileName,string dekan,string fname) принимает на вход имя файла с данными о преподавателях и записывает данные из файла в поле класса, представляющее собой массив объектов класса **Student**. Второй параметр – это фамилия, имя и отчество декана, представленные в виде одной строки (разделитель слов – пробел), третий параметр – это название факультета;
- метод public float Sball_Gruppy(string grupp) возвращает средней балл для заданной группы;
- метод public Fio Lider() возвращает фамилию, имя и отчество самого самого лучшего студента на факультете в виде объекта класса Fio.

Класс **Fio** реализует объект: совокупность фамилии имени и отчества человека и содержит следующие элементы:

- закрытые поля: строки фамилия, имя, отчество;
- конструктор public Fio(string fam, string name, string otch) принимает на вход фамилию, имя и отчество в виде трех отдельных строк;

– Свойства: public string Fam, public string Name, public string Otch возвращают значения фамилии, имени и отчества соответственно (только для чтения).

Класс **Student** описывает объект – студент факультета и содержит следующие элементы:

– закрытые поля: Фамилия, имя, отчество (тип Fio) студента, название группы (string), средний балл (float);
– конструктор public **Student** (Fio FIO,string gruppa, float s_ball) принимает на вход фамилию, имя и отчество в виде объекта класса Fio, а также группу, средний балл;
– свойства: public Fio FIO, public string Gruppa, public float S_ball предназначены только для чтения.

2. Составить спецификации на все тестовые случаи для проверки взаимодействия классов **Fakultet** и **Student**.
3. Создать модульные тесты на все тестовые случаи с использованием MVSTE.
4. Провести тестирование и описать выявленные дефекты

Порядок выполнения.

1. Создать в Visual Studio, выбрав язык C#, проект Class Library (Библиотека классов) (см. рисунок 2.1) и сохранить под именем StudentsLibrary1.
2. Файл Class1.cs переименовать в Students.cs.
3. Класс Class1 переименовать в класс Fakultet.
4. Добавить еще два класса: класс Fio, класс Student, выбрав пункт меню Проект, а затем Добавить класс...

На рисунках 6.1 – 6.3 представлены классы Fio, Student, Fakultet.

5. Добавить проект для тестирования. Правой кнопкой щёлкните по решению, выберите «Добавить» и затем «Создать проект…» (см. рисунок 6.4).
6. В открывшемся окне в группе Visual C# щёлкните «Тест», а затем выберите «Проект модульного теста», назовите проект StudentsTestProject1 и нажмите «ОК». Таким образом, проект будет создан (см. рисунок 6.5).
7. Затем в References (Ссылки) проекта StudentsTestProject1 необходимо добавить ссылку на проект, код которого будем тестировать. Правой кнопкой щёлкаем на References, а затем выбираем «Добавить ссылку…» . В появившемся окне раскрываем группу «Решение», выбираем «Проекты» и ставим

галочку напротив проекта StudentsLibrary1. Затем нажать «ОК» (см. рисунок 6.6).

8. Подключить с помощью директивы using следующее пространство имён: `using StudentsLibrary1;`
9. В папке /Bin/Debug проекта StudentsTestProject1 сохранить тестовый файл test.txt (см. рисунок 6.7).
10. Провести тестирование методов класса Fakultet (см рисунок 6.8)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace StudentsLibrary1
{
    public class Fio
    {   private string fam;
        private string name;
        private string otch;
        public Fio(string fam, string name, string otch)
        {   this.fam = fam;
            this.name = name;
            this.otch = otch;
        }
        public string Fam
        {
            get { return fam; }
        }
        public string Name
        {
            get { return name; }
        }
        public string Otch
        {
            get { return otch; }
        }
    }
}
```

Рисунок 6.1 – Класс Fio

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace StudentsLibrary1
{
    public class Student
    {
        Fio fio;
        string gruppa;
        float s_ball;
        public Student(Fio fio, string gruppa, float s_ball)
        {
            this.fio = new Fio(fio.Fam, fio.Name, fio.Otch);
            this.gruppa = gruppa;
            this.s_ball = s_ball;
        }
        public Fio FIO
        {
            get { return fio; }
        }
        public string Gruppa
        {
            get { return gruppa; }
        }
        public float S_ball
        {
            get { return s_ball; }
        }
    }
}
```

Рисунок 6.2 – Класс Student

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace StudentsLibrary1
{   public class Fakultet
    {   Student[] students;
        Fio dekan;
        string fname;
        public Fakultet(string Filename, string dekan, string fname)
        {   string[] s = System.IO.File.ReadAllLines(Filename);
            int n = s.Length;
            students = new Student[n];
            for (int i = 0; i < n; i++)
            {   Fio fio = new Fio(s[i].Split()[0], s[i].Split()[1], s[i].Split()[2]);
                string gruppa = s[i].Split()[3];
                float s_ball = Single.Parse(s[i].Split()[4]);
                students[i] = new Student(fio, gruppa, s_ball);
            }
            this.fname = fname;
            this.dekan = new Fio(dekan.Split()[0], dekan.Split()[1], dekan.Split()[2]);
        }
        public float Sball_Gruppy(string grupp)
        {   int k = 0;
            float S = 0;
            for (int i = 0; i < students.Length; i++)
            {
                if (students[i].Gruppa == grupp)
                {   k++;
                    S += students[i].S_ball;
                } }
            return S / k;}
        public Fio Lider()
        {   Fio lider = new Fio("", "", "");
            float mark = 0;
            for (int i = 0; i < students.Length; i++)
            {  if (mark < students[i].S_ball)
                {   mark = students[i].S_ball;
                    lider = new Fio(students[i].FIO.Fam, students[i].FIO.Name, students[i].FIO.Otch);
                } }
            return lider;
        } }}
```

Рисунок 6.3 – Класс Fakultet

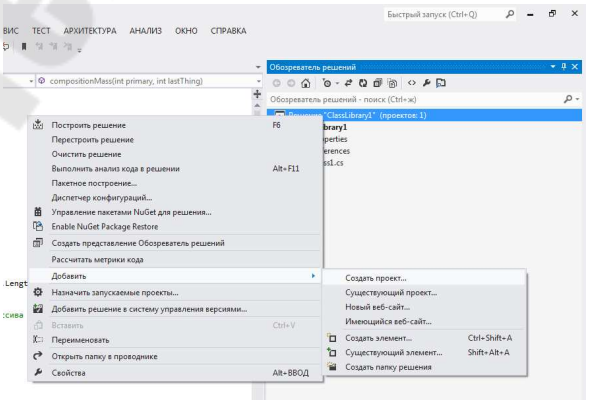


Рисунок 6.4 – Добавить новый проект

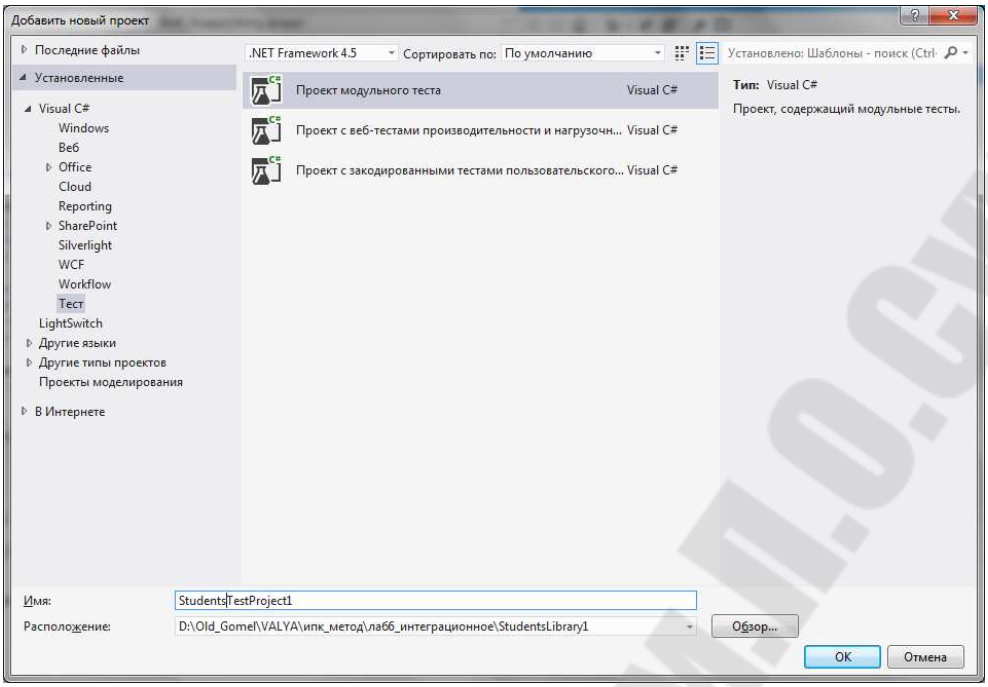


Рисунок 6.5 – Выбор проекта модульного теста

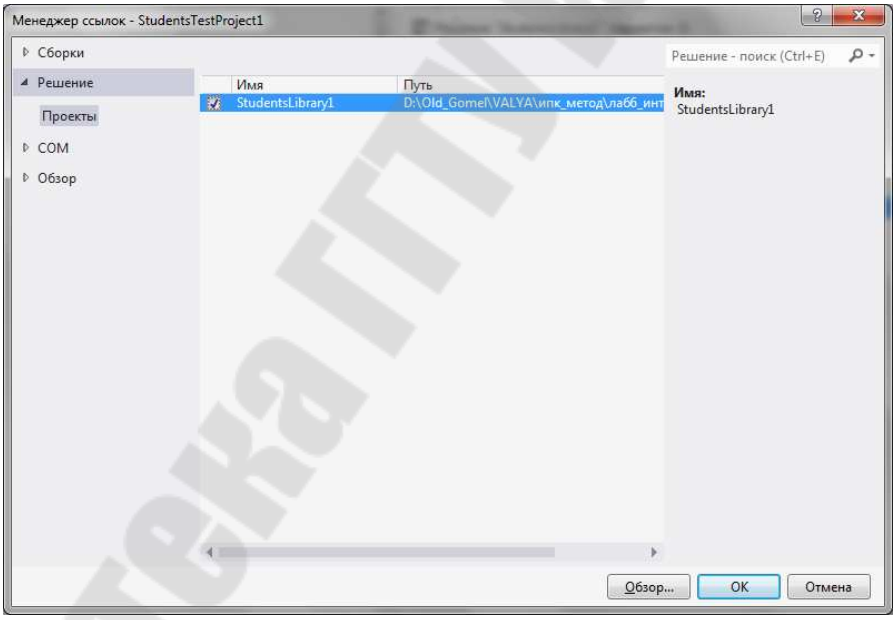


Рисунок 6.6 – Добавление ссылки

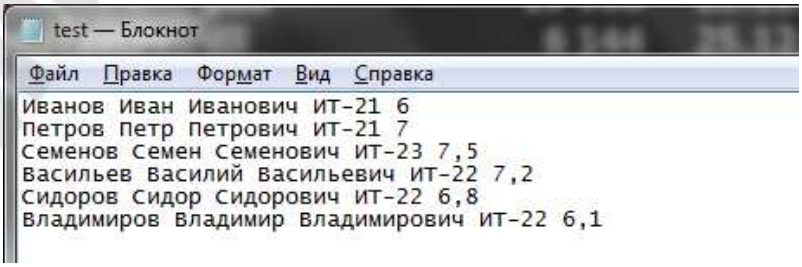


Рисунок 6.7 – Файл с данными для тестирования

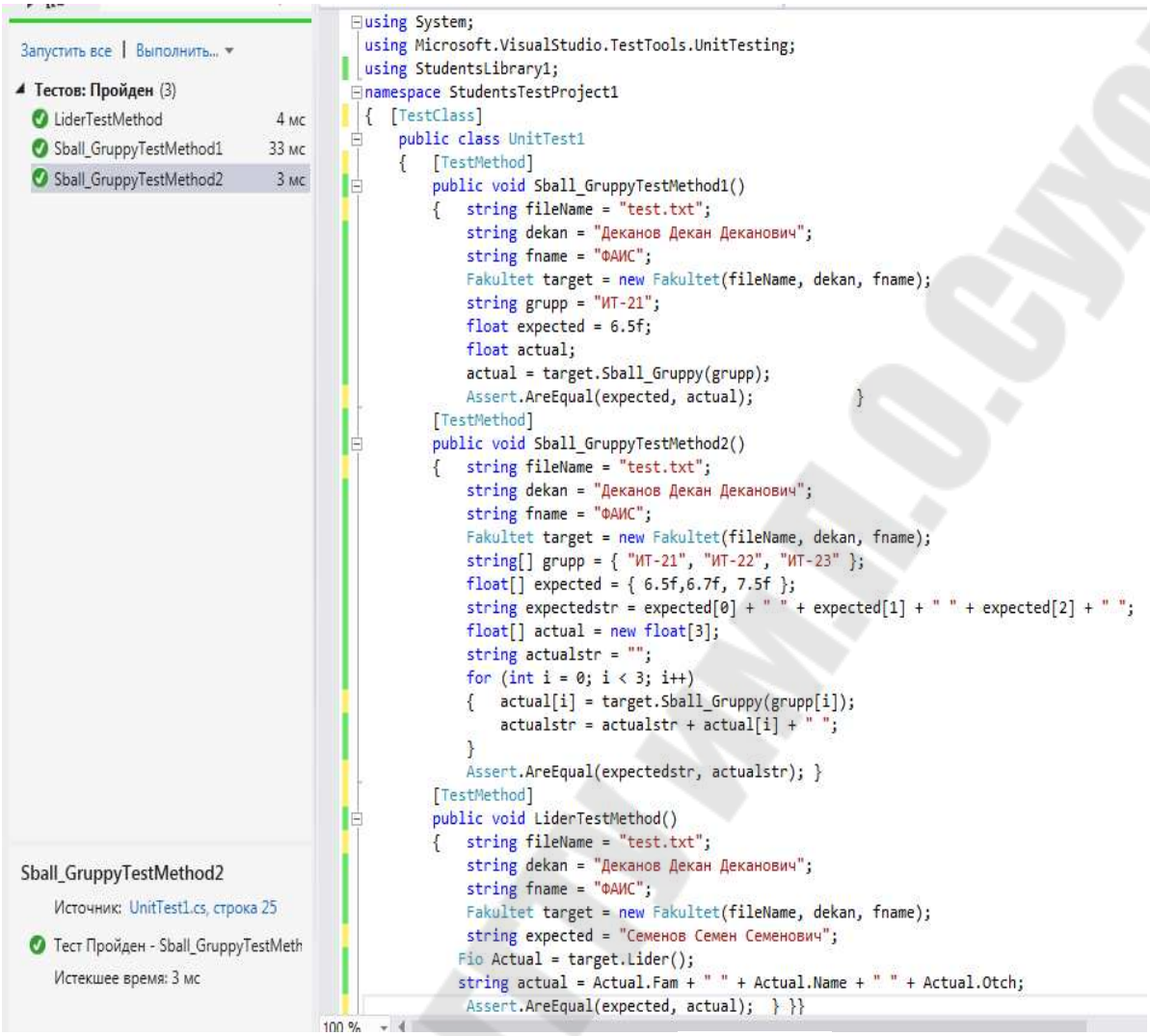


Рисунок 6.8 – Класс UnitTest1

11. Добавим в класс UnitTest1 еще один тестовый метод с именем группы, которой нет в файле с тестовыми данными:

```
[TestMethod]
        public void Sball_GruppyTestMethod3()
        {
            string fileName = "test.txt";
            string dekan = "Деканов Декан
Деканович";
            string fname = "ФАИС";
            Fakultet target = new Fakultet(fileName,
dekan, fname);
            string grupp = "ИТ-25";
            float expected = 6.5f;
```

```
            float actual;
            actual = target.Sball_Gruppy(grupp);
            Assert.AreEqual(expected, actual);

        }
```

12. Выполним все тесты. Видим, что последний тест провалился (см. рисунок 6.9).
13. Выявлен дефект в методе public float Sball_Gruppy(string grupp). Если заданной группы нет в массиве объектов Student, то происходит деление на 0.

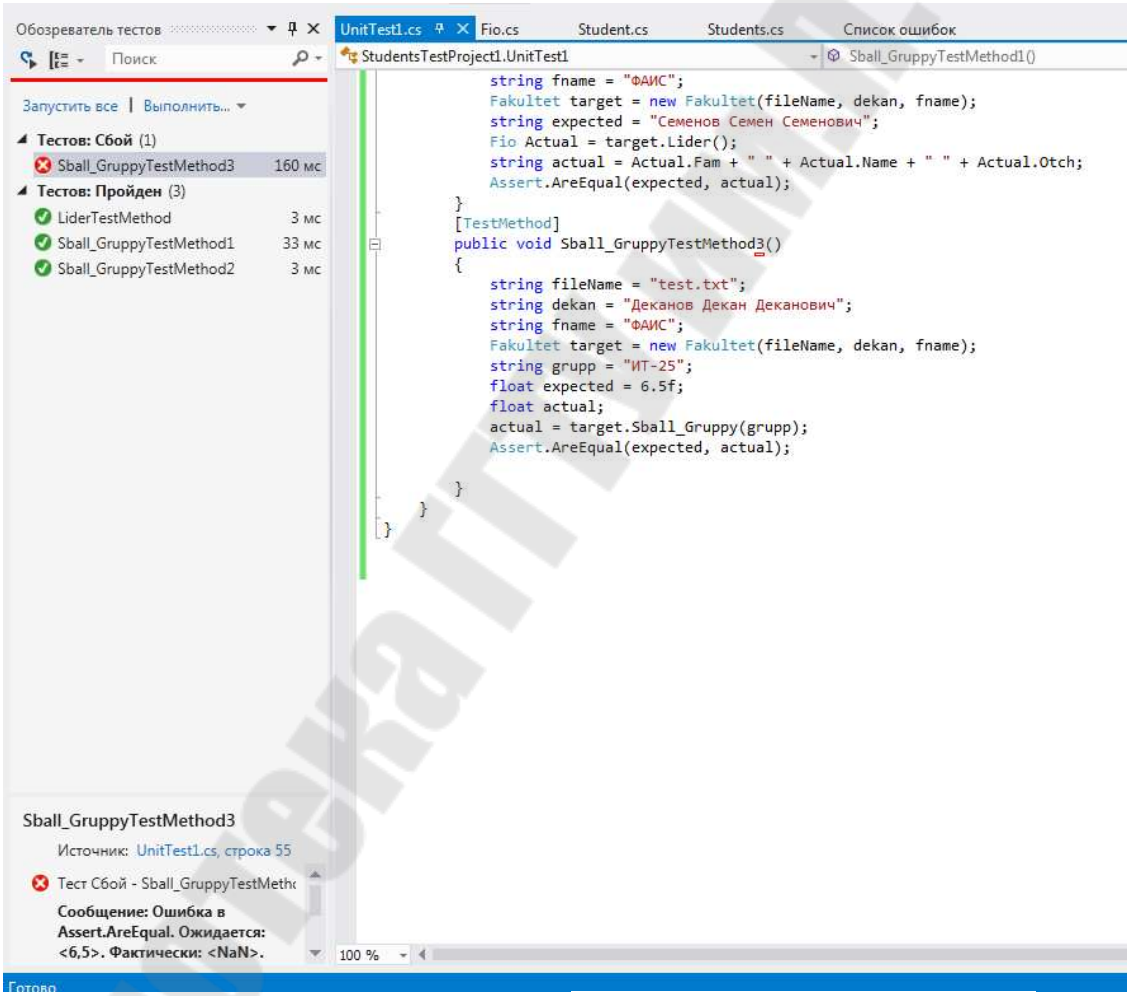


Рисунок 6.9 – Провал теста Sball_GruppyTestMethod3()

Список рекомендуемой литературы

1. Бек К. Экстремальное программирование: разработка через тестирование. – СПб.: Питер, 2003. – 224 с.
2. Винниченко И. В. Автоматизация процессов тестирования: производственно-практическое издание. – СПб. : Питер, 2005. – 202 с.
3. Иванова Г.С. Технология программирования. – М.: Из-во МГТУ им. Баумана 2002, – 241 с
4. Липаев В.В. Тестирование компонентов и комплексов программ. – М.: Синтег, 2010. – 399 с.
5. Майерс Гленфорд Дж. Искусство тестирования программ. – М.: Финансы и статистика, 1982. – 176 с.

Мурашко Валентина Семеновна

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Практикум
по одноименной дисциплине
для слушателей специальности переподготовки
1-40 01 73 «Программное обеспечение информационных систем»
заочной формы обучения

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 14.01.19.
Рег. № 79Е.
http://www.gstu.by