

Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Кузбасский государственный технический университет
имени Т. Ф. Горбачева»

Институт профессионального образования
Кафедра информатики и информационных систем

Романова Вера Васильевна

ТЕСТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Методические материалы к практическим занятиям
и самостоятельной работе

Рекомендовано цикловой методической комиссией
специальности 09.02.07 Информационные системы
и программирования в качестве электронного издания
для использования в образовательном процессе

Кемерово 2024

Рецензенты: О.С. Семенова – кандидат тех. наук, заведующий кафедрой информатики и информационных систем ИПО ФГБОУ ВО «Кузбасский государственный технический университет имени Т. Ф. Горбачева».

Романова, В.В. Тестирование информационных систем: методические материалы к практическим занятиям и самостоятельной работе для обучающихся специальности СПО 09.02.07 «Информационные системы и программирование» всех форм обучения / сост. Романова В.В; Кузбасский государственный технический университет имени Т. Ф. Горбачева. – Кемерово, 2024. – Текст: электронный.

Приведено содержание практических работ, порядок их оформления, а также материал, необходимый для успешного изучения дисциплины. Назначение издания – помощь обучающимся в получении знаний по дисциплине «Тестирование информационных систем» и организация практических работ.

© Кузбасский государственный
технический университет
имени Т. Ф. Горбачева, 2024
© Романова В.В.,
составление, 2024

ОГЛАВЛЕНИЕ

ПРАКТИЧЕСКАЯ РАБОТА №1. ИЗУЧЕНИЕ СРЕДСТВ ТЕСТИРОВАНИЯ В MS VISUAL STUDIO	4
ПРАКТИЧЕСКАЯ РАБОТА №2. ВЫПОЛНЕНИЕ МОДУЛЬНОГО ТЕСТИРОВАНИЯ В MS VISUAL STUDIO	12
ПРАКТИЧЕСКАЯ РАБОТА №3. РАЗРАБОТКА ТЕСТ-КЕЙСА ДЛЯ УЧЕБНОЙ ПРОГРАММЫ	20
ПРАКТИЧЕСКАЯ РАБОТА №4. СОСТАВЛЕНИЕ ПЛАНА ТЕСТИРОВАНИЯ УЧЕБНОЙ ПРОГРАММЫ	30
ПРАКТИЧЕСКАЯ РАБОТА №5. ОСНОВЫ РАБОТЫ В СИСТЕМЕ ОТСЛЕЖИВАНИЯ ОШИБОК	32
ПРАКТИЧЕСКАЯ РАБОТА №6. ОСНОВЫ РАБОТЫ В СИСТЕМЕ SELENIUM	38
ПРАКТИЧЕСКАЯ РАБОТА №7. АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ ПРИЛОЖЕНИЯ В СИСТЕМЕ SELENIUM ..	43
ПРАКТИЧЕСКАЯ РАБОТА №8. АВТОМАТИЗАЦИЯ	54
ТЕСТИРОВАНИЯ БАЗЫ ДАННЫХ.....	54
ПРАКТИЧЕСКАЯ РАБОТА №9. ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ПРИЛОЖЕНИЯ	59
ПРАКТИЧЕСКАЯ РАБОТА №10. ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ ВЕБ-СЕРВИСА В СИСТЕМЕ SELENIUM....	69
ПРАКТИЧЕСКАЯ РАБОТА №11. НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ WEB-ПРИЛОЖЕНИЯ В СИСТЕМЕ SELENIUM	72
СОДЕРЖАНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ.....	77
ЛИТЕРАТУРА	78

ПРАКТИЧЕСКАЯ РАБОТА №1. ИЗУЧЕНИЕ СРЕДСТВ ТЕСТИРОВАНИЯ В MS VISUAL STUDIO

Цель работы – изучение средств тестирования в MS Visual Studio.

Задачами работы является: написание теста, использование Visual Studio для запуска и отладки тестов, совершенствование навыков написания тестов.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

В этой работе вы ознакомитесь с инструментами тестирования Visual Studio. В примерах будут использоваться C# и .NET, однако многие функции запуска одинаковы для множества языков, поддерживаемых инструментами тестирования Visual Studio.

Вы начинаете тестирование программы с трех основных задач.

Написание теста. Изучите основные части написания теста и воспользуйтесь тестовыми проектами, которые ссылаются на код продукта.

Использование Visual Studio для запуска и отладки тестов. Просматривайте выходные данные тестов и работайте с целым набором тестов.

Совершенствование навыков написания тестов. Используйте Fluent Assertions, тесты на основе данных и макетирование для расширения навыков тестирования. (изучить средства тестирования MS Visual Studio, выполнить задания)

По окончании этого вы сможете создать тестовый проект, добавить ссылки на код продукта и написать тесты. Вы также научитесь использовать инструменты тестирования в Visual Studio для запуска, упорядочения и отладки тестов. Модульное тестирование, или юнит-тестирование (англ. unit testing) – процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Цель модульного тестирования – изолировать отдельные части программы и показать, что по отдельности эти части работоспособны.

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №1

1. Задание 1. Создание проекта программы, модули которого будут тестироваться

Разработаем проект содержащий класс, который вычисляет площадь прямоугольника по длине двух его сторон.

Создадим в Visual Studio новый проект Visual C# -> Библиотека классов. Назовём его MathTaskClassLibrary.

Class1 переименуем в Geometry.

В классе реализуем метод, вычисляющий площадь прямоугольника. Для демонстрации остановимся на работе с целыми числами. Код программы приведён ниже (рис.1.1).

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace MathTaskClassLibrary
8 {
9     public class Geometry
10    {
11        public int RectangleArea(int a, int b)
12        {
13            return a * b;
14        }
15    }
16 }

```

Рисунок 1.1 – Код программы

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...”. (рис.1.2).

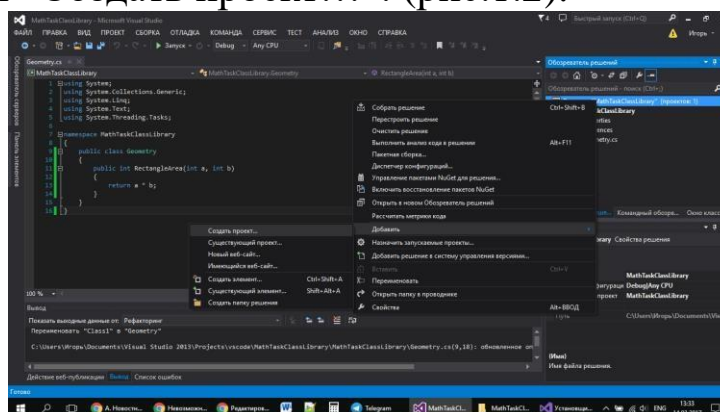


Рисунок 1.2 – Окно создания проекта

В открывшемся окне в группе Visual C# щёлкните “Тест”, а затем выберите “Проект модульного теста”. Введите имя проекта MathTaskClassLibraryTests и нажмите “ОК”. Таким образом проект будет создан (рис.1.3).

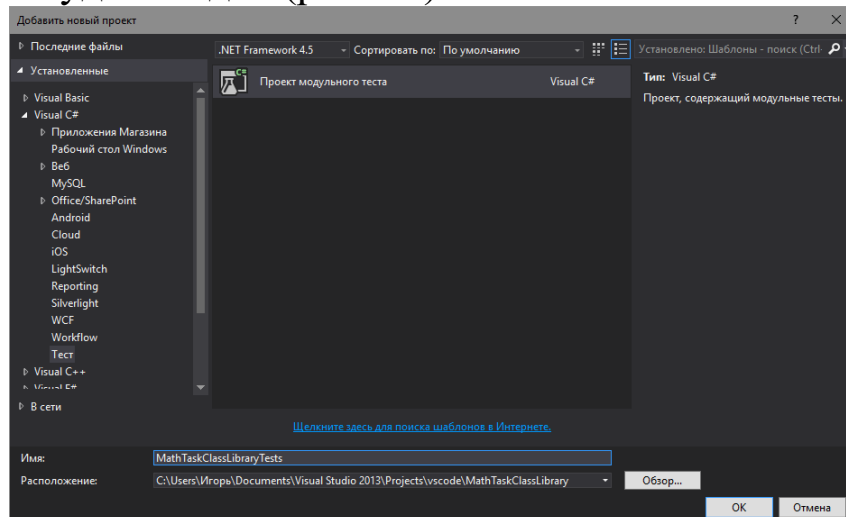


Рисунок 1.3 – Окно создания проекта модульного теста

Перед Вами появится следующий код (рис.1.4):

```

1 using System;
2 using Microsoft.VisualStudio.TestTools.UnitTesting;
3
4 namespace MathTaskClassLibraryTests
5 {
6     [TestClass]
7     public class UnitTest1
8     {
9         [TestMethod]
10        public void TestMethod1()
11        {
12        }
13    }
14 }

```

Рисунок 1.4 –Код unit теста

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.

В соответствии с принятыми соглашениями переименуем класс UnitTest1 в GeometryTests.

Затем в References 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

MathTaskClassLibrary. Затем жмём “OK” (рис.1.5).

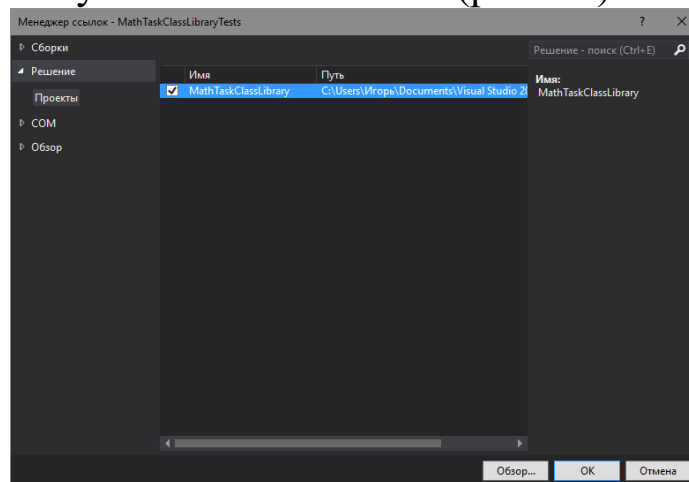


Рисунок 1.5 – Добавление ссылки на проект

Также в коде необходимо подключить с помощью директивы `using` следующее пространство имён: `using MathTaskClassLibrary;`

Займёмся написание теста. Проверим правильно ли вычисляет программа площадь прямоугольника со сторонами 3 и 5. Ожидаемый результат (правильное решение) в данном случае это число 15.

Переименуем метод `TestMethod1()` в `RectangleArea_3and5_15returned()`. Новое название метода поясняет, что будет проверяться (RectangleArea – площадь прямоугольника) для каких значений (3 и 5) и что ожидается в качестве правильного результата (15 returned).

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 будет таким(рис.1.6):

```

1 using System;
2 using Microsoft.VisualStudio.TestTools.UnitTesting;
3 using MathTaskClassLibrary;
4
5 namespace MathTaskClassLibraryTests
6 {
7     [TestClass]
8     public class GeometryTests
9     {
10         [TestMethod]
11         public void RectangleArea_3and5_15returned()
12         {
13             // исходные данные
14             int a = 3;
15             int b = 5;
16             int expected = 15;
17
18             // получение значения с помощью тестируемого метода
19             Geometry g = new Geometry();
20             int actual = g.RectangleArea(a, b);
21
22             // сравнение ожидаемого результата с полученным
23             Assert.AreEqual(expected, actual);
24         }
25     }
26 }

```

Рисунок 1.6 – Тестирующий код

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

Теперь, 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”(рис.1.7).

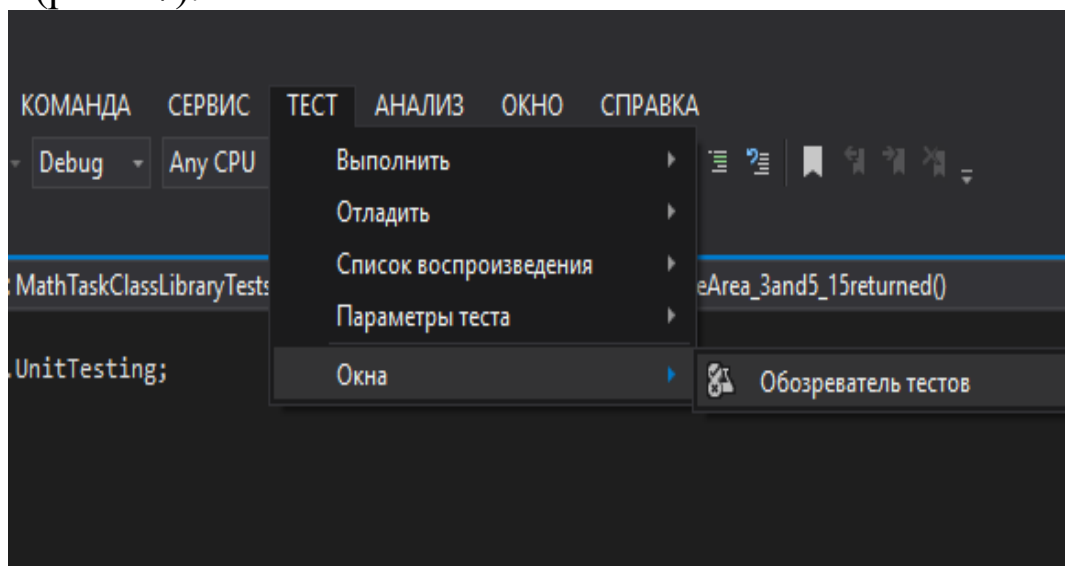


Рисунок 1.7 – Обозреватель тестов

В студии появится следующее окно (рис.1.8):

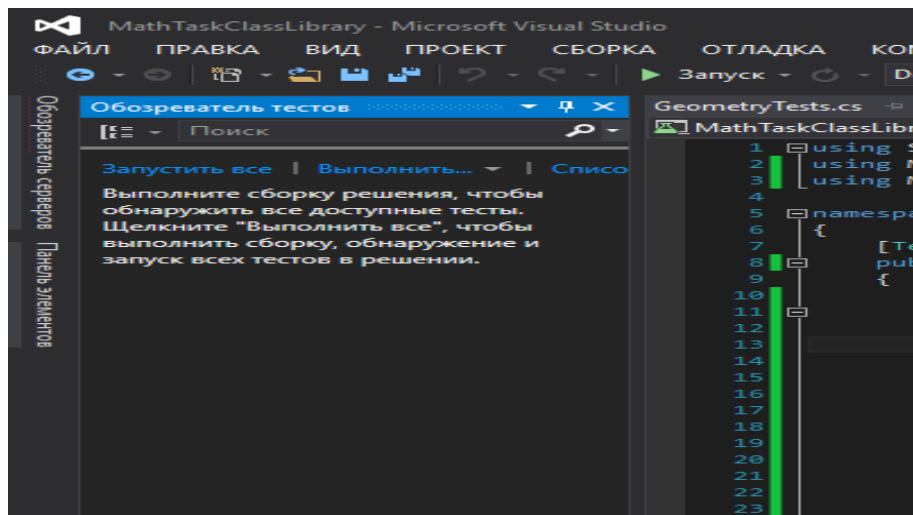


Рисунок 1.8 – Окно Обозревателя тестов

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 (рис.1.9).

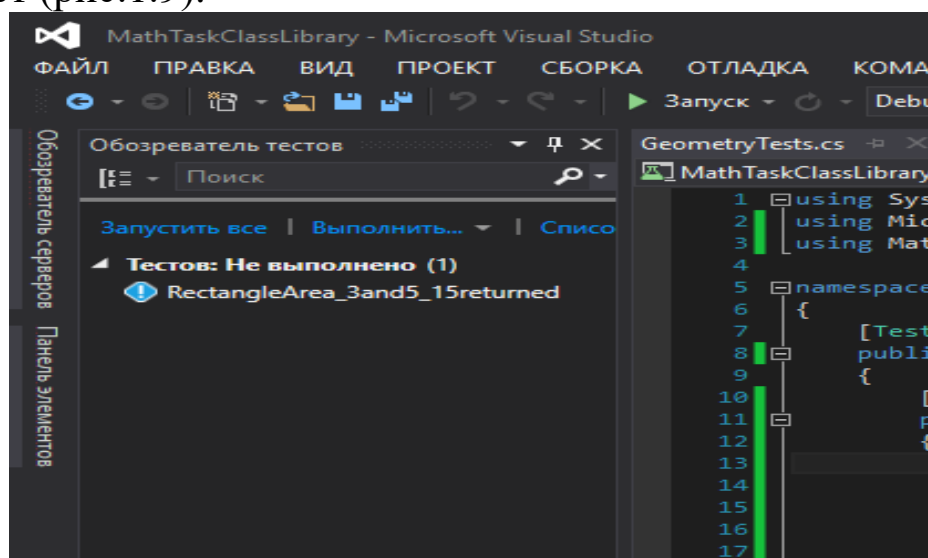


Рисунок 1.9 – Запуск теста

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”(рис.1.10).

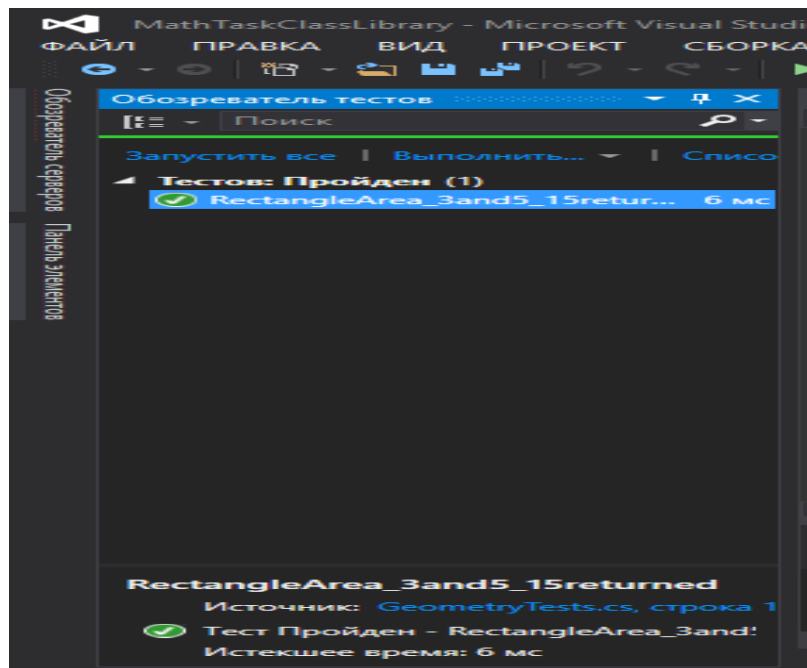


Рисунок 1.10 – Модульный тест успешно пройден

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

Изменим код метода `RectangleArea`, вычисляющего площадь прямоугольника, чтобы симитировать провал теста и посмотреть, как поведёт себя Visual Studio. Прибавим к возвращаемому значению 10.

Запустим unit-тест.

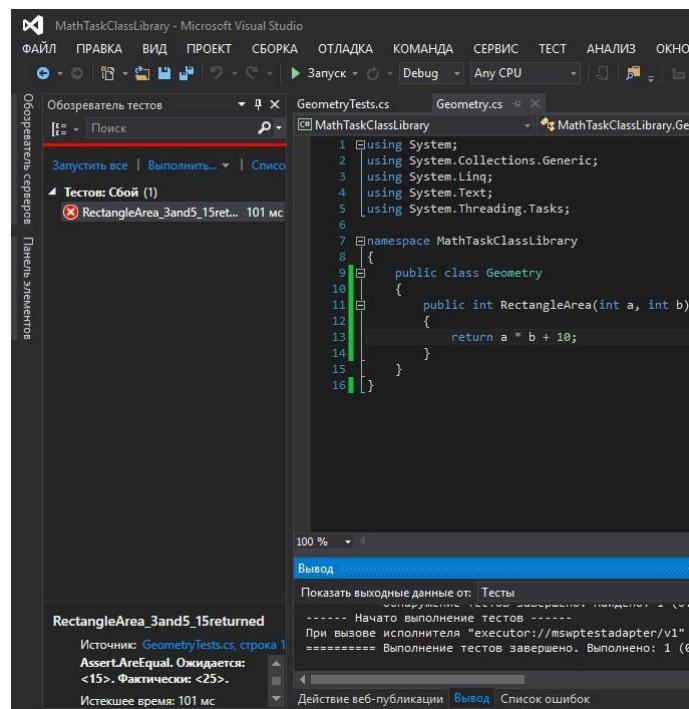


Рисунок 1.11 – Провал модульного теста

Как Вы видите, красный круг с крестиком показывает провал модульного теста ”(рис.1.11), а ниже указано, что при проверке ожидалось значение 15, а по факту оно равно 25.

Задание 2. Разработать программу для подсчета объема цилиндра и создать модульный тест.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Понятие тестирования ПО.
2. Цели тестирования.
3. Виды тестирования.
4. Какие виды тестирования обязательно применять в процессе разработки сложного ПО?
5. Модульное тестирование. Понятие модуля.
6. Валидация и верификация. Тестирование методом "чёрного" и "белого" ящика.

ПРАКТИЧЕСКАЯ РАБОТА №2. ВЫПОЛНЕНИЕ МОДУЛЬНОГО ТЕСТИРОВАНИЯ В MS VISUAL STUDIO

Цель работы: изучить возможность создания автоматических тестов, для модульного тестирования, получение базовых навыков разработки и реализации модульных тестов в среде NUnit

Задачами работы является: написание теста, использование Visual Studio для запуска и отладки тестов, совершенствование навыков написания тестов.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Фреймворки для unit-тестирования в основном используются для того, чтобы помочь быстро и легко написать модульные тесты. Большинство языков программирования не поддерживают модульное тестирование встроенным компилятором. Чтобы сделать unit-тестирование еще более увлекательным, можно использовать сторонние инструменты с открытым исходным кодом и платные инструменты.

Список популярных инструментов unit-тестирования для различных языков программирования:

Java framework – JUnit

PHP-фреймворк – PHPUnit

Фреймворки C++ – UnitTest++ и Google C++

Фреймворк .NET – NUnit

Фреймворк Python – pytest

Создание простого набора тестов с использованием NUnit:

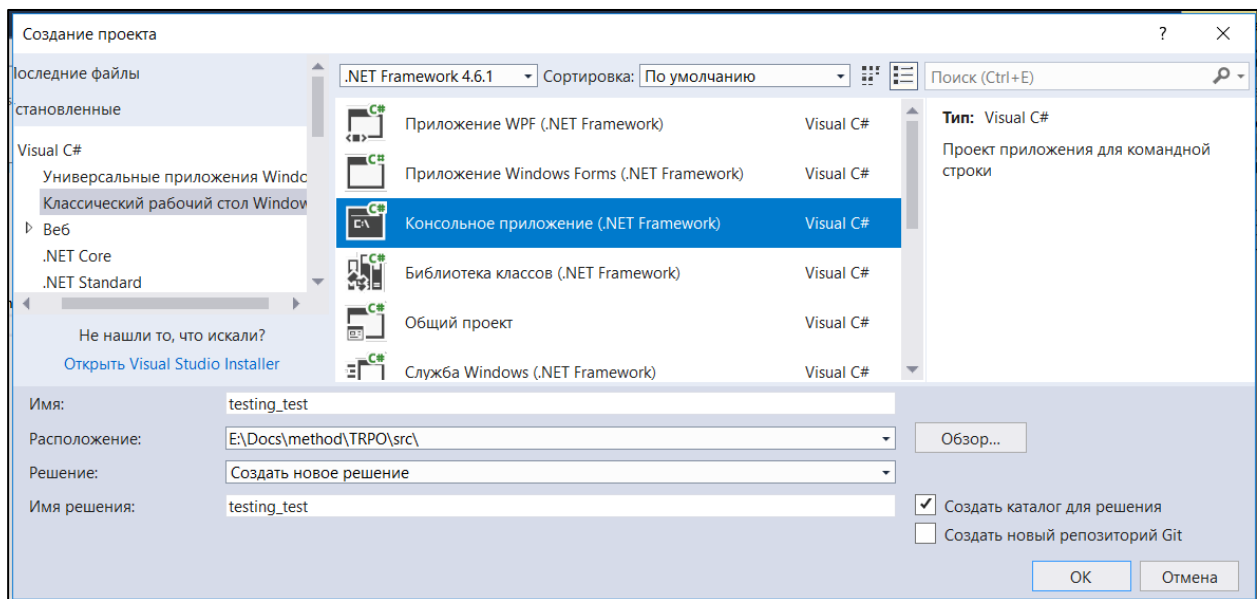


Рисунок 2.1 – Создать проект

Создайте проект типа “Консольное приложение”(рис.2.1).
Добавьте в проект новый класс (рис.2.2):

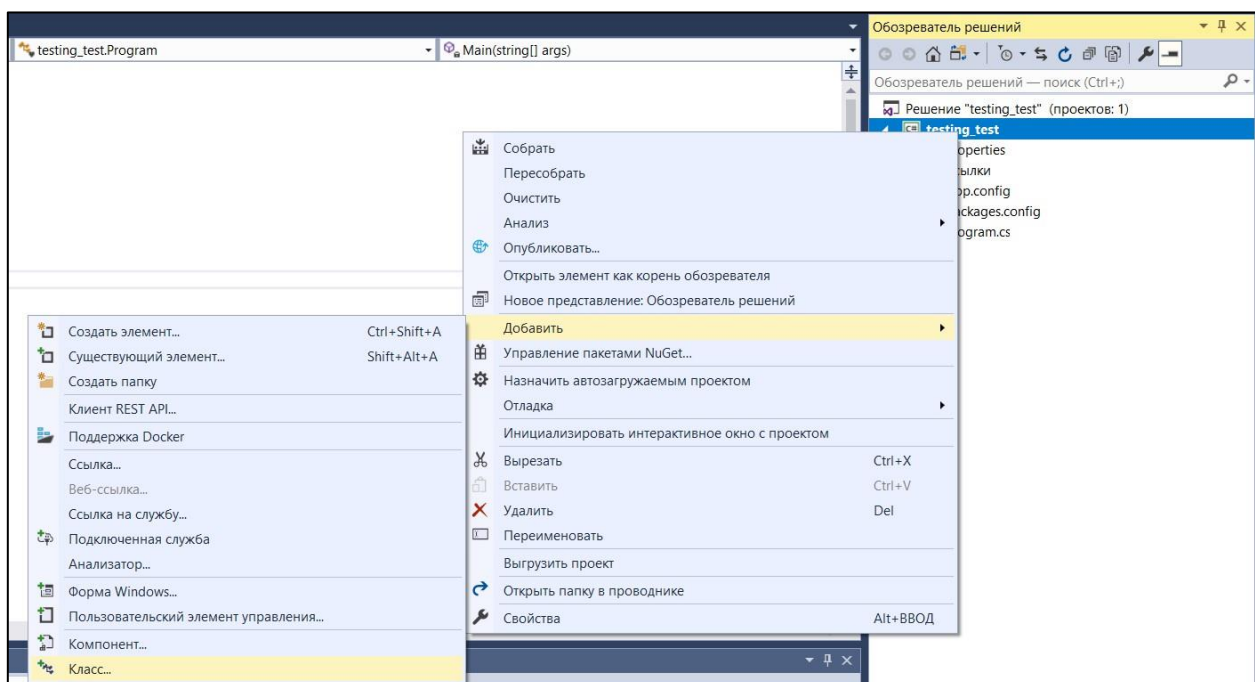


Рисунок 2.2 – Добавление в проект новый класс

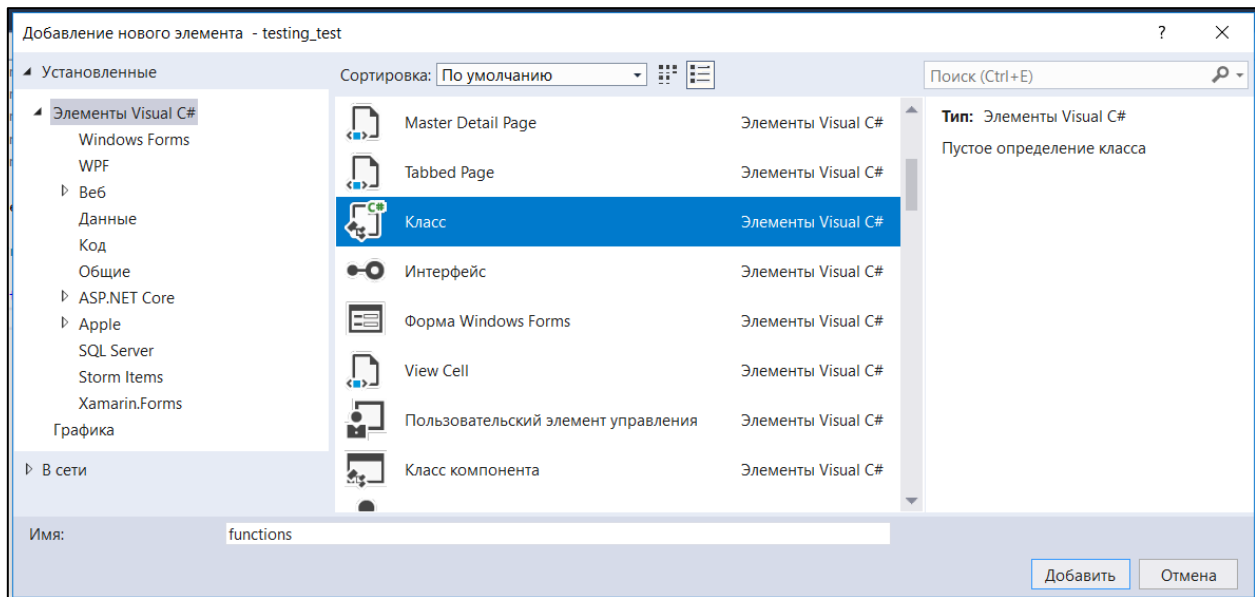


Рисунок 2.3 – Создаваемый класс

В создаваемом классе (рис.2.3) будут находиться функции для тестирования (рис.2.4):

Пример:

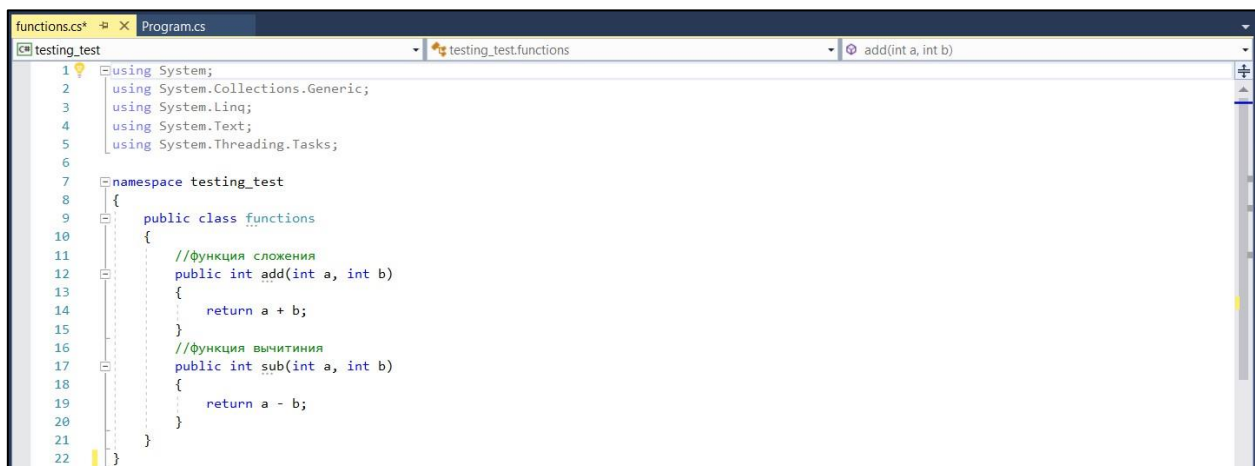


Рисунок 2.4 – Функция

Для того, чтобы добавить к проекту NUnit, нажмите правой кнопкой мыши на название проекта, а затем выберите “Управление пакетами NuGet...”:

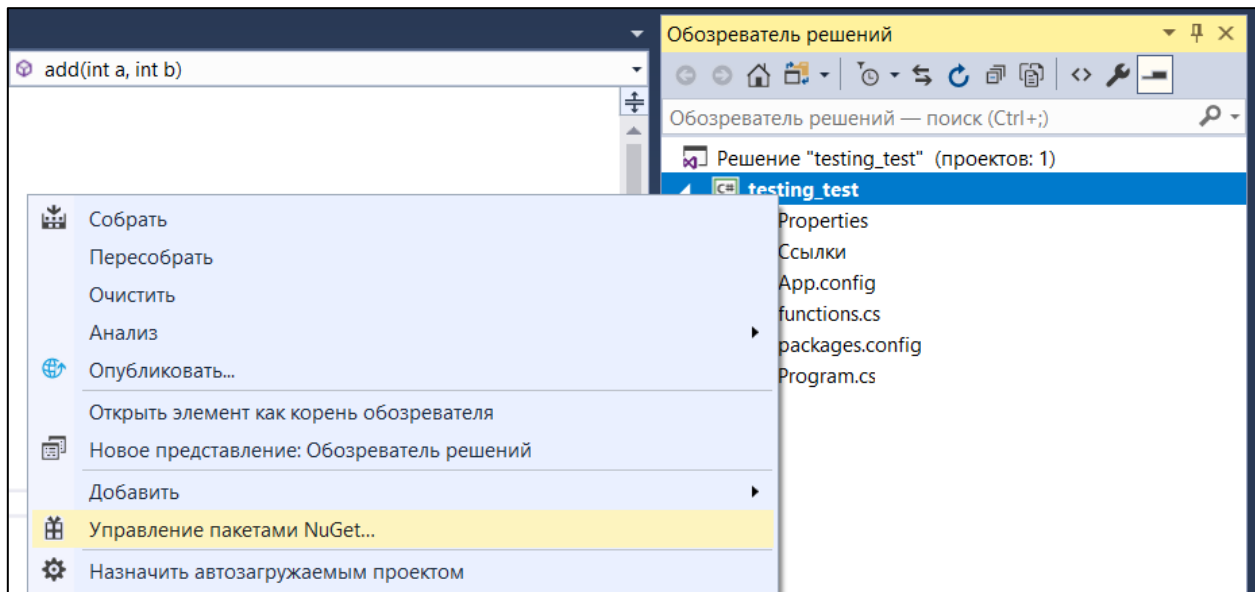


Рисунок 2.5 – Добавление пакета NUnit

В появившемся окне, выберите “Обзор”, затем из списка выберите NUnit и нажмите “Установить” (рис.2.5, рис.2.6, рис.2.7):

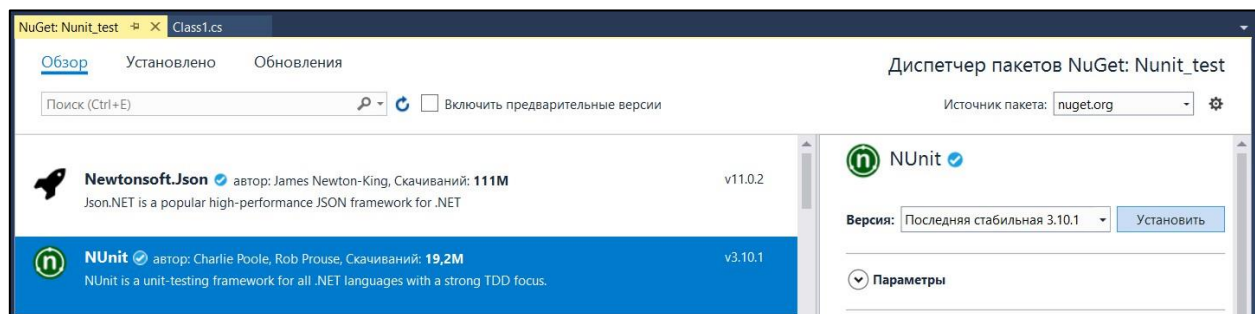


Рисунок 2.6 – Добавление пакета NUnit

После этого, перейдите в Средства -> Расширения и обновления...:

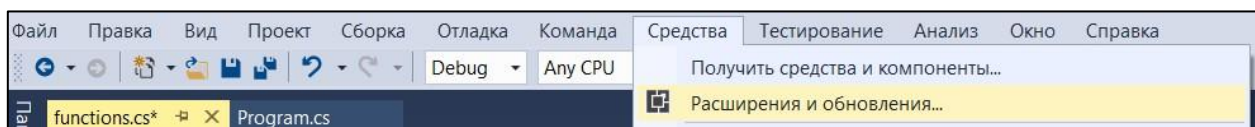


Рисунок 2.7 – Добавление пакета NUnit

Выберите раздел “В сети”, введите в строке поиска “nunit” и установите NUnit 3 Test Adapter:

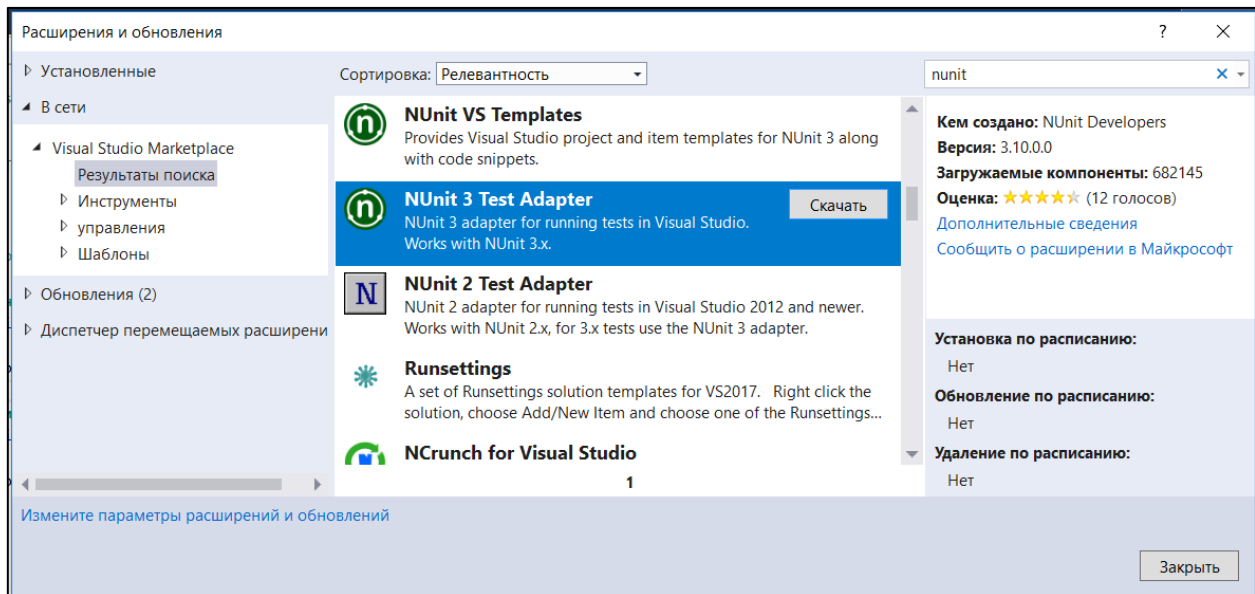


Рисунок 2.8 – Добавление нового класса к проекту

После этого, добавьте новый класс к проекту ” (рис.2.8) Желательно, что бы в названии класса присутствовало слово – test.

Описание тестов:

В новом классе, могут быть описаны тестовые сценарии (рис.2.9). Тестовый сценарий, это вызов функции или последовательность вызова функций, с проверкой возвращаемых значений. В случае если возвращаемые значения не соответствуют ожидаемым, тестовый сценарий считается не пройденным:

```
//подключение NUnit
using NUnit.Framework;

namespace testing_test
{
    //атрибут, указывающий на то, что это класс с тестами
    [TestFixture]
    class tasteCase
    {
        //атрибут, указывающий на то, что это тестовый метод
        [TestCase]
        //название тестового метода
        public void Add()
        {
            //создание класса, содержащего функции, которые будут тестированы
            functions f = new functions();
            //тестирование функции сложения
            //если результат вызова функции не равен 33, тест не будет пройден
            Assert.AreEqual(33, f.add(3, 30));
        }

        [TestCase]
        public void Sub()
        {
            functions f = new functions();
            //тестирование функции вычитания
            Assert.AreEqual(33, f.sub(3, 30));
        }
    }
}
```

Рисунок 2.9 – Класс с тестовым сценарием

После того, как описание тестовых сценариев будет завершено, запустить их можно через меню “Тестирование” -> “Выполнить” -> “Все тесты” (рис.2.10):

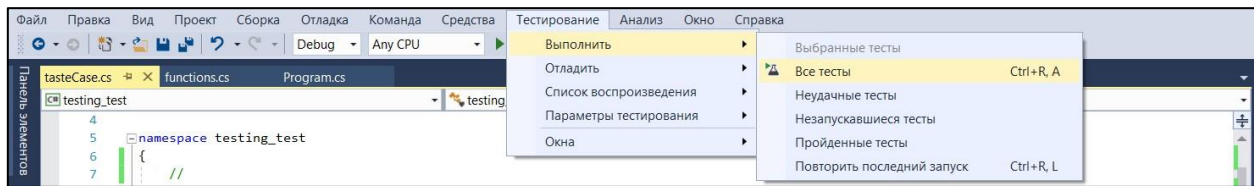


Рисунок 2.10 – Запуск тестов

Либо интерфейс “Обозреватель тестов”:

После запуска приведённых выше тестов, обозреватель будет выглядеть следующим образом (рис.2.11):

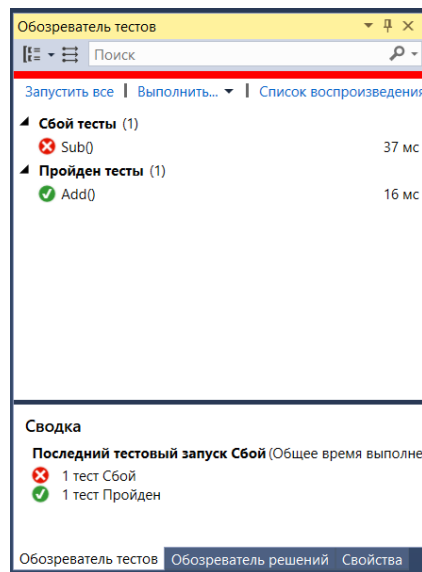


Рисунок 2.11 – Обозреватель тестов

Дополнительные функции NUnit:

В процессе работы над выполнением лабораторной работы вам могут понадобиться следующие функции тестирования:

Проверка функций на возникновение исключительных ситуаций (рис.2.12).

Предположим, существует функция вычисления остатка от деления:

```
public int ostatok(int a, int b)
{
    if (b <= 0) throw new ArgumentException("Делитель должен быть >= 0");

    return a % b;
}
```

Рисунок 2.12 – Проверка функции

В функции, описано условие, которое может привести к

возникновению исключительной ситуации.

Для того, что бы протестировать корректность срабатывания условия, можно использовать тест следующего вида (рис.2.13):

```
[TestCase]
public void Ostatok()
{
    //создание объекта, содержащего функцию
    functions f = new functions();

    //получение исключения
    var ex = Assert.Throws<ArgumentException>(() => f.ostatok(2, 0));
    //сравнение полученного сообщения с ожидаемым
    Assert.That(ex.Message, Is.EqualTo("Делитель должен быть >= 0"));

    //проверка выполняется успешно, если исключение не было сгенерировано
    Assert.DoesNotThrow(() => f.ostatok(2, 1));
}
```

Рисунок 2.13 – Тест

Проверка функций на возвращаемое состояние.

Например, для функции проверки числа на чётность:

```
[TestCase]
public void Chet()
{
    //создание объекта, содержащего функцию
    functions f = new functions();

    //проверка возвращаемого значения, в первом случае оно должно быть истинно
    //во втором ложно
    Assert.IsTrue(f.chet(4));
    Assert.IsFalse(f.chet(5));
}
```

Рисунок 2.13 – Тест проверки для функции

Помимо истинности, можно узнать является ли объект пустым, неопределённым и т.д. Полный список Assert функций (с примерами) можно посмотреть в официальном репозитории NUnit

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Реализовать набор функций:
2. Функция конвертации дюймов в сантиметры.
3. Функция проверки числа на чётность.
4. Функция выбора наибольшего значения из массива це-

лых чисел.

5. Функция вычисления остатка от деления.

В случае возникновения ошибки, функции должны возвращать код ошибки.

6. Написать набор тестовых сценариев для реализованных тестовых функций. Тестовые сценарии должны охватывать все ветви выполнения функций.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как создать модули-тестирования?
2. Что является пройденным и не пройденным тестированием?
3. Что из себя представляет Атрибут?

ПРАКТИЧЕСКАЯ РАБОТА №3. РАЗРАБОТКА ТЕСТ-КЕЙСА ДЛЯ УЧЕБНОЙ ПРОГРАММЫ

Цель: Разработать рабочую тестовую документацию для тестирования (тест-кейсы).

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Основным объектом изучения в данной работе являются наборы тестовых сценариев (тест-кейсов). Для удобства управления процессом тестирования и повышения структурированности документации применяют группировку тестовых сценариев в тест-сюиты.

Рабочая тестовая документация значительно улучшает качество последующего тестирования за счет анализа и детального планирования тестов. После завершения тестирования наличие тестовой документации позволяет оценить, насколько успешно были проведены все этапы тестирования, а для заказчика является подтверждением реального объема работ.

Рабочую тестовую документацию тестировщик может разрабатывать исключительно на основе спецификации еще до поставки программного обеспечения. В этом случае после поставки на тестирование версии программного продукта специалист по тестированию может сразу приступить к поиску дефектов.

Существуют следующие виды рабочей тестовой документации:

Check List.

Acceptance Sheet.

Test Survey.

Test Cases.

Основные факторы выбора тестовой документации – сложность бизнес-логики проекта, сроки проекта, размер команды и объем проекта.

На одном проекте могут комбинироваться несколько типов тестовой документации. Например, для всего проекта составлен Acceptance Sheet, но для наиболее сложных частей составлены Test Cases. Если какие-либо модули программного продукта будут подвергаться автоматизированному тестированию, то для та-

ких модулей в обязательном порядке составляются Test Cases.

Таблица 3.1 – Виды рабочей тестовой документации и их характеристика

Тип документации		Что описывают	Когда используют
Checklist		Вспомогательный тип документации, содержащий Список основных проверок.	Для типовой функциональности.
Acceptance Sheet		Перечень всех модулей и функций приложения, подлежащих проверке.	Небольшие (до 3 месяцев), простые по бизнес-логике проекты.
Test Survey		Перечень всех модулей и функций, а также конкретные проверки для них. Может содержать ожидаемый результат.	Средние или большие проекты с понятной бизнес-логикой.
Test Cases		Набор входных значений, пред-условий, пошаговое описание сложной бизнес-постусловия для каждой проверки. Всегда содержит ожидаемый результат.	Большие и долгосрочные проекты со сложной бизнес-логикой, проекты с большой командой.

Примеры фрагментов рабочей тестовой документации приведены в таблице 3.1.

При составлении рабочей тестовой документации необходимо указать номер тестируемой сборки, тип выполняемой тестовой активности, период времени тестирования, ФИО тестировщика, тестовое окружение (операционная система, браузер, др.).

Рабочая тестовая документация представляет собой перечень всех проверок для модулей/подмодулей приложения. В качестве одного модуля как правило выступает рабочее окно приложения, в качестве подмодулей – логически завершенные блоки этого окна.

Для каждого модуля в обязательном порядке выполняется тестирование GUI, а также общие функциональные проверки (General). Далее в рамках модуля в качестве функциональных проверок выступают действия над активными элементами пользовательского интерфейса (полями, кнопками, чекбоксами и т.д.). Степень детализации каждой из таких функциональных проверок зависит от выбранного типа тестовой документации (Acceptance Sheet, Test Survey, Test Cases). В частности, для Acceptance Sheet все активные элементы пользовательского интерфейса только перечисляют. Для Test Survey для каждого элемента приводят позитивные и негативные проверки, источником которых являются базовые проверки (в виде чеклиста) для соответствующих элементов GUI. Для Test Cases каждую из позитивных и негативных проверок описывают в виде последовательности шагов с указанием ожидаемого результата.

Для Test Survey, Test Cases напротив каждой проверки указывается глубина тестирования: Smoke. Для Acceptance Sheet в качестве глубины тестирования всегда указывается АТ в таблице 3.2.

Тестовый случай (Test Case) – совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или ее части. Атрибуты тест-кейса:

ID.

Дата.

Автор. Заголовок / описание.

Список действий, переводящих систему из одного состояния в другое, для получения результата, на основании которого мож-

но сделать вывод об удовлетворении реализации поставленным требованиям.

Выполняемые шаги.

Приоритет выполнения.

Предусловия. Список действий, которые приводят систему к состоянию, пригодному для проведения основной проверки. Либо список условий, выполнение которых говорит о том, что система находится в пригодном для проведения основного теста состоянии.

Постусловия. Список действий, переводящих систему в первоначальное состояние.

Дополнительные сведения.

Таблица 3.2 – Примеры рабочей тестовой документации

Checklist	Acceptance Sheet	Test Survey	Test Cases
Протестировать форму авторизации	Форма авторизации: GUI. General. Поле «Эл.адрес или телефон». Поле «Пароль». Кнопка «Войти». Чекбокс «Не выходить из системы». Ссылка «Забыли пароль».	Форма авторизации: GUI. General. Валидный эл.адрес + валидный пароль. Валидный телефон + валидный пароль. Валидный эл.адрес + невалидный пароль. Валидный телефон + невалидный пароль. Невалидный эл.адрес или телефон + валидный пароль. Невалидный эл.адрес или телефон + невалидный пароль. Запомнить данные: выйти из системы и зайти обратно. Ссылка «Забыли пароль».	Авторизация с помощью e-mail: Открыть страницу abc.com. Ввести в поле «Эл.адрес или телефон» e-mail abc@mail.ru . Ввести в поле «Пароль» пароль qwerty. Нажать на кнопку «Войти». Ожидаемый результат: пользователь переходит на свою домашнюю страницу.

Порядок создания тестовой документации определяется

условиями реализации программных систем. Если к началу разработки существовала документация, описывающая требования, то создание тестовой документации должно опираться на эти требования и начинаться непосредственно после их выявления. Выполнение лабораторной работы не предусматривает использование какого-либо специального программного обеспечения для учета документации, будет достаточно любого текстового редактора. Но системы ведения тестовой документации могут значительно сокращать трудоемкость для тестировщика, поскольку обладают возможностями ведения нескольких типов тестовой документации и выполнения манипуляций с ней в рамках процедур тестирования. Примеры тестовой документации приведены в таблицах 3.3 и 3.4.

Таблица 3.3 – Примеры оформления списка тест-сьютов

ID	Автор	Приоритет	Заголовок	Список тест-кейсов
1	user	1	Модуль пользователя	Авторизация Регистрация Редактирование профиля
2	user	1	Модуль товаров	Получение новинок Получение товаров в категории Получение товара
3	user	1	Модуль корзины	Добавление товара в корзину Увеличение количества товара в корзине Уменьшение количества товара в корзине
4	user	1	Модуль оформления заказа	Оформление заказа Оформление заказа Оформление заказа

Таблица 3.4. Примеры оформления списка тест-кейсов

ID	Описание (Тип)	Предусловия	Шаги	Ожидаемый результат
1.1	Авторизация (позитивный)	Пользователь находится на странице входа в личный кабинет Пользователь был ранее зарегистрирован в системе	Ввести в поля «Логин» и «Пароль» логин и пароль пользователя Нажать кнопку «Войти»	Пользователь перенаправляется на страницу личного кабинета
1.2	Регистрация (негативный)	Пользователь находится на странице регистрации Пользователь был ранее зарегистрирован в системе	Ввести в обязательные поля «Имя», «Фамилия», «Пароль», «Подтвердите пароль», «Электронная почта» данные. Также можно ввести данные в поля «Номер телефона» и «Адрес». Нажать кнопку «Зарегистрироваться»	Пользователь на странице регистрации получает сообщение «Пользователь с таким email существует»
1.3	Редактирование профиля (негативный)	Пользователь находится на странице личного кабинета	Нажать на пункт «Профиль» Получить страницу редактирования профиля Нажать на кнопку	Пользователю отображается сообщение «Пароли не совпадают»

			«Изменить» Ввести в поля «Пароль» и «Подтвердите пароль» раз- ные значения Нажать на кнопку «Сохранить»	
2.1	Получение новинок (Позитив- ный)	Пользователь находится на од- ной из страниц магазина	Пользователь нажимает на название ин- тернет- магазина, рас- положенного в шапке сайта	Пользователю отображается главная стра- ница магазина со списком но- винок
2.2	Получение товаров в категории (Позитив- ный)	Пользователь находится на одной из следующих стра- ниц: на главной странице, на странице «Корзина», на странице входа в личный кабинет, на странице реги- страции, на стра- нице каталога товаров, на стра- нице товара	Нажать на пункт меню категорий	Пользователь перенаправля- ется на стра- ницу катего- рий На странице категорий отображается заголовок, совпадающий с названием ранее выбран- ного пункта меню

2.3	Получение товара (Позитивный)	Пользователь находится на одной из следующих страниц: на главной странице, на странице «Корзина», на странице категорий На данной странице отображается товар	Нажать на кнопку «Товар»	Пользователь перенаправляется на страницу товара Информация о товаре на странице товара должна совпадать с информацией на странице, с которой был осуществлен переход
3.1	Добавление товара в корзину (Позитивный)	Пользователь находится на одной из следующих страниц: на главной странице, на странице категорий, на странице товара На данной странице отображается товар	Нажать на кнопку «Добавить в корзину»	Страница, на которой находится пользователь, обновляется Пользователю отображается сообщение «Товар в корзину успешно добавлен» При переходе на страницу «Корзина» отображается добавленный товар
3.2	Увеличение количества товара в корзине (Позитивный)	Пользователь находится на странице «Корзина»	Нажать на кнопку «+»	В колонке «Кол-во» в строке товара увеличивается значение на 1

3.3	Уменьшение количества товара в корзине (Негативный)	Пользователь находится на странице «Корзина» Значение в колонке «Кол-во» в строке товара равно 1	Нажать на кнопку «-»	Значение в колонке «Кол-во» в строке товара не изменяется
-----	---	---	----------------------	---

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Выбрать веб-приложение для тестирования, согласовать с преподавателем.

2. Сформировать отчет с тестовой документацией: список тест-кейсов и тест-сюетов в соответствии с требованиями к атрибутам документации. Не нужно рассматривать функции авторизации / регистрации, поскольку они одинаковые для программных систем.

3. Требования к наличию тестов: smoke-тесты, тестирование навигации, тестирование ввода данных (как минимум две формы),

4. Тестирование бизнес-логики. Обязательно сделать как позитивные, так и негативные тест-кейсы.

5. В отчет по лабораторной работе включить:

6. Цель работы.

7. Описание тестируемого приложения.

8. Тестовую документацию.

9. Выводы по работе.

10. Список использованных источников.

11. Оформить и защитить отчет.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое тест-кейс?
2. Назовите правила составления тест-кейсов.
3. Что такое наборы тест-кейсов?
4. Опишите жизненный цикл тест-кейса.
5. Назовите атрибуты (поля) тест-кейса.
6. Что такое чек-лист?
7. Назовите свойства чек-листа.
8. Назовите атрибуты (поля) чек-листа.
9. Приведите пример негативных тест-кейсов для трех видов тестирования.
10. Перечислите требования к тест-сютам.
11. Перечислите требования к тест-кейсам.

ПРАКТИЧЕСКАЯ РАБОТА №4. СОСТАВЛЕНИЕ ПЛАНА ТЕСТИРОВАНИЯ УЧЕБНОЙ ПРОГРАММЫ

Цель: Разработать рабочую тестовую документацию для тестирования (тест-план).

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

К тестовой документации относятся: план тестирования (тест-план), набор тестов (тест-сьют), тестовый случай (тест-кейс), отчет о дефектах (баг-репорт). Тест-план описывает следующие ключевые элементы процесса тестирования:

1. что надо тестировать,
2. режимы тестирования,
3. график тестирования,
4. критерии начала тестирования,
5. критерии окончания тестирования,
6. окружение тестируемой системы,
7. возможные риски и пути их решения.

Более детально требования к содержанию плана тестирования изложены в ГОСТ Р 56922– 2016/ISO/IEC/IEEE 29119- 3:2013

Тест-план Структура тест-плана может соответствовать структуре тест требований или следовать логике внешнего поведения системы. Каждый пункт тест-плана описывает, как производится проверка правильности функционирования программной реализации, и содержит: ссылку на требования, которое проверяется этим пунктом; конкретное входное воздействие на программу (значения входных данных); ожидаемую реакцию программы (тексты сообщений, значения результатов); описание последовательности действий, необходимых для выполнения пунктов тест-плана. Возможные формы подготовки тест-планов:

1. В виде текстовых документов, в которых отдельные разделы представляют собой описания тестовых примеров, каждый пример включает в себя перечисление последовательности действий, которые необходимо выполнить – сценария теста, а также ожидаемые отклики системы на эти действия.

2. Для автоматизированного тестирования сценарий может записываться на формальном языке.

Атрибутами тест-плана являются:
Заголовок / версия / Автор,
Техническое задание на продукт или иная документация,
Задачи / функциональность, которая должна быть протестирована,
Виды проводимого тестирования,
Список тестовой документации (тест-кейсы, тест-сьюты),
Список инструментов тестирования
Сервер (или иное расположение тестируемой программной системы),
Ответственные лица (ФИО/ Должность / занятие),
График тестирования,
Оценка риска,
Примечание.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Выбрать веб-приложение для тестирования, согласовать с преподавателем.
2. Сформировать отчет с тестовой документацией: тест-план в соответствии с требованиями к атрибутам документации. Не нужно рассматривать функции авторизации / регистрации, поскольку они одинаковые для программных систем.
3. В отчет по лабораторной работе включить:
 - a. Цель работы.
 - b. Описание тестируемого приложения.
 - c. Тестовую документацию.
 - d. Выводы по работе.
 - e. Список использованных источников.
 - f. Оформить и защитить отчет.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Перечислите требования к тест-плану.
2. Какова связь этапа жизненного цикла разработки программного обеспечения и вида тестовой документации?

ПРАКТИЧЕСКАЯ РАБОТА №5. ОСНОВЫ РАБОТЫ В СИСТЕМЕ ОТСЛЕЖИВАНИЯ ОШИБОК

Цель работы – протестировать web-приложение и описать найденные дефекты (ошибки). Задачами работы являются:

1. Изучить теоретические сведения.
2. Выполнить практическое задание по лабораторной работе.
3. Оформить отчёт и ответить на контрольные вопросы.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Дефекты, обнаруженные тестировщиком, должны быть корректно и понятно описаны, чтобы разработчик смог воспроизвести данный дефект и устранить его. Описание каждого дефекта сохраняется в специализированной – багтрэкинговой – системе (например, JIRA, Bugzilla, Mantis, Redmine и др.) или в предварительно созданном в программной среде Microsoft Excel файле (пример приведен в таблице 5.1).

Таблица 5.1 – Пример описания дефекта

№	Название дефекта	Важность	Алгоритм воспроизведения	Фактический результат	Ожидаемый результат	Приложение	Примечание
39	Администраторская часть: Файлы: Выбор файла: Ссылка "Скачать файл": Администратор не имеет возможности скачать загруженные студентом файлы.	Critical	Шаги по воспроизведению: Входим на web сайт ... Проходим авторизацию: admin/admin. Выбираем в меню категорию "Файлы".	Переход к странице "HTTP Status 404".	Происходит скачивание выбранного файла.	39.png	REQ-26

			4.Выбираем подкатегорию меню "Выбор файла". 5.Выбираем в поле "Обзор файла" любой доступный файл. 6.Нажимаем на ссылку "Скачать файл".				
--	--	--	--	--	--	--	--

Описание дефекта включает следующие обязательные поля:

Headline – название дефекта.

Severity – степень критичности (важность дефекта).

Description – алгоритм воспроизведения.

Result – фактический результат.

Expected result – ожидаемый результат.

Attachment – прикрепленные файлы (приложение).

В багтрэкингвых системах для каждого дефекта автоматически генерируется его уникальный номер, в случае использования Microsoft Excel номер дефекту необходимо присваивать вручную.

Требование спецификации, которое нарушает обнаруженный дефект, можно дополнительно вынести в примечание.

Дополнительно в описании дефекта может быть указана Priority – степень срочности исправления дефекта разработчиком.

Рассмотрим подробно каждую категорию описания дефекта.

Headline (название дефекта). Цель составления заголовка дефекта – предоставить краткую и в тоже время понятную информацию о том, где, что и в результате чего произошло. Характеристиками качественного заголовка являются краткость, информативность, точная идентификация проблемы.

Заголовок дефекта должен отвечать на три вопроса:

Где? В каком месте интерфейса пользователя находится проблема.

В данной части заголовка следует также дополнительно указать особенности теста, если это поможет разобраться в проблеме (версия операционной системы, браузера, сторонних приложений, которые имеют отношение к тестируемому программному средству).

Что? Что происходит или не происходит согласно спецификации или представлению о нормальной работе программного продукта. При этом необходимо указывать на наличие проблемы, а не на ее содержание (его указывают в описании). Если содержание проблемы варьируется, все известные варианты указываются в описании.

Когда (при каких условиях)? В какой момент работы программы или по наступлению какого события проблема проявляется.

Severity (степень критичности). Степень критичности (серьезности, важности) показывает степень ущерба, который наносится проекту существованием дефекта. В общем случае выделяют следующие градации критичности дефектов (таблица 5.2): Critical (критический), Major (значительный), Average (средней значимости), Minor (незначительный), Enhancement (предложение по улучшению).

Таблица 5.1 – Степени критичности дефектов

Severity	Описание	Примеры
Critical (критический)	Функциональная ошибка, которая блокирует работу части функционала или всего приложения. Функциональная ошибка, которая нарушает ключевую (с точки зрения конечного пользователя или бизнеса заказчика) функциональность	Заблокирована вкладка категория меню). Неправильно подсчитывается итоговая сумма при вводе скидочного промокода. Раскрывается конфиденциальная информация.

	приложения.	
Major (значительный)	Функциональная ошибка, которая нарушает нормальную работу приложения, но не блокирует работу части функционала в целом.	Невозможно загрузить видео- файлы на персональной страничке. Не работает система e-mail нотификации. Не работает интеграция с социальными сетями. Необходимо перезапускать приложение при выполнении типичных сценариев работы
Average (средней значимости)	Не очень важная функциональная ошибка. Критичные дефекты GUI.	Не работает сортировка. «Уехал» текст за пределы окна.
Minor (незначительный)	Редко встречающиеся незначительные функциональные дефекты. 90 % дефектов GUI.	Введены необязательные для заполнения поля, которых нет в спецификации. Грамматические, пунктуационные ошибки.
Enhancement (предложение по улучшению)	Функциональные предложения, советы по улучшению дизайна (оформления), навигации и др. Такой дефект является необязательным для исправления.	Добавить кнопку «Наверх» на длинных формах. Увеличить размер шрифта.

Правильно оформить баг-репорт

Для начала нужно убедиться, что найденный баг ещё не был оформлен. Следует провести поиск его в соответствующем проекте по всем подходящим ключевым словам и\или полям. Если баг уже есть, следует обновить его описание.

Если баг не найден – нажимаем на кнопку создания бага. Не стоит забывать важное правило: один дефект – один баг в треке-ре.

Далее нужно постараться кратко описать, что не работает – это и будет заголовок баг-репорта.

После этого перейти к подробному описанию бага: указать шаги к воспроизведению.

Указать ожидаемый результат. Можно добавить ссылку на спецификацию.

Указать полученный результат.

Указать версию ПО, также указать версию окружения.

Если необходимо, приложить соответствующие артефакты: логи, скриншоты, дампы и т.д.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Выбрать объект реального мира (например, холодильник, блендер, лифт и др.), выделить в нем модули.

2. Разработать 20 и более тестовых проверок для выбранного объекта реального мира с указанием тестируемого модуля и глубины тестового покрытия.

3. Сформулировать по два возможных дефекта на каждый уровень Severity (Critical, Major, Average, Minor, Enhancement) для выбранного объекта реального мира.

4. Описать по одному дефекту на каждый уровень Severity (Critical, Major, Average, Minor, Enhancement) для выбранного объекта реального мира.

5. Протестировать web приложение в соответствии с составленной ранее тестовой документацией.

Описать все найденные дефекты в отчете о дефектах в среде Microsoft Excel и Jira.

(Зайдите на сайт проекта и зарегистрируйтесь с помощью электронной почты Microsoft или Google-аккаунта.

После регистрации придумайте имя проекта. Система реко-

мендует выбирать простое и понятное, например, название компании, чтобы никто из команды его не забыл. Но нужно уникальное имя – некоторые уже могут быть заняты.

После регистрации инструменты для управления будут доступны)

6. В отчете о дефектах указать номер тестируемой сборки, название приложения, период времени тестирования, ФИО тестирующего, тестовое окружение (операционная система, браузер).

7. Для каждого дефекта указать его порядковый номер, заголовок, важность, алгоритм воспроизведения, фактический результат, ожидаемый результат, приложение, примечание.

8. Для каждого дефекта обязательно сделать скриншоты.

9. В рабочую тестовую документацию внести результаты тестирования с указанием напротив соответствующей проверки степени критичности обнаруженного дефекта, его номера и заголовка.

10. Оформить отчет и защитить лабораторную работу.

Содержание отчета:

1. Цель работы.
2. Отчет о результатах тестирования выбранного объекта реального мира с перечислением тестовых проверок, сформулированных дефектов на каждый уровень Severity, описания дефектов.
3. Отчет о найденных дефектах web приложения.
4. Рабочая тестовая документация с внесенными дефектами web приложения.
5. Выводы по работе.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое отчет о дефекте?
2. Что такое ожидаемый результат?
3. Что такое фактический результат?
4. Назовите атрибуты (поля) отчета о дефекте.
5. Какие системы баг-трекинга вы знаете?

ПРАКТИЧЕСКАЯ РАБОТА №6. ОСНОВЫ РАБОТЫ В СИСТЕМЕ SELENIUM

Цель работы – познакомить с инструментом Selenium, его преимуществами и функциональными возможностями, а также показать, как создать тест.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Это бесплатное программное обеспечение с открытым исходным кодом, один из самых популярных инструментов автоматизации. Selenium используется для написания функциональных тестов без знания языка программирования тест-скрипта (при помощи расширения Selenium IDE). Он также предоставляет тестовый предметно ориентированный язык (DSL) для написания тестов на ряде популярных языков программирования, среди которых JavaScript (Node.js), C#, Groovy, Java, Perl, PHP, Python, Ruby и Scala. Selenium поддерживает браузеры Mozilla Firefox, Google Chrome, Safari и Internet Explorer.

Selenium – это набор программ с открытым исходным кодом, которые применяют для тестирования веб-приложений и администрирования сайтов локально и в сети. Программы Selenium позволяют автоматизировать действия браузера. Среди программ проекта:

Selenium IDE – плагин для браузера Firefox для записи действий пользователя.

Selenium RC – устаревшая библиотека для управления браузерами.

Selenium WebDriver – библиотека для управления браузерами.

Selenium Grid – кластер Selenium-серверов для управления браузерами на разных компьютерах в сети.

Selenium Server – сервер для управления браузером удаленно по сети.

Это самый популярный инструмент из семейства Selenium, который предоставляет пользователю готовый API, позволяющий взаимодействовать с браузером. Как клиент интернет-браузера, используя протокол HTTP, он отправляет JSON-файл, созданный

на одном из языков программирования. Затем код осуществляет навигацию по браузеру с помощью специального драйвера. Кроме того, WebDriver поддерживает тесты в мобильной версии.

Большое преимущество Selenium WebDriver в том, что при хорошо подобранных сценариях и автоматизированном обслуживании тестов он позволяет регулярно проверять в системе области с высоким риском появления дефектов. Регулярное повторное и регрессионное тестирование позволяет проверить корректность работы наиболее важных функций приложения. Кроме того, используя объектную модель Page Object Model, вы можете легко вносить исправления в тестовые сценарии.

Работа с Selenium WebDriver

Для начала работы с Selenium WebDriver необходимо установить IDE, то есть интегрированную среду программирования, например, Rider или Visual Studio.

После установки вы можете начать работу над первым тестом с помощью Selenium WebDriver (мы будем работать в среде Rider).

1. Создание нового *Unit Test* -проекта (рис. 6.1). с использованием языка C#.

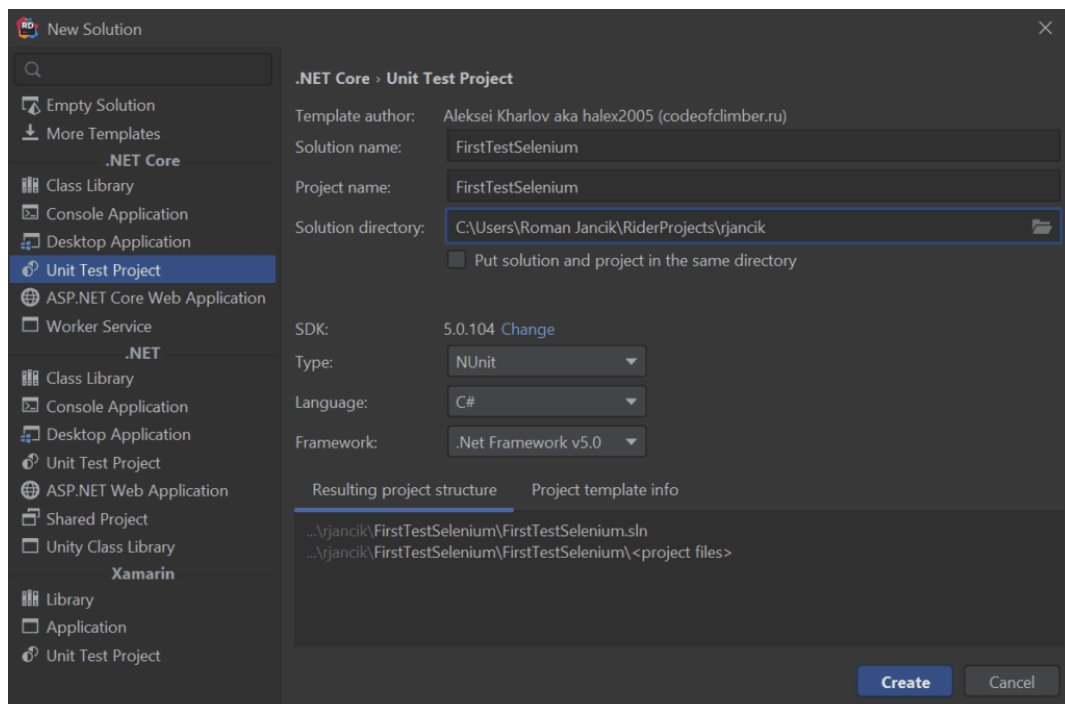


Рисунок 6.1 – Создание нового *Unit Test* -проекта

2. Загрузка *NuGet*-пакетов (рис. 6.2).

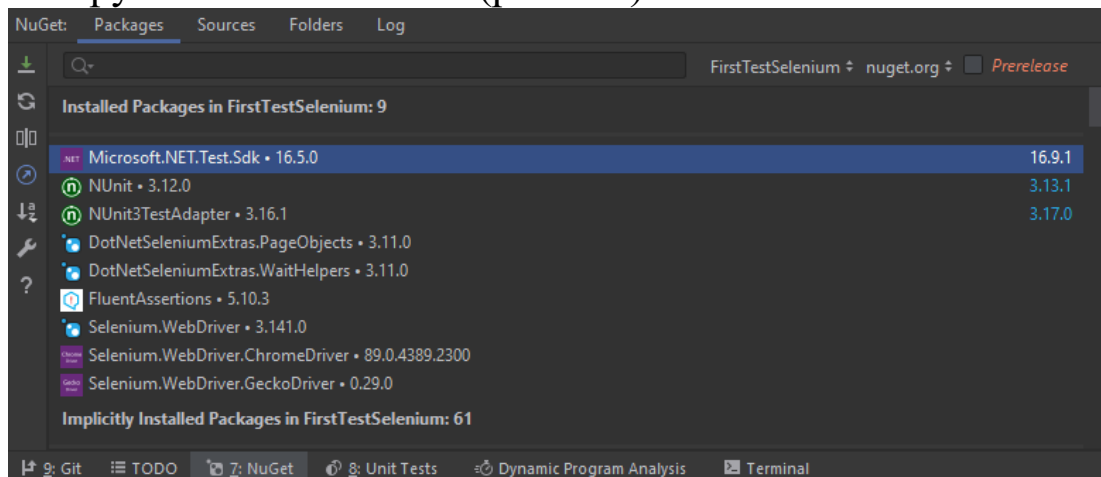


Рисунок 6.2 – Загрузка *NuGet*-пакетов

3. Создание класса FirstTestSelenium (рис. 6.3) и определение полей `_driver` и `_wait`.

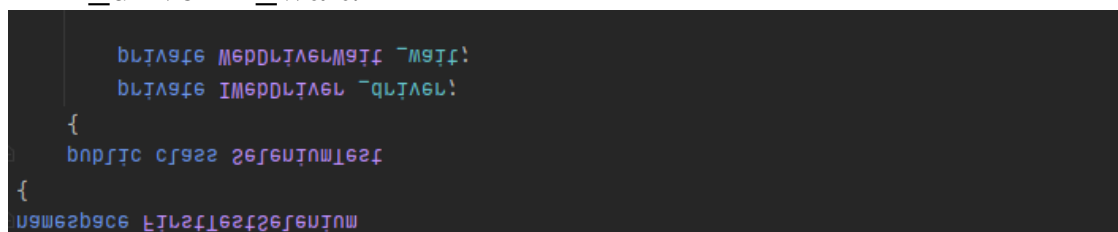


Рисунок 6.3 – Создание класса FirstTestSelenium

4. Создание метода Setup() (рис. 6.4), который, благодаря атрибуту [SetUp], будет выполняться перед каждым тестом. В этом методе `_driver` и `_wait` должны быть инициализированы, а окно браузера максимизировано через `Window.Maximize()`.

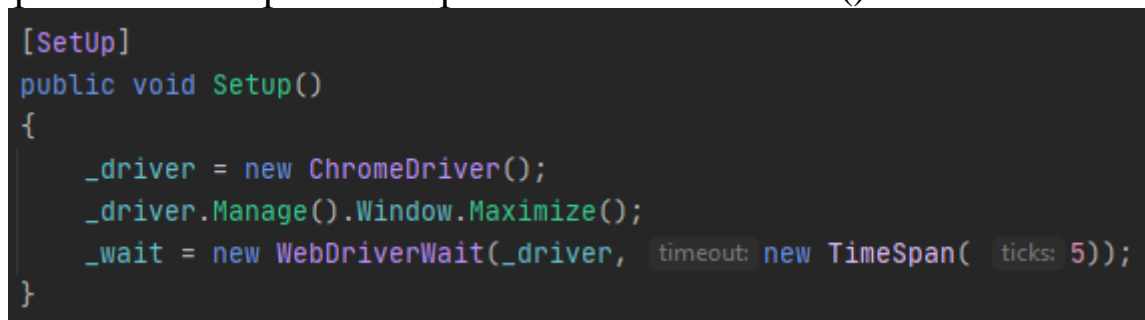


Рисунок 6.4 – Создание метода Setup()

5. Создание тестового метода с атрибутом [Test].

Внутри метода (рис. 6.5), дающего команду драйверу загрузить указанный сайт, нужно найти элемент и присвоить его переменной `actualLogo`.


```
[Test]
public void When_I_go_to_iteo_website_logo_is_visible()
{
    _driver.Navigate().GoToUrl("https://iteo.com/");
    var actualLogo :IWebElement = _wait.Until(ExpectedConditions.ElementIsVisible( locator: By.ClassName("pageHeader__logo")));

    actualLogo.GetAttribute("href").Should().Be("https://iteo.com/");
}
```

Рисунок 6.5 – Создание тестового метода с атрибутом

Чтобы тест был полноценным, необходимо использовать утверждение, чтобы убедиться в том, что код нашел нужный нам элемент.

6. Последний элемент теста – создание метода с атрибутом [TearDown] (рис. 6.6), который будет запускаться после каждого выполненного теста, закрывать окно браузера и завершать сессию WebDriver.

```
[TearDown]
public void TearDown()
{
    _driver.Quit();
}
```

Рисунок 6.6 – Создание тестового метода с атрибутом

7. Следующее изображение показывает, что тест был выполнен положительно (рис. 6.7).

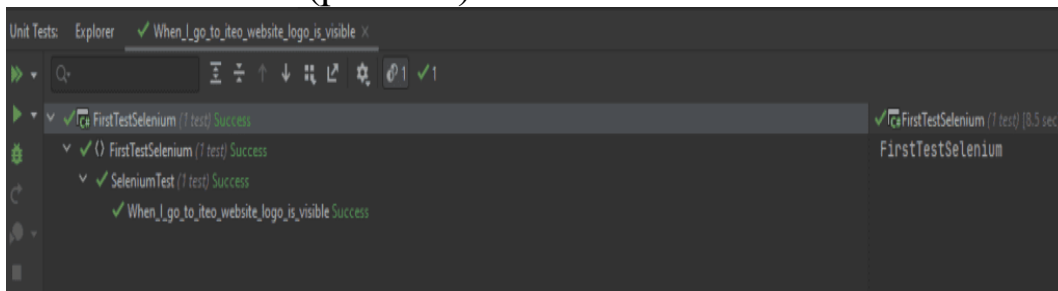


Рисунок 6.7 – Выполнение теста

`_driver` выполнил свою работу, утверждение было верным, и вы можете сказать, что операция прошла успешно.

8. Весь класс вместе с выполненным тестом выглядит так, как показано ниже (необходимо подчеркнуть, что это простейший тест, использующий Page Object Pattern) (рис. 6.8):

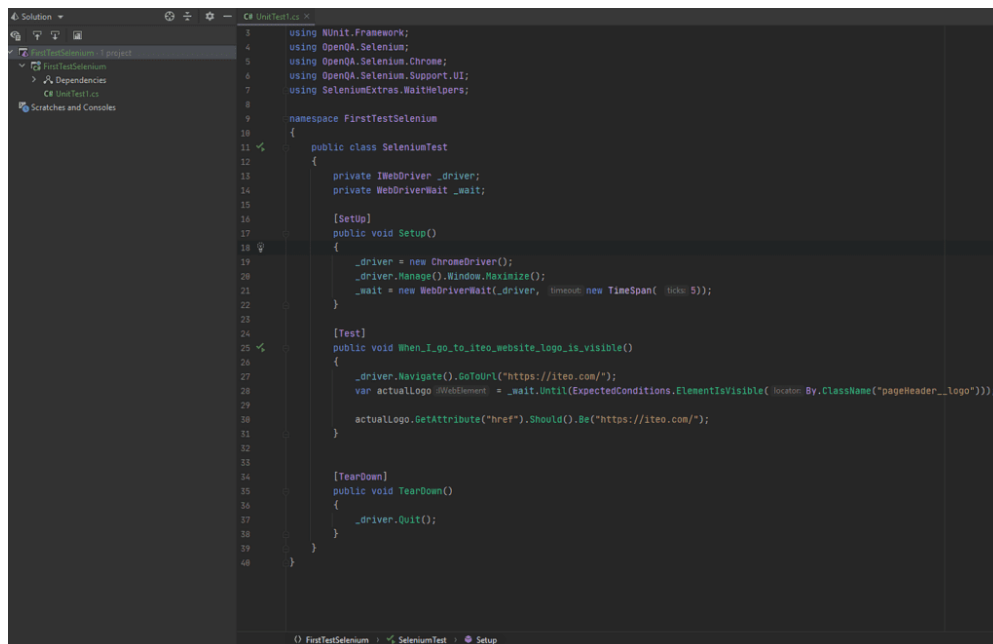


Рисунок 6.8 – Выполнение теста с классом

Стоит отметить, что тестирование проводилось в браузере Chrome, и на оборудовании с системой Windows.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Подготовить текстовый файл words.txt с 10 начальными словами.

Создать консольное приложение на языке C#, выполняющее следующие функции:

1)Открывает браузер и переходит на сайт <https://www.bukvarix.com/>(далее Букварикс)

2)По очереди берет слово из файла (пока слова не закончились) и вбивает в поле для поиска на сайте Букварикс и нажимает кнопку «Найти».

3)После загрузки страницы с данными перейти в конце страницы и нажать кнопку «Скачать (файл .csv)».

4. Написать тест для проверки загрузки страницы сайта

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое тестирование автоматизации? Каковы преимущества автоматизации тестирования?
2. Каковы ограничения Selenium?
3. Какие существуют типы локаторов в Selenium?
4. В чем разница между командами assert и verify?

ПРАКТИЧЕСКАЯ РАБОТА №7. АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ ПРИЛОЖЕНИЯ В СИСТЕМЕ SELENIUM

Целью данной работы является знакомство с автоматизацией функционального тестирования при помощи C#, NUnit и Selenium WebDriver.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Создание и настройка решения:

Разрабатываемый проект, по сути, не является программным приложением, поэтому, при создании решения, следует выбрать “Библиотека классов”(рис. 7.1).:

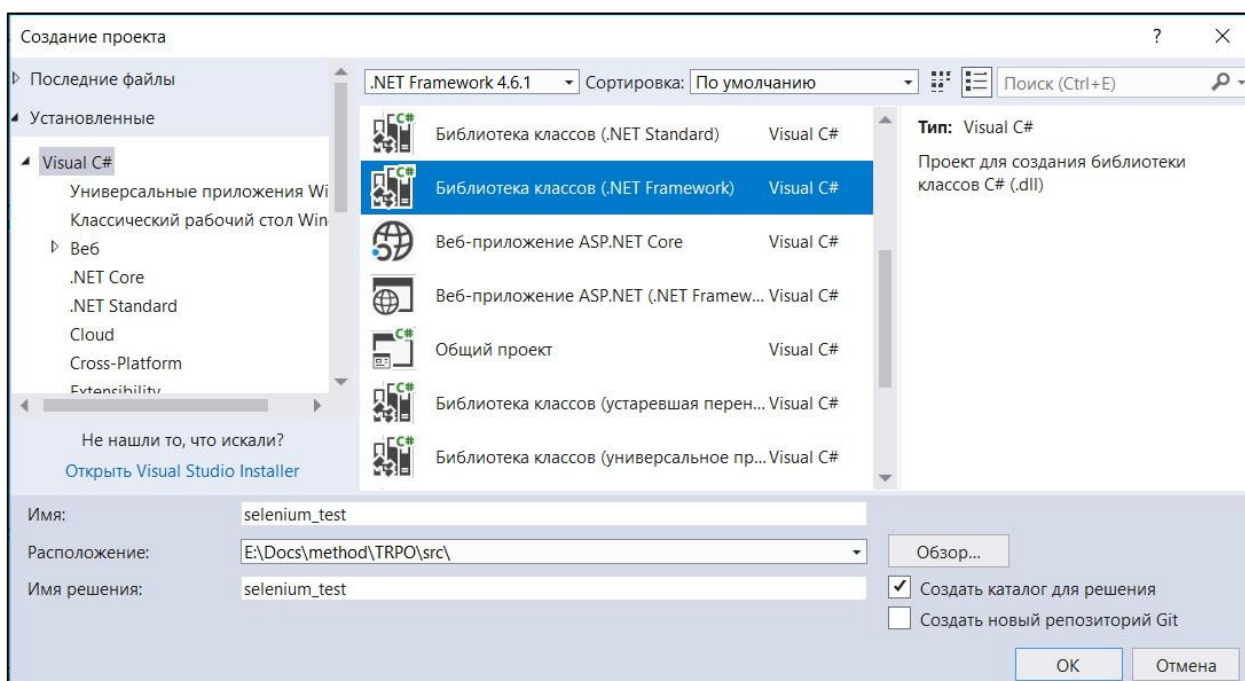


Рисунок 7.1 – Создание библиотеки классов

После создания проекта, подключите к нему NUnit при помощи пакетов NuGet (рис.7.2):

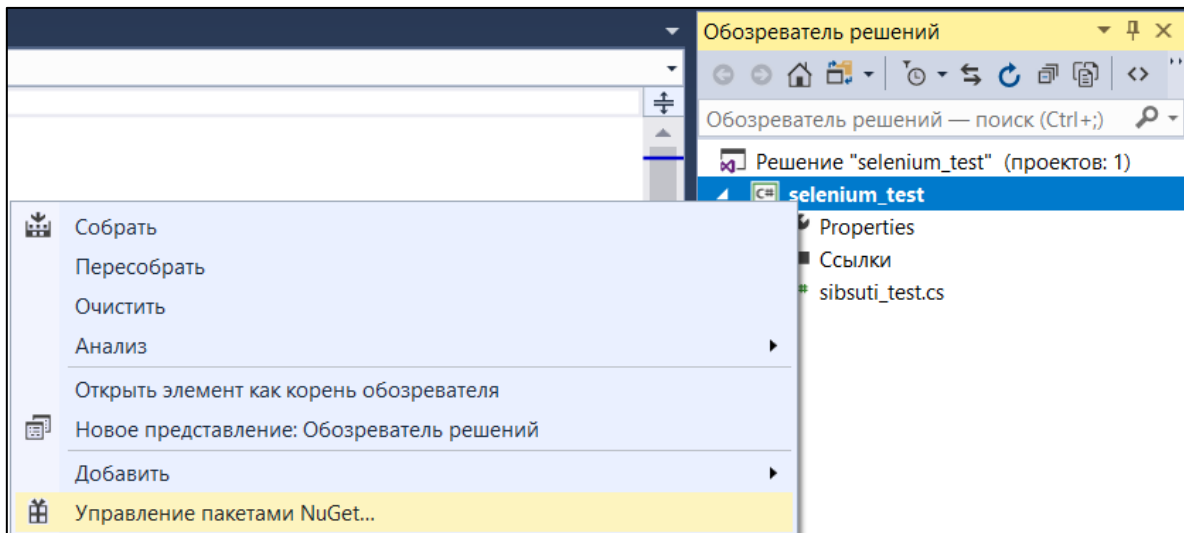


Рисунок 7.2 – Добавление пакетов NuGet

Выберите “Обзор” и введите в строке поиска NUnit (рис. 7.3).

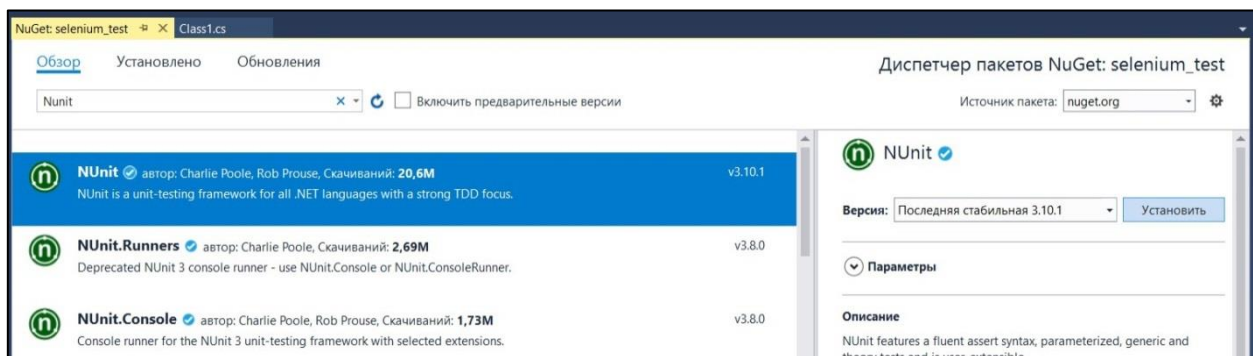


Рисунок 7.3 – Добавление пакетов

Выберите NUnit и нажмите “Установить”
Затем, введите в строке поиска “Selenium” и установите Selenium.WebDriver (рис. 7.4, 7.5):

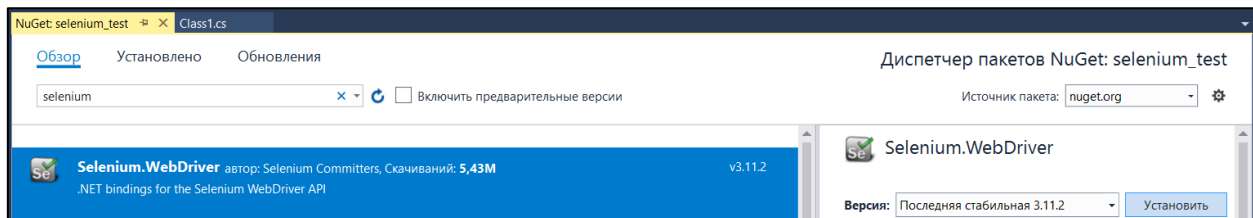


Рисунок 7.4 – Установка Selenium.WebDriver

После чего, на той же странице, выберите и установите драйвер для вашего браузера:

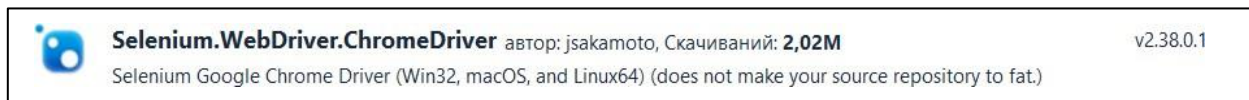


Рисунок 7.5 – Установка Selenium.WebDriver

В примерах используется Google Chrome, однако можно использовать любой браузер, для которого имеется WebDriver.

Подключение библиотек и написание простого теста:

Для того, чтобы получить доступ к функциям тестирования, подключите следующие пространства имён (рис. 7.6):

```
//инструменты для создания тестов
using NUnit.Framework;
//инструменты функционального тестирования сайтов
using OpenQA.Selenium;
//инструменты для работы с браузером Google Chrome
using OpenQA.Selenium.Chrome;
```

Рисунок 7.6 – Подключение пространства имён

В качестве простого теста, можно использовать проверку заголовка веб страницы (рис. 7.7):

```
namespace selenium_test
{
    //создание набора тестов
    [TestFixture]
    public class sibsuti_test
    {
        //создание и инициализация ссылки на браузер
        IWebDriver webDriver = new ChromeDriver();

        //тестовый случай
        [TestCase]
        public void maintTitle()
        {
            //установка адреса сайта, автоматически открывает окно Google Chrome и переходит по адресу
            webDriver.Url = "https://sibsutis.ru/";
            //предположение, что заголовок сайта равен указанной строке
            Assert.AreEqual("Сибирский государственный университет телекоммуникаций и информатики", webDriver.Title);
            //закрытие окна браузера
            webDriver.Close();
        }

        //метод, который вызывается после выполнения всех тестов
        [TearDown]
        public void testEnd()
        {
            //освобождение ссылки на браузер
            webDriver.Quit();
        }
    }
}
```

Рисунок 7.7 – Тест для проверки веб-страницы

После сборки решения и запуска теста, будет открыто окно браузера Google Chrome, в нём будет открыт сайт

“https://sibsutis.ru/”, будет выполнена проверка заголовка страницы, после чего, окно браузера будет закрыто.

Получение доступа к веб элементам:

Для того, чтобы получить доступ к веб элементу страницы, удобнее всего использовать XPath. Для того, чтобы получить XPath объекта, кликните по нему правой кнопкой мыши и нажмите “Просмотреть исходный код” (рис. 7.8):

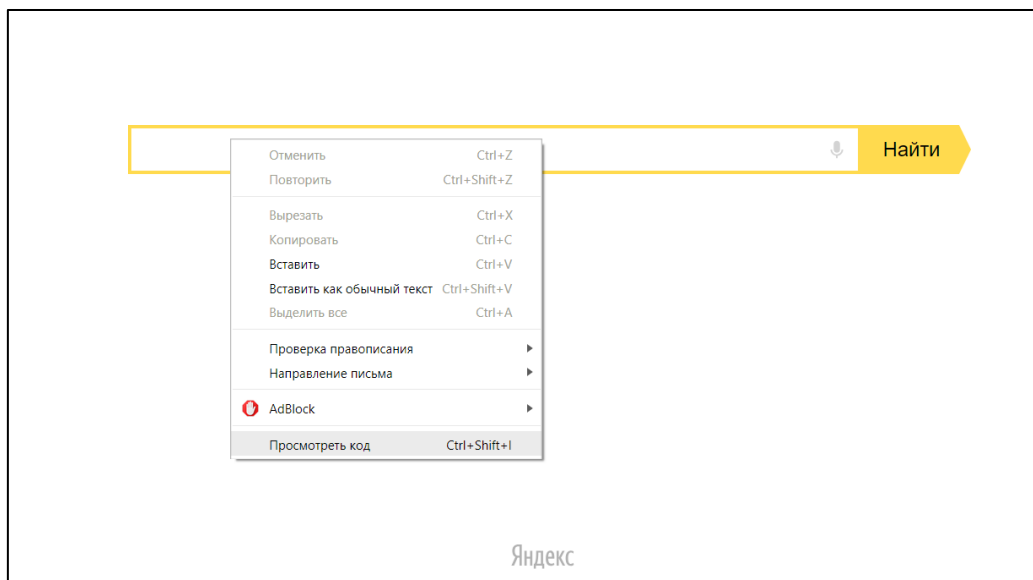


Рисунок 7.8 – Просмотр исходного кода

В появившемся окне, нажмите правой кнопкой мыши на нужный элемент → Сору->Сору XPath(рис. 7.9):

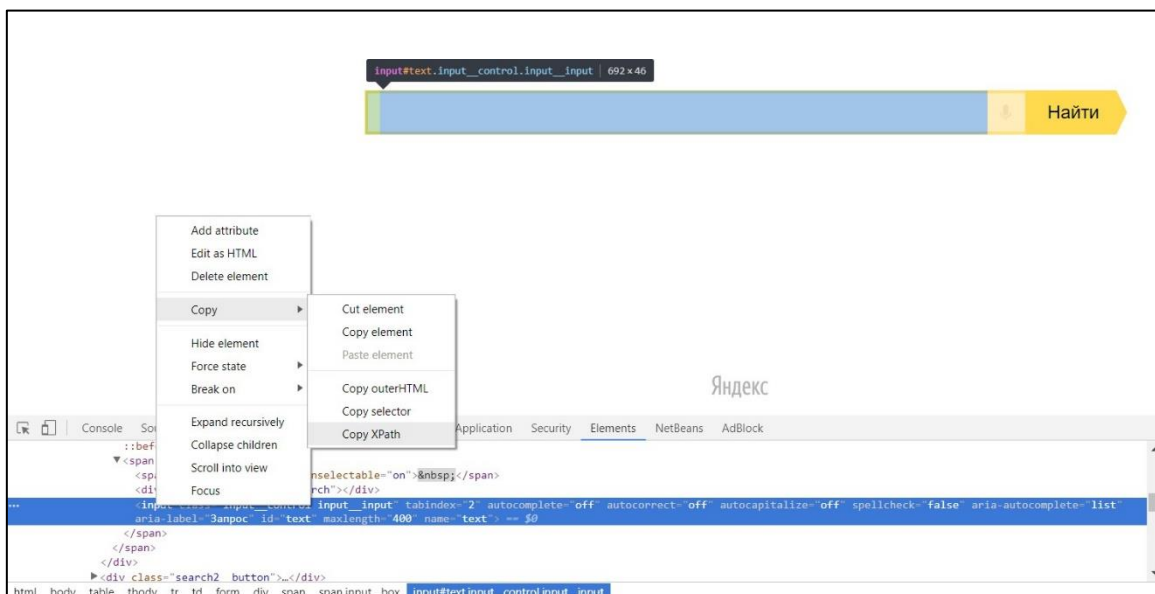


Рисунок 7.9 – Просмотр исходного кода разработчика

В исходном коде (рис. 7.10):

```
[TestCase]
public void yaru_request()
{
    webDriver.Url = "https://ya.ru/";
    //получение ссылки на строку поиска
    IWebElement search = webDriver.FindElement(By.XPath("//*[@id=\"text\"]"));
    //ввод запроса
    search.SendKeys("функциональное тестирование");

    //получение ссылки на кнопку "Найти"
    IWebElement button = webDriver.FindElement(By.XPath("/html/body/table/tbody/tr[2]/td/form/div[2]/button"));
    button.Click();
}
```

Рисунок 7.10 – Код теста

Доступные команды:

Существует три типа команд:

Команды браузера.

Команды Веб Элемента.

Dropdown commands.

Ниже приведён список команд, которые могут потребоваться при выполнении лабораторной работы:

Команды браузера (IWebDriver) представлено в таблице 7.1.

Таблица 7.1 – Команды браузера (WebDriver)

Имя команды	Описание	Синтаксис
Click command	Используется для щелчка по веб-элементу. Чтобы элемент был кликабельным, он должен быть виден на веб-странице. Эта команда также используется для операций с флажками и переключателями.	IWebElement element = driver.FindElement(By.xpath("xpath of WebElement")); element.Click();

Clear command	Используется специально для очистки существующего содержимого текстовых полей.	<code>IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); element.Clear();</code>
SendKeys command	Используется для ввода значения в текстовые поля. Вводимое значение должно быть передано в качестве параметра команде	<code>IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); element.SendKeys("guru99");</code>
Displayed command	Используется для определения видимости конкретного элемента на веб-странице. Команда возвращает булево значение, true или false, в зависимости от видимости веб-элемента.	<code>IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); Boolean status = element.Displayed;</code>
Enabled command	Используется для определения того, включен ли определенный веб-элемент на веб-странице. Команда возвращает булево значение: true или false.	<code>IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); Boolean status = element.Enabled;</code>
Selected	Используется	<code>IWebElement element = driv-</code>

command	для определения того, выбран ли конкретный веб-элемент. Она используется для флажков, радиокнопок и операций выбора.	<code>er.FindElement(By.xpath("xpath of Webelement")); Boolean status = element.Selected;</code>
Submit command:	Аналог команды <code>click</code> , разница заключается в том, есть ли в HTML-форме кнопка с типом <code>Submit</code> . Если команда <code>click</code> щелкает по любой кнопке, то команда <code>submit</code> щелкает только по кнопкам с типом <code>submit</code> .	<code>IWebelement element = driver.FindElement(By.xpath("xpath of Webelement")); element.submit();</code>
Text command	Возвращает внутренний текст <code>Webelement</code> . В качестве результата она возвращает строковое значение.	<code>IWebelement element = driver.FindElement(By.xpath("xpath of Webelement")); String text=element.Text;</code>
TagName command	Возвращает HTML-тег веб-элемента. В качестве результата возвращается строковое значение.	<code>IWebelement element = driver.FindElement(By.xpath("xpath of Webelement")); String tagName = element.TagName;</code>

GetCSSValue Command:	Используется для возврата цвета веб-элемента в виде строки rgba (Red, Green, Blue и Alpha).	IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); String color = element.getCSSValue; Output– If the color of element is red, output would be rgba(255,0,0,1)
----------------------	---	--

Выпадающие команды:

Операции с выпадающими элементами в C# могут быть выполнены с помощью класса SelectElement.

Ниже перечислены различные операции с выпадающими элементами, доступные в C# представлены в таблице 7.2.

Таблица 7.2 – Операции с выпадающими элементами

Имя команды	Описание	Синтаксис
SelectByText Command	Выбирает опцию выпадающего списка на основе текста опции.	IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); SelectElement select = new SelectElement(element); select.SelectByText("Guru99");
SelectByIndex Command	Используется для выбора опции по ее индексу. Индекс выпадающего списка начинается с 0.	IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); SelectElement select = new SelectElement(element); select.SelectByIndex("4");
SelectByValue Command	Используется для выбора параметра на основе его значения.	IWebElement element = driver.FindElement(By.xpath("xpath of Webelement")); SelectElement select = new SelectElement(element); select.SelectByValue("Guru99");
Options Command	Используется для получения	IWebElement element = driver.FindElement(By.xpath("xpath

	списка опций, отображаемых в раскрывающемся списке.	of WebElement”)); SelectElement select = new SelectElement(element); List<WebElement> options = select.Options; int size = options.Count; for(int i=0;i<options.size();i++) { String value = size.elementAt(i).Text; Console.WriteLine(value); } The above code prints all the options onto console within a dropdown.
IsMultiple command	Используется для определения того, является ли выпадающий список многовариантным; многовариантный список позволяет пользователю выбрать несколько опций в выпадающем списке одновременно. Эта команда возвращает булево значение.	WebElement element = driver.FindElement(By.xpath(“xpath of WebElement”)); SelectElement select = new SelectElement(element); Boolean status = select.IsMultiple();
DeSelectAll command	Используется в раскрывающихся списках с несколькими вариантами выбора. Она очищает уже выбранные варианты.	WebElement element = driver.FindElement(By.xpath(“xpath of WebElement”)); SelectElement select = new SelectElement(element); select.DeSelectAll();
DeSelectByIndex command	Снимает выделение с уже вы-	WebElement element = driver.FindElement(By.xpath(“xpath

	бранного значения по его индексу.	of WebElement”)); SelectElement select = new SelectElement(element); select.DeSelectByIndex(“4”);
DeSelectByValue command	Снимает выделение с уже выбранного значения, используя его значение.	IWebElement element = driver.FindElement(By.xpath(“xpath of WebElement”)); SelectElement select = new SelectElement(element); select.DeSelectByValue(“Guru99”);
DeSelectByText command	Снимает выделение с уже выделенного значения, используя его текст.	IWebElement element = driver.FindElement(By.xpath(“xpath of WebElement”)); SelectElement select = new SelectElement(element); select.DeSelectByText(“Guru99”);

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Выбрать web-site (например, sibsutis.ru).
2. Составить план функционального тестирования для выбранного сайта.
3. Реализовать набор тестов при помощи NUnit и Selenium WebDriver. План тестирования должен включать в себя как минимум:

1. Проверка заголовка страницы.
2. Проверка видимости объектов
3. Переход по ссылке.
4. Заполнение текстового поля.
5. Эмуляция нажатия на кнопку.

Процесс разработки должен вестись в локальном репозитории. В процессе разработки необходимо:

1. Совершать сохранение изменений в локальный репозиторий после реализации каждой отдельной функции.

Синхронизировать локальный и удалённый репозиторий в конце каждого сеанса работы над проектом.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как запустить браузер с помощью WebDriver?
2. Какие типы ожидания доступны в WebDriver?
3. Как Selenium обрабатывает всплывающие окна?

ПРАКТИЧЕСКАЯ РАБОТА №8. АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ БАЗЫ ДАННЫХ

Целью выполнения работы является рассмотрения вопросов связанных с тестированием баз данных, как частью процесса обеспечения качества ПО. В практической работе мы рассмотрим основные аспекты тестирования баз данных, а также предложенные подходы и методики для эффективного выполнения этой задачи.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Виды тестирования баз данных:

Тестирование структуры данных (DDL) – проверка корректности создания, изменения и удаления таблиц, индексов, ограничений и других объектов базы данных.

Тестирование манипуляции данными (DML) – проверка правильности выполнения операций добавления, изменения, удаления и выборки данных.

Тестирование хранимых процедур и функций – проверка корректности работы программного кода, хранящегося в базе данных.

Тестирование интеграции с приложениями – проверка правильности взаимодействия базы данных с приложениями, использующими её для хранения и обработки данных.

Тестирование производительности – проверка скорости выполнения запросов, а также определение возможных узких мест и бутылочных горлышек в работе базы данных.

Тестирование безопасности – проверка защищённости базы данных от несанкционированного доступа, а также от потери и утечки данных.

Типы тестирования БД

Тестирование структуры БД – проверка таблиц и колонок, схем, процедур и представлений, триггеров и ключей

Функциональное – тестирование основных функций БД; по методу черного или белого ящика

Нефункциональное – в основном тестирование производительности: нагрузочное, стрессовое и другие виды

При тестировании баз данных проверяются бэкэнд-записи, введенные через веб или десктоп-приложение. Данные, которые

отображаются в приложении, должны совпадать с данными, хранящимися в базе данных.

Чтобы тестировать базы данных, тестировщик должен знать следующее:

Тестировщик должен понимать функциональные требования, бизнес-логику, основной сценарий приложения и дизайн базы данных.

Тестировщик должен разбираться в таблицах, триггерах, процедурах хранения, способах отображения и указателях, используемых для приложения.

Тестировщик должен понимать логику триггеров, процедур хранения, способов отображения и указателей.

Тестировщик должен понимать, какие таблицы затрагиваются, когда операции вставки, обновления и удаления выполняются в приложении.

Понимая вышеперечисленные пункты, тестировщик может легко написать сценарии для тестирования баз данных.

Сценарии тестирования баз данных:

Проверьте название базы данных: оно должно совпадать со спецификацией.

Проверьте таблицы, колонки, типы колонок и значения по умолчанию: все это должно совпадать со спецификацией.

Проверьте, позволяет ли колонка значение null.

Проверьте первичный и внешний ключ каждой таблицы.

Проверьте процедуры хранения.

Протестируйте, установлена ли процедура хранения.

Проверьте название процедуры хранения.

Проверьте названия параметров, их типы и количество.

Проверьте, обязательны параметры или нет.

Проверьте процедуру хранения, удалив некоторые параметры.

Проверьте базу данных, если на выходе ноль – записи с нулем должны быть задействованы.

Проверьте процедуру хранения, задав простые SQL-запросы.

Убедитесь, что процедура возвращает значения.

Проверьте процедуру вводом тестовых данных.

Проверьте поведение каждого флага в таблице.

Убедитесь, что данные правильно сохраняются в базе данных после каждого ввода.

Проверьте данные при каждой операции обновления, удаления и вставки.

Проверьте длину каждого поля. Длина на бэкэнде и фронтэнде должны совпадать.

Проверьте названия баз данных QA, UAT и прода. Имена должны быть уникальными.

Проверьте зашифрованные данные в базе.

Проверьте размер базы и время отклика на каждый запрос.

Проверьте данные, отображающиеся на фронтэнде, и убедитесь, что они совпадают с бэкэндом.

Проверьте целостность данных, вводя невалидные значения в базу.

Проверьте триггеры.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Для тестирования базы данных MySQL или MariaDB можно использовать различные инструменты и методы, включая:

1. Написание и выполнение тестовых запросов. Это может включать проверку структуры таблиц, индексов, уникальных и внешних ключей, а также проверку правильности вставки, обновления и удаления данных.

Например:

-- Проверка структуры таблицы

DESCRIBE mytable;

-- Проверка индексов и ключей

SHOW INDEX FROM mytable;

SHOW CREATE TABLE mytable;

-- Вставка данных

INSERT INTO mytable (col1, col2) VALUES ('value1', 123);

SELECT * FROM mytable WHERE col1 = 'value1';

-- Обновление данных

UPDATE mytable SET col1 = 'value2' WHERE col2 = 123;

SELECT * FROM mytable WHERE col2 = 123;

-- Удаление данных


```
DELETE FROM mytable WHERE col1 = 'value2';
SELECT * FROM mytable;
```

2. Использование инструментов автоматического тестирования базы данных, таких как DBUnit, JMeter, Gatling и другие. Эти инструменты позволяют написать тесты и проверять производительность базы данных, ее масштабируемость и отказоустойчивость.

3. Использование систем мониторинга и профилирования базы данных, таких как MySQL Workbench, phpMyAdmin и другие. Эти системы позволяют отслеживать производительность базы данных, анализировать запросы и выявлять узкие места в работе базы данных.

4. Использование инструментов для сбора статистики и анализа производительности базы данных, таких как MySQL Performance Schema, MariaDB MaxScale и другие. Эти инструменты позволяют отслеживать производительность базы данных в режиме реального времени, а также проводить анализ накопленных данных.

Примеры тестирования базы данных MySQL:

Конкретные запросы на проверку безопасности и производительности MySQL и MariaDB зависят от конкретных требований и настроек сервера. Ниже приведены некоторые примеры таких запросов:

1. Проверка наличия анонимных пользователей:

```
SELECT user, host FROM mysql.user WHERE user = '';
```

2. Проверка наличия слабых паролей:

```
SELECT user, host FROM mysql.user WHERE
LENGTH(password) < 8;
```

3. Проверка наличия пользователей с доступом к базам данных, к которым они не должны иметь доступ:

```
SELECT DISTINCT USER,HOST FROM mysql.db WHERE
db NOT IN ('mysql','information_schema','performance_schema');
```

4. Проверка уровня изоляции транзакций:

```
SELECT @@GLOBAL.tx_isolation, @@tx_isolation;
```

5. Проверка производительности базы данных с помощью EXPLAIN:

```
EXPLAIN SELECT * FROM table_name WHERE id = 1;
```

6. Проверка производительности базы данных с помощью

индексов:

```
SELECT COUNT(*) FROM table_name WHERE indexed_column = 'value';
```

7. Проверка производительности базы данных с помощью инструмента pt-query-digest:

```
pt-query-digest /var/log/mysql/mysql-slow.log
```

Создать отчет по работе, включающий отображение в виде копий экрана этапов выполнения работы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что нужно проверять во время тестирования базы данных?
2. Что такое повторное тестирование баз данных?
3. Что такое нагрузочное тестирование баз данных?

ПРАКТИЧЕСКАЯ РАБОТА №9. ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ПРИЛОЖЕНИЯ

Целью работы является проведение функционального тестирования приложений. Следует ознакомиться с основными принципами функционального тестирования, изучить инструменты, создавать тест-кейсы и проводить практическое тестирование. Для этого можно использовать любой сайт или приложение.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Функциональное тестирование необходимо для проверки продукта на соответствие заявленным требованиям. Например, проверка API, базы данных, пользовательского интерфейса, функциональности тестируемого продукта. Проверяется на соответствие спецификациям, бизнес-требованиям. Основано на требованиях клиента. Оно гарантирует, что пользователь сможет использовать продукт по назначению.

При функциональном тестировании:

Мы проверяем, что система выполняет те задачи, которые были заявлены в требованиях к приложению;

Обычно выполняется с использованием тест-кейсов, которые описывают шаги для проверки функций продукта;

Проще провести проверку, не прибегая к использованию дополнительных инструментов.

Существует много инструментов для автоматизации функционального тестирования. Про самые популярные мы поговорим ниже:

Selenium – один из самых популярных инструментов для автоматизации тестирования веб-приложений. Selenium позволяет работать на различных операционных системах (Windows, Mac, Linux) и браузерах (Chrome, Firefox и т.д), поддерживает различные языки для написания скриптов.

Appium – это кроссплатформенный инструмент для автоматизации тестирования мобильных приложений (нативные, гибридные и веб). Позволяет создавать и запускать тесты на реальных устройствах и эмуляторах, использовать различные языки программирования для написания тестов.

Katalon Studio – инструмент для автоматизации API, веб, десктоп и мобильных приложений. Для написания тестовых скриптов используется язык программирования Java или Groovy. Плюс данного инструмента – для работы с ним достаточно начальных знаний языков программирования. Благодаря чему он отлично подходит для новичков.

Ranorex Studio – еще один универсальный инструмент для автоматизации тестирования. Этот инструмент, как и предыдущий, хорошо подойдет для новичков, так как поддерживает возможность создания тестов без написания кода.

Инструменты для тестирования API:

Postman – с его помощью можно составлять и отправлять запросы, собирать коллекции и делиться ими с коллегами. Также в Postman можно писать автотесты для тестирования API.

SoapUI – с помощью данного инструмента можно легко и удобно тестировать как SOAP, так и REST-сервисы. Можно проверять работоспособность веб-сервисов, устанавливать доступность, работу различных запросов и отслеживать получение ответов.

Swagger UI – инструмент для описания и проверки API-методов. К каждому запросу есть пример ответа и описание приходящих в них параметров. Не требует установки на устройство пользователя.

Системы логирования:

Kibana – инструмент визуализации и анализа данных в реальном времени, отслеживания трендов и прогнозирования будущих событий.

Graylog – система для сбора, хранения, мониторинга и анализа логов из различных источников в режиме реального времени. Позволяет сохранять большие объемы логов и анализировать их, используя поисковые запросы и фильтры.

Android Studio и Xcode – можно собирать логи с мобильных устройств – Android и Xcode соответственно – в режиме реального времени. Удобно собирать, читать и анализировать логи.

Этапы функционального тестирования

1. Определить и проанализировать, какую функциональность необходимо протестировать. Перед началом необходимо изучить тестируемый функционал: какие у нее требования, как

она должна работать, как пользователь будет её использовать.

2. Написать тест-кейсы. В них тестирующий пошагово описывает сценарий проверки определенной функциональности.

3. Подготовка тестовых данных. Для тестирования используются данные, которые максимально приближены к тем, что могут использовать пользователи. Сбор тестовых данных основывается на требованиях.

4. Проведение тестирования. Происходит сравнение фактического результата и ожидаемого.

5. Составление отчета по результатам тестирования. По завершении тестирования необходимо собрать отчет с результатами прогонки тестов, списком багов и рекомендациями по улучшению продукта.

Для любого приложения выполняется тестирование графического интерфейса пользователя (таблица 9.1) На всех последующих шагах оставим настройки по умолчанию и на самом последнем экране для установки нажмем на кнопку «Установить». Перечень основных GUI проверок для всего приложения представлен в таблице 9.1.

Таблица 9.1 – Перечень основных GUI проверок для всего приложения

Название проверки	Описание проверки
1. Правописание	Лексические, грамматические и пунктуационные ошибки
2. Расположение и выравнивание	Выравнивание по левому или правому краю (в зависимости от требований приложения), отступы, идентичность расстояний между названием и полем. Корректное расположение текста, длинный текст не выходит за границы поля при вводе.
3. Длинные названия	Длинные названия корректно обрезаются с помощью многоточия в конце, при наведении возникают хинты с полнотекстовым вариантом.

4. Соответствие названий форм / элементов GUI их назначению	Проверка названий форм / элементов GUI с точки зрения их смысловой нагрузки.
5. Унификация (стиля, цвета, шрифта, названий)	Единообразие цвета, шрифта, размеров (высоты/ширины), выравнивания полей, названий полей, категорий меню и др. в рамках всего приложения.
6. Эффект «нажатия»	Изменение вида ссылок, кнопок, позиций меню и др. при наведении курсора. Изменение вида курсора при наведении на ссылки, кнопки, позиции меню и др.
7. Хинты	Проверка всплывающих подсказок с точки зрения правописания, выравнивания, соответствия назначению и др.
8. Сообщения об успешном / неуспешном завершении действия, о подтверждении действия	Проверка верхней панели (логотипа и названия) формы с сообщением. Если присутствует кнопка «Отмена», то в правом верхнем углу формы с сообщением присутствует «крестик» для альтернативной возможности закрыть форму. Сообщения о подтверждении удаления по умолчанию активированы на кнопку «нет».
9. Изменение размеров окна, изменение масштаба страницы	Появление скроллинга при уменьшении размера окна. Сохранение взаимного расположения элементов при уменьшении окна, изменении масштаба). Перераспределение элементов с сохранением пропорций при изменении масштаба страницы.

Общие проверки функциональности для всех типов приложений, а также общие проверки для web приложений приведены в таблицах 9.2.

Таблица 9.2 – Перечень общих проверок функциональности для любого типа приложения

Название проверки	Описание проверки
1. Табуляция	Перемещение с помощью клавиатуры должно осуществляться сверху вниз слева направо. Недоступные поля должны пропускаться.
2. «Хлебные крошки»	«Хлебные крошки» – элемент навигации, являющийся признаком удобства пользования приложением в целом и перемещением по его структуре.
3. Скроллинг	Отсутствие скроллинга в случае, если текст вмещается на странице без прокрутки. Соответствующее изменение текста при использовании скроллинга. Возможность изменения положения скроллинга при помощи мыши, кнопок Page up/down, Home/End.
4. Взаимосвязь компонентов	Поведение одного компонента при изменении/удалении другого (например, при удалении категории товара не должны удаляться все товары в этой категории).
5. Фокус на кнопке для исполнения действий	Ввод данных нажатие Enter действие осуществилось.

Таблица 9.3– Перечень общих проверок функциональности для web приложений

Название проверки	Описание проверки
1. Подготовка к тестированию	Перед тестированием каждой новой сборки необходимо осуществить очистку кэша и cookies. Для этого можно воспользоваться приложением CCleaner.

2. 404 Error	Переход по некорректному адресу должен вести на страницу с Error 404, а не на страницу Page cannot be found, например. Страница Error 404 должна быть реализована в общем дизайне тестируемого приложения.
3. Логотип	Логотип должен быть ссылкой на главную страницу.
4. Email нотификации	Проверка работоспособности отправки email нотификаций (как администратору, так и пользователю), если только отсутствие писем не является спецификой проекта.
5. Отображение flash-элементов при отключенном или неустановленном в браузере flash-плеере	Пользователю должно быть предложено скачать и установить последнюю версию flash-плеера; на месте flash-объекта должно отображаться альтернативное изображение.
6. Проверка работоспособности приложения при отключенном JavaScript	Основная функциональность и навигация должна работать.

Для любого приложения проверяют функциональность элементов пользовательского интерфейса: поле для ввода данных, поле для загрузки файлов, поле для ввода даты, поле со списком/выпадающий список, кнопка, радиобаттон, чекбокс, меню, таблица, календарь, ссылка, сообщения, поп-ап (всплывающие окна). В таблицах 9.4, 9.5, 9.6 приведены основные проверки для указанных элементов пользовательского интерфейса.

Таблица 9.4 – Перечень основных проверок для поля ввода данных

Functional Test	GUI Test
<ol style="list-style-type: none"> 1. Обязательность ввода. 2. Обработка только пробелов. 3. Использование пробелов в тексте (3.1. пробелы в начале и в конце строки должны отсекается при сохранении, 3.2. пробелы внутри текста отсекается не должны). 4. Минимально/максимально допустимое количество символов. 5. Формат данных (исходя из его логического назначения и требований приложения). 6. Формат числовых данных (если допускаются): негативные, дробные с точкой и запятой. 7. Использование специальных символов (введенные символы должны отобразиться в том же виде, в котором они были введены, если только ввод спец. символов не запрещен требованиями приложения). 8. Возможность редактирования введенных значений. 9. Корректное распределение текста по строкам (переход на новую строку автоматически). 10. Уникальные данные (например, уникальность логина, email). 11. Автоматическая постановка курсора в первое поле для ввода при открытии формы. 12. Ввод тегов и скриптов (введенные теги и скрипты должны отобразиться в том же виде, в котором они были введены). 	<ol style="list-style-type: none"> 1. Название поля (правописание, соответствие названия тематики модуля/страницы). 2. Выравнивание названий полей (выравнивание по левому или правому краю в зависимости от требований приложения, отступы, идентичность расстояний между названием и полем). 3. Корректное расположение текста, длинный текст не выходит за границы поля при вводе. 4. Унификация дизайна по отношению ко всему приложению (цвет, шрифт, размер (высота/ширина), выравнивание полей). 5. Расположение вводимого текста внутри поля (унификация, выравнивание).

Таблица 9.5 – Перечень основных проверок для меню

Functional Test	GUI Test
Осуществление соответствующего перехода при выборе пункта меню.	Подсветка категории меню при наведении курсора.
Визуальное различие в момент работы на определенной вкладке (подсветка, подчеркивание).	Изменение курсора при наведении на категорию меню. Если в данный момент выполняется работа в выбранной вкладке, то в меню она отличается визуально и является некликабельной. Совпадение названий категорий меню в случае, если меню дублируется в нескольких местах.

Таблица 9.6 – Перечень основных проверок для ссылки

Functional Test	GUI Test
Функционирование ссылки (должен осуществиться переход на соответствующую страницу).	Унификация стилей (в соответствии с дизайном сайта).
Переход по загруженной ссылке должен осуществляться в новой вкладке или в всплывающем окне.	Расположение ссылок (в соответствии с дизайном сайта). Например, расположение всех ссылок слева или справа от элементов.
Форматы ссылок и префиксов.	Названия (унификация, идентичность названий ссылок одинакового назначения, спеллинг, соответствие с открытым модулем или страницей, вместимость названия ссылки в отведенном блоке).
Срабатывание ссылки только при клике на саму ссылку, а не на пустую область возле нее.	Изменение вида курсора при наведении на ссылку. Изменение вида ссылки при наведении курсора (подчеркивание).

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Выбрать любой интернет-сайт
2. Провести функциональное тестирование этого сайта.
3. Найти дефекты в работе сайта, сравнив отображение на

ПК в различных браузерах.

4. Заполнить тест-план и 10 тест-кейсов для тестирования сайта.

5. Защита отчета по лабораторной работе заключается в предъявлении преподавателю полученных результатов, демонстрации полученных навыков в ответах на вопросы преподавателя.

6. Составить итоговый отчет по результатам тестирования web приложения.

7. Указать общую информацию о тестируемом продукте (название, номер сборки, виды выполненных тестов, количество обнаруженных дефектов, вид рабочей тестовой документации).

8. Указать, кто и когда тестировал программный продукт.

9. Описать тестовое окружение (ссылку на web приложение, браузер).

10. Указать общую оценку качества протестированного приложения и подробно ее обосновать.

11. Графически (в виде круговой диаграммы) отразить процентное соотношение дефектов GUI и функциональных дефектов.

12. Графически (в виде столбчатой диаграммы) отразить распределение дефектов по различным степеням критичности.

13. Графически (в виде столбчатой диаграммы) отразить распределение дефектов по модулям.

14. Произвести детальный анализ качества всех модулей протестированного приложения с аргументацией выставленных уровней качества.

15. Привести список пяти наиболее критичных дефектов.

16. Сформулировать рекомендации по улучшению качества программного продукта.

17. Оформить отчет и защитить лабораторную работу.

18. Содержание отчета:

19. Цель работы.

20. Итоговый отчет о результатах тестирования web приложения.

21. Выводы по работе.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какая структура итогового отчета о результатах тестирования?
2. Что содержится в разделе Общая информация?
3. Что содержится в разделе Тестовое окружение?
4. Как выставляется общая оценка качества приложения?
5. Как обосновать выставленную оценку качества?
6. Для чего используется графическое представление результатов тестирования в итоговом отчете?
7. Что содержится в разделе Детализированный анализ качества?
8. Что содержится в разделе Рекомендации?

ПРАКТИЧЕСКАЯ РАБОТА №10. ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ ВЕБ-СЕРВИСА В СИСТЕМЕ SELENIUM

Целью работы – провести функциональное тестирование программы, составить сценарии тестирования. Составить тестовый план и провести тестирование согласно тестовому плану.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Наиболее популярной областью применения Selenium является автоматизация тестирования веб-приложений. Однако при помощи Selenium можно (и даже нужно!) автоматизировать любые другие рутинные действия, выполняемые через браузер. Разработка Selenium поддерживается производителями популярных браузеров. Они адаптируют браузеры для более тесной интеграции с Selenium, а иногда даже реализуют встроенную поддержку Selenium в браузере. Selenium является центральным компонентом целого ряда других инструментов и фреймворков автоматизации. Selenium поддерживает десктопные и мобильные браузеры. Selenium позволяет разрабатывать сценарии автоматизации практически на любом языке программирования. С помощью Selenium можно организовывать распределённые стенды, состоящие из сотен машин с разными операционными системами и браузерами, и даже выполнять сценарии в облаках.

Автоматическое тестирование с использованием Selenium.

Чтобы точно и точно идентифицировать веб-элементы, Selenium использует различные типы локаторов. Они бывают следующие:

- Id locator.
- Name locator.
- linkText & Partial linkText.
- CSS Selector.
- XPath.

Теперь рассмотрим пример, в котором с помощью средств автоматизации будет осуществляться навигация по определенной веб-странице.

Пример простого функционального теста веб-сервиса с использованием Selenium WebDriver на языке Python для тестиро-

вания страницы:

```
from selenium import webdriver
# Инициализация драйвера браузера
driver = webdriver.Chrome()

# Открытие страницы для тестирования
driver.get("https://www.example.com")
# Проверка заголовка страницы
assert "Example Page Title" in driver.title
# Нахождение элемента на странице (например, поле ввода)
input_field = driver.find_element_by_id("input_field_id")
# Ввод текста в найденное поле
input_field.send_keys("Текст для ввода")
# Нажатие на кнопку (например, кнопка отправки формы)
submit_button = driver.find_element_by_id("submit_button_id")
submit_button.click()
# Проверка результата теста (например, успешного ввода
данных)
assert "Success Message" in driver.page_source
# Закрытие браузера
driver.quit()
```

Это простой пример теста, который открывает страницу, вводит текст в поле, нажимает на кнопку и проверяет успешность выполнения действий. Не забудьте предварительно установить библиотеку Selenium для Python.

Таким образом, все, что вы делали вручную для перехода по различным страницам и ссылкам, делается автоматически с помощью средств автоматизации, таких как Selenium.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Выбрать web-ресурс, включающий в себя форму авторизации, форум и страницу поиска. Разработать и протестировать для него следующие сценарии с помощью Selenium:

- Тестирование регистрации на форуме
- Тестирование телефонного справочника
- Тестирование системы голосования
- Тестирование поиска

Необходимо использовать разные браузеры. Сценарии могут быть выбраны другие, но они должны быть согласованы с преподавателем.

Пример тестирования поиска:

- Зайти на сайт <https://miet.ru/search/>.
- Проверить наличие поля поиска
- Проверить наличие текста на странице “Поиск по сайту”.
- Ввести в строке поиска заведомо бессмысленный набор букв.
- Нажать кнопку “Поиск”.
- Проверить наличие на странице текста “К сожалению, на ваш поисковый запрос ничего не найдено”.
- Ввести в строке поиска слово “.....”.
- Нажать кнопку “Поиск”.
- Проверить наличие таблицы результатов
- Проверить наличие на странице текста “..... ” в категории “Люди”.
- Закрыть окно браузера.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое функциональное тестирование?
2. Список методов, используемых для выполнения функционального тестирования
3. Чем функциональное тестирование отличается от нефункционального тестирования?
4. Этапы функционального тестирования
5. Когда требуется функциональное тестирование сайта?

ПРАКТИЧЕСКАЯ РАБОТА №11. НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ WEB-ПРИЛОЖЕНИЯ В СИСТЕМЕ SELENIUM

Цель работы – получить практические навыки тестирования производительности для оценки соответствия системы или компонента специфичным требованиям к производительности.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Нагрузочное тестирование позволяет оценить производительность программного обеспечения при различных нагрузках от действий определенного количества пользователей. Нагрузочное тестирование позволяет снизить риск сбоя после запуска приложения в реальных условиях.

Рынок программного обеспечения сегодня полон различных инструментов нагрузочного тестирования, начиная от приложений с открытым исходным кодом и заканчивая инструментами для автоматизированного нагрузочного тестирования:

1. Apache JMeter
2. LoadRunner
3. Load Ninja
4. WebLOAD
5. LoadUI Pro
6. BlazeMeter
7. Selenium
8. Яндекс.Танк
9. Gatling
10. Boomq

Пример простого нагрузочного теста на Java с использованием Selenium WebDriver (java):

```
import org.openqa.selenium.chrome.ChromeDriver;  
import org.openqa.selenium.WebDriver;  
import java.util.concurrent.TimeUnit;  
public class LoadTestingExample {  
    public static void main(String[] args) {  
        int numberOfUsers = 10; // Количество пользователей  
        long testDurationSeconds = 60; // Продолжительность теста в секундах
```



```

    for (int i = 0; i < numberOfUsers; i++) {
        new Thread(() -> {
            WebDriver driver = new ChromeDriver();
            driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

            long startTime = System.currentTimeMillis();
            long currentTime = System.currentTimeMillis();

            while (currentTime - startTime < testDurationSeconds
* 1000) {
                driver.get("https://www.example.com"); // Замените
ссылку на адрес вашего тестируемого сайта
                // Добавьте здесь действия, которые вы хотите
проверить в рамках нагрузочного теста
                currentTime = System.currentTimeMillis();
            }

            driver.quit();
        }).start();
    }
}

```

Этот пример создает 10 потоков, каждый из которых открывает браузер Chrome, посещает указанный URL (замените его на свой сайт) и выполняет определенные действия в течение 60 секунд. Обратите внимание, что для выполнения этого теста вам потребуется установить WebDriver для браузера Chrome и подключить все необходимые зависимости к проекту.

Нагрузочное тестирование в системе Selenium пример теста на C#

Пример простого нагрузочного теста на C# с использованием Selenium WebDriver:

```

using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using System;
using System.Threading;

```

```

class LoadTestingExample
{
    static void Main()
    {
        int numberOfUsers = 10; // Количество пользователей
        int testDurationSeconds = 60; // Продолжительность теста
        в секундах

        for (int i = 0; i < numberOfUsers; i++)
        {
            Thread thread = new Thread(() =>
            {
                IWebDriver driver = new ChromeDriver();
                driver.Manage().Timeouts().ImplicitWait           =
                TimeSpan.FromSeconds(10);
                long          startTime          =          DateTimeOff-
                set.Now.ToUnixTimeMilliseconds();
                long          currentTime        =          DateTimeOff-
                set.Now.ToUnixTimeMilliseconds();
                while (currentTime – startTime < testDurationSeconds
                * 1000)
                {
                    driv-
                    er.Navigate().GoToUrl("https://www.example.com"); // Замените
                    ссылку на адрес вашего тестируемого сайта
                    // Добавьте здесь действия, которые вы хотите
                    проверить в рамках нагрузочного теста
                    currentTime          =          DateTimeOff-
                    set.Now.ToUnixTimeMilliseconds();
                }
                driver.Quit();
            });
            thread.Start();
        }
    }
}

```

Этот код создает 10 потоков, каждый из которых открывает

браузер Chrome, посещает указанный URL (замените его на свой сайт) и выполняет определенные действия в течение 60 секунд. Не забудьте установить WebDriver для браузера Chrome и подключить все необходимые зависимости к вашему проекту.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

С помощью программного пакета Apache JMeter провести нагрузочное и стресс-тестирование веб-приложения в соответствии с вариантом задания.

В ходе нагрузочного тестирования необходимо протестировать 3 конфигурации аппаратного обеспечения и выбрать среди них наиболее дешёвую, удовлетворяющую требованиям по максимальному времени отклика приложения при заданной нагрузке (в соответствии с вариантом).

В ходе стресс-тестирования необходимо определить, при какой нагрузке выбранная на предыдущем шаге конфигурация перестаёт удовлетворять требованиями по максимальному времени отклика. Для этого необходимо построить график зависимости времени отклика приложения от нагрузки.

Приложение для тестирования доступно только во внутренней сети кафедры.

Если запрос содержит некорректные параметры, сервер возвращает HTTP 403.

Если приложение не справляется с нагрузкой, сервер возвращает HTTP 503.

Отчёт по работе должен содержать:

Текст задания.

Описание конфигурации JMeter для нагрузочного тестирования.

Графики пропускной способности приложения, полученные в ходе нагрузочного тестирования.

Выводы по выбранной конфигурации аппаратного обеспечения.

Описание конфигурации JMeter для стресс-тестирования.

График изменения времени отклика от нагрузки для выбранной конфигурации, полученный в ходе стресс-тестирования системы.

Выводы по работе.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Назначение нагрузочного тестирования?
2. Каковы цели нагрузочного тестирования?
3. Назовите этапы нагрузочного тестирования
4. Какие виды нагрузочных тестов вы знаете?
5. Принципы реализации нагрузочного тестирования ПО.
6. Инструменты для реализации нагрузочного тестирования.
7. Apache JMeter – архитектура, поддерживаемые протоколы, особенности конфигурации.

СОДЕРЖАНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Цель самостоятельной работы обучающихся – получить новые знания по дисциплине «Тестирование информационных систем».

Самостоятельная работа необходима для формирования у обучающихся способности самостоятельно решать задачи профессиональной деятельности, формирования умения и навыков планирования времени, формирования стремления развиваться и совершенствоваться.

Аттестационной работой по курсу «Тестирование информационных систем» является экзамен, поэтому в качестве самостоятельной работы выступают этапы выполнения самостоятельной работы.

Виды самостоятельной работы обучающихся указаны в таблице 1.

Таблица 1 – Виды самостоятельной работы

№ п/п	Вид СРС
1	Выполнение интеграционного тестирования в MS Visual Studio
2	Разработка пакета тест-кейсов для учебной программы
3	Оформление отчетов об ошибках в системе
4	Функциональное тестирование веб-сервиса в системе Selenium
5	Нагрузочное тестирование web-приложения в системе Selenium

ЛИТЕРАТУРА

1. Федорова, Г. Н. Разработка, внедрение и адаптация программного обеспечения отраслевой направленности : Учебное пособие / Г. Н. Федорова. – Москва : НИЦ ИНФРА-М, 2021. – 336 с.
2. Гвоздева, В. А. Информатика, автоматизированные информационные технологии и системы : Учебник / В. А. Гвоздева. – Москва : НИЦ ИНФРА-М, 2023. – 542 с.
3. Гаврилов, М. В. Информатика и информационные технологии: учебник для СПО / Гаврилов М. В., Климов В. А.. – 4-е изд., пер. и доп. – Москва : Юрайт, 2021. – 383 с.
4. Голицына, О. Л. Информационные системы и технологии : Учебное пособие / О. Л. Голицына, Н. В. Попов И. И. Максимов. – Москва : НИЦ ИНФРА-М, 2023. – 400 с.
5. Спирина, М. С. Дискретная математика. Сборник задач с алгоритмами решений : учебное пособие для студентов среднего профессионального образования, обучающихся по специальностям 09.02.07 "Информационные системы и программирование", 09.02.06 "Сетевое и системное администрирование" / М. С. Спирина, П. А. Спирин ; М. С. Спирина, П. А. Спирин. – 4-е изд., стер. – Москва : Академия, 2020. – 288 с.
6. Семакин, И. Г. Основы алгоритмизации и программирования : учебник для образовательных организаций, реализующих программы среднего профессионального образования по специальностям "Информационные системы и программирование", "Сетевое и системное администрирование", "Обеспечение информационной безопасности автоматизированных систем", "Обеспечение информационной / И. Г. Семакин, А. П. Шестаков ; И. Г. Семакин, А. П. Шестаков. – 4-е изд., стер. – Москва : Академия, 2020. – 300 с.