МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования
«Витебский государственный технологический университет»

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические указания по выполнению
лабораторных работ для студентов специальности
6-05-0611-01 «Информационные системы и технологии»
и слушателей ФПКиПК специальности
9-09-0612-02 «Программное обеспечение информационных систем»

Витебск
2025

УДК 004.9
ББК 32.97
Т36

Составители:

Д. В. Черненко, А. С. Соколова

Одобрено кафедрой «Информационные системы и технологии»
УО «ВГТУ», протокол № 12 от 17.04.2025.

Рекомендовано к изданию редакционно-издательским советом
УО «ВГТУ», протокол № 8 от 23.04.2025.

Тестирование программного обеспечения: методические указания по выполнению лабораторных работ / сост.: Д. В. Черненко, А. С. Соколова, – Витебск УО «ВГТУ», 2025. – 30 с.

Методические указания предназначены для проведения лабораторных работ по дисциплине «Тестирование программного обеспечения». Рассматриваются вопросы анализа, планирования, проведения тестовых испытаний и оценки качества программного обеспечения на всех стадиях его жизненного цикла. Основной упор сделан на процессы модульного и интеграционного тестирования: рассмотрены их этапы, возможности автоматизации, особенности работы с инструментами автоматизации.

Методические указания предназначены для студентов специальности 6-05-0611-01 «Информационные системы и технологии» и слушателей ФПКиПК специальности 9-09-0612-02 «Программное обеспечение информационных систем».

Издание в электронном виде расположено в репозитории библиотеки УО «ВГТУ».

УДК 004.9
ББК 32.97

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА 1. ТЕСТ-ПЛАН, ЧЕК-ЛИСТ И ТЕСТ-КЕЙС

Цель работы: познакомиться с основными видами тестовой документации: план тестирования (Test Plan), чек-лист (Check-List), тест-кейс (Test Case).

Краткие теоретические сведения

План тестирования – это документ, который описывает объем работ по тестированию проекта. Он описывает стратегию тестирования, цели, график, оценку, результаты, а также ресурсы, необходимые для тестирования, и составляется до начала тестирования.

Форм-фактор у тест-плана может быть разный (схема, интеллектуальная карта и т. д.). Мы будем составлять тест-план в виде документа.

Тест-план должен содержать следующие элементы[1]:

- Описание объекта тестирования: системы, приложения, инструмент разработки.
- Список функций и описание тестируемой системы и её компонентов в отдельности.
- Стратегия тестирования, а именно: виды тестирования и их применение по отношению к объекту тестирования.
- Последовательность проведения работ: подготовка, тестирование, анализ результатов в разрезе запланированных фаз разработки.
- Критерии начала тестирования – это готовность тестируемого проекта, законченность разработки требуемого функционала, наличие всей необходимой документации и т. д.
- Критерии окончания тестирования, когда результаты тестирования удовлетворяют критериям качества продукта: требования к количеству открытых багов, тестовое покрытие[2], использование тестов с разным уровнем детализации (модульное, интеграционное, системное) и т. д.

ПРИМЕР: приведем пример составления тест-плана для проекта «Калькулятор».

Введение

Целью составления данного тест-плана является описание процесса тестирования проекта «Калькулятор».

Объект тестирования

Библиотека calc.jar разработанная на языке Java, web-интерфейс для калькулятора.

Предполагаемые проверки:

---

[1] Подробнее в главе 2.6.2 «Тест-план и отчёт о результатах тестирования» (стр. 209) книги:

Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – 3-е изд. – Минск: Четыре четверти, 2020. – 312 с.

[2] Отношение атомарных методов, проверяемых хотя бы одним тестом к общему количеству методов, выраженное в процентах.

- модульное тестирование;
- функциональное тестирование;
- тестирование пользовательского интерфейса.

Тестируемый функционал:

- дымовое тестирование;
- ввод данных;
- операции;
- специальные тесты;
- завершение работы.

Нагрузочное тестирование и тестирование безопасности проводиться не будет в виду отсутствия необходимых ресурсов.

Процесс тестирования

Для максимального покрытия проекта предполагается составление чек-листов и тест-кейсов.

Автоматизация тестирования

При написании кода будет использоваться модульное тестирование с помощью фреймворка JUnit.

На этапе интеграционного тестирования будет использован инструмент линейки Selenium. Выбор конкретного инструмента будет произведен на основании наличия необходимых ресурсов.

Критерии начала и окончания тестирования

Тестирование может быть начато, если выполнены следующие условия:

1. Готова и утверждена необходимая документация.
2. Тестируемый функционал окончен и готов для передачи в тестирование.

Тестирование окончено, если выполнены следующие условия:

Все найденные дефекты задокументированы.

Таблица 1.1 – План работ

Задача	Время	Дата начала	Дата окончания
Составление тест-плана и чек-листа	4 час	03.04.2023	03.04.2023
Корректировка тест-плана и чек-листа	1 час	04.04.2023	04.04.2023
Написание модульных тестов[3]	5 % от времени разработки	03.04.2023	26.04.2023
Разработка интеграционных тестов	6 часов	27.04.2023	27.04.2023
Выполнение интеграционных тестов	0,5 часа	28.04.2023	28.04.2023
Написание баг-репорта	2 часа	28.04.2023	28.04.2023

Чек-лист – документ, описывающий, что должно быть протестировано в проекте. По сути, это набор идей, которые можно сформировать либо на основе опыта работы над подобными проектами, либо на основе поиска информации в интернете, либо на основании коллективного мозгового штурма.

Разработка чек-листа проводится на этапе, когда сформированы

[3] В процессе разработки кода.

пользовательские и системные требования к разрабатываемому продукту, но ещё до начала работы над кодом проекта, однако может корректироваться и на этапе разработки и отладки.

Чек-лист необходим чтобы:

- не забыть, что нужно протестировать;
- оптимизировать и упорядочить тестирование;
- структурировать последовательность процесса тестирования.

Пример чек-листа для тестирования десктопного приложения «Калькулятор» составлен в виде таблицы и приведен в отдельном документе[4]. Данный чек-лист достаточно подробный. Это хорошо для реального проекта и способствует обеспечению высокого качества[5]. Для более объемного чем приведенное приложение проекта детальность чек-листа может быть уменьшена. Однако это не означает ограничение фокуса тестирования, на какой-то отдельной функциональности. *Тесты должны охватывать все модули проекта!*

Тест-кейс – это чёткое описание действий, которые необходимо выполнить, для того чтобы проверить работу программы (поля для ввода, кнопки и т. д.). Данное описание содержит: действия, которые надо выполнить до начала проверки – предусловия; действия, которые надо выполнить для проверки – шаги; описание того, что должно произойти, после выполнения действий для проверки – ожидаемый результат.

Один тест-кейс соответствует одному пункту чек-листа. Но это скорее исключение, чем правило. Чек-лист – это набор идей, а тест-кейс – это непосредственная реализация идеи и вариантов её использования намного больше. Поэтому одну идею чек-листа нужно максимально покрыть тестами (оформленных в виде тест-кейсов). Поскольку исчерпывающее тестирование невозможно, то необходимо, используя теорию тестирования, покрыть минимальным количеством тестов максимальный объем разрабатываемого продукта. Для этого используют техники тест-дизайна.

Тест-кейс[6] состоит из следующих пунктов:

- идентификатор;
- приоритет;
- связанное с тест-кейсом требование;
- модуль и подмодуль приложения;
- заглавие (суть) тест-кейса;
- исходные данные, приготовления;
- шаги тест-кейса и ожидаемые результаты.

[4] Документ «ChekListCalc» в материалах лабораторной работы 1 курса «Тестирования программного обеспечения» Виртуальной образовательной среды ВГТУ: https://sdo.vstu.by/course/view.php?id=728.

[5] Под качеством программного продукта понимают удовлетворение требований клиента и степень готовности продукта.

[6] Наиболее полный формат тест-кейса описан в главе 2.4.3 «Тест-план и отчёт о результатах тестирования» (стр. 126) книги:

Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – 3-е изд. – Минск: Четыре четверти, 2020. – 312 с.

Пример тест-кейсов для тестирования нескольких функций приложения «Калькулятор» составлен в виде таблицы и приведен в отдельном документе[7]. С примером также можно ознакомиться в рабочей тетради С. С. Куликова «Автоматизированное тестирование»[8].

Порядок выполнения работы

В качестве объекта тестирования для лабораторной работы можно взять:

- приложение, разрабатываемое студентом в рамках курсового проектирования;
- информационную систему, указанную в таблице 5.1;
- заданное преподавателем web-приложение.

Для выполнения работы рекомендуется следующий порядок действий:

1. Изучение теории

- Ознакомьтесь с основными определениями и принципами создания и работы с тестовой документацией. Отметьте различия между Test Plan, Check-List и Test Case.
- Просмотрите материалы лекций или рекомендованную литературу по данной тематике.

2. Создание тестовой документации

- Составьте примерный *план тестирования (Test Plan)*, включающий цель тестирования, охват функциональности, используемые инструменты, сроки выполнения и участников.
- Разработайте *чек-лист (Check-List)* для конкретной области тестирования, например, проверки интерфейса или функциональности.
- Создайте *тест-кейс (Test Case)*, подробно описывающий шаги выполнения, ожидаемые результаты и критерии успешности.

3. Прикладная часть

- Выполните тестирование по созданным документам.
- Занесите результаты тестирования в соответствующие поля тест-кейсов.

4. Анализ и выводы

- Оцените результативность составленных документов, отметьте их сильные и слабые стороны.
- Напишите краткий отчет, описывающий процесс работы и полученные выводы.

Содержание отчета

Отчет по лабораторной работе должен содержать:

- название лабораторной работы;

[7] Документ «TestCaseCalc» в материалах лабораторной работы 1 курса «Тестирования программного обеспечения» виртуальной образовательной среды ВГТУ: https://sdo.vstu.by/course/view.php?id=728.

[8] Главы 1.4.3 «Создание и оформление тест-кейсов» (стр. 31) и 1.4.4 «Свойства качественных тест-кейсов» (стр. 33) документа «Автоматизированное тестирование. Святослав Куликов» материалов курса «Тестирования программного обеспечения» виртуальной образовательной среды ВГТУ: https://sdo.vstu.by/course/view.php?id=728.

- ФИО студента и дату выполнения работы;
- цель работы;
- разработанную тестовую документацию (план тестирования, чек-лист, четыре тест-кейса);
- подробное описание действий и результатов с использованием скриншотов;
- выводы.

В выводах необходимо отразить всё новое, что узнали, выполняя лабораторную работу. Например, что представляет собой тест-план, чек-лист и тест-кейс. *Перечислять, что сделано в самой лабораторной работе, в выводе нет необходимости*.

Контрольные вопросы

1. Что такое тестирование программного обеспечения?
2. Какие разделы включены в план тестирования?
3. Что такое чек-лист?
4. Что содержит ожидаемый результат тест-кейса?
5. Что относятся к атрибутам тест-кейса?

ЛАБОРАТОРНАЯ РАБОТА 2. МОДУЛЬНОЕ ТЕСТИРОВАНИЕ, ИСПОЛЬЗОВАНИЕ JUNIT

Цель работы: познакомиться с модульным тестированием, которое является наиболее ранним видом тестирования, выполняемом на стадии кодирования и выполняемого самими разработчиками.

Краткие теоретические сведения

Модульное тестирование (unit testing, юнит-тестирование) – вид тестирования, направленный на оценку корректности исходного кода программы. Модульные тесты покрывают атомарные участки кода, что позволяет удостовериться в их работоспособности (в т. ч. после внесения изменений). Модульные тесты проверяют как отдельные методы, так и взаимодействие объектов в проекте.

Преимущества модульных тестов:

- повышают качество архитектуры приложения;
- стимулируют написание простых методов;
- поощряют изменения в коде;
- упрощают интеграцию кода;
- помогают документированию кода;
- минимизируют зависимости в системе;
- создаются на основе бесплатных фреймворков;
- созданы для многих языков программирования.

В лабораторной работе мы будем использовать JUnit – фреймворк

модульного тестирования ПО на языке Java. Он имеет широкий набор расширений – таких как jMock, HtmlUnit и т. д. Адаптирован для других языков, таких как: PHP, C#, Python, Delphi, Perl, C++, JavaScript и т. д.

Порядок выполнения работы

1. На начальном этапе выполним задачу по тестированию уже готового кода, представленного в виде файла calculator-1.0.jar. В данной работе есть возможность выполнять тестирование в различных средах программирования языка Java. Выберете одну из сред программирования и откройте соответствующую инструкцию. Задание нужно сделать только в одной из сред программирования: IntelliJ[9], NetBeans[10] или в Eclips[11].

Пример, описанный в инструкции, нужно сделать обязательно, чтобы увидеть правильные результаты.

2. На втором этапе протестируем с помощью JUnit один из методов библиотеки calculator-1.0.jar.

При решении данной задачи необходимо использовать такие методы тест-дизайна как классы эквивалентности и граничные условия[12] [1]. Чтобы определиться с классами эквивалентности и граничными условиями в данной задаче проще всего использовать для методов с одним параметром числовые прямые, разбитые на участки, а при наличии двух параметров метода двумерные числовые плоскости с параметрами метода, отложенного по осям. Например, для тестирования метода сложения a + b создадим числовую плоскость, показанную на рисунке 2.1.

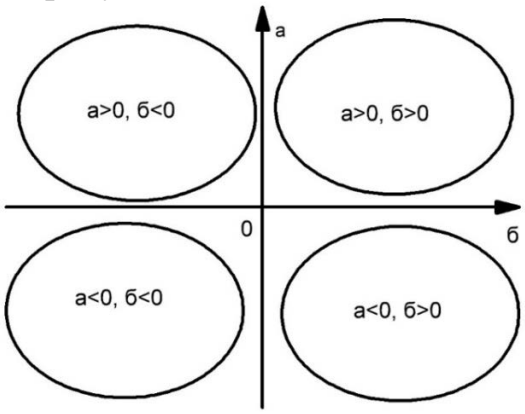

Рисунок 2.1 – Графическое изображение классов эквивалентности для методов с двумя параметрами

[9] Документ «Создание теста Junit в IntelliJ» в материалах лабораторной работы 2 курса «Тестирования программного обеспечения» виртуальной образовательной среды ВГТУ: https://sdo.vstu.by/course/view.php?id=728.

[10] Документ «Создание теста Junit в NetBeans» в материалах лабораторной работы 2 курса «Тестирования программного обеспечения» виртуальной образовательной среды ВГТУ: https://sdo.vstu.by/course/view.php?id=728.

[11] Документ «Создание теста Junit вEclips» в материалах лабораторной работы 2 курса «Тестирования программного обеспечения» виртуальной образовательной среды ВГТУ: https://sdo.vstu.by/course/view.php?id=728.

[12] Подробнее в главе 2.7.2. «Классы эквивалентности и граничные условия» (стр. 235) книги:

Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – 3-е изд. – Минск: Четыре четверти, 2020. – 312 с.

Таким образом, мы получаем четыре класса эквивалентности и граничные условия, находящиеся на границе классов эквивалентности.

Для максимального тестового покрытия данного метода необходимо создать четыре тестовых метода для проверки классов эквивалентности и пять тестовых методов для проверки граничных условий.

Задания выбрать согласно варианту из таблицы 2.1.

Таблица 2.1 – Варианты заданий

№ варианта	Тестируемый метод	№ варианта	Тестируемый метод
1	`sum(double, double)`	9	`pow(double, double)`
2	`sub(long, long)`	10	`sqrt(double)`
3	`sub(double, double)`	11	`tg(double)`
4	`mult(long, long)`	12	`ctg(double)`
5	`mult(double, double)`	13	`cos(double)`
6	`div(long, long)`	14	`sin(double)`
7	`isPositive(long)`	15	`isNegative(long)`
8	`div(double, double)`		

Содержание отчёта

Отчёт по лабораторной работе должен содержать:

- название лабораторной работы;
- ФИО студента и дату выполнения работы;
- цель работы;
- описание классов эквивалентности и граничные условия для тестирования библиотеки «калькулятор»;
- фрагменты исходного кода модульных тестов для тестирования библиотеки «калькулятор»;
- анализ успешных и неуспешных тестов с предоставлением скриншотов выполненных тестов;
- выводы по качеству кода на основе тестов.

В выводах необходимо отразить всё новое, что вы узнали, выполняя лабораторную работу. Например, что такое классы эквивалентности и граничные условия, *приведя примеры из собственного варианта*. Перечислять, что вы сделали в самой лабораторной работе, в выводе нет необходимости.

Контрольные вопросы

1. Что такое классы эквивалентности?
2. Что такое модульные тесты?
3. Что проверяют модульные тесты?
4. Какие элементы подвергаются проверкам в рамках модульного тестирования?

ЛАБОРАТОРНАЯ РАБОТА 3. РАЗРАБОТКА ПРОЕКТА НА JAVA ПОД УПРАВЛЕНИЕМ ТЕСТИРОВАНИЕМ

Цель работы: научиться разрабатывать проект в стиле объектно-ориентированного программирования на примере языка Java.

Краткие теоретические сведения

Разработка под управлением тестированием (Test-Driven Development, TDD)[13] – техника разработки ПО, опирающаяся на очень короткие повторяющиеся циклы, в которых написание тестов предшествует написанию кода.

Особенности модульных тестов. Модульные тесты всегда должны проходить на 100 %, т. к. в ином случае у нас есть ТОЧНО неработоспособный код, что ГАРАНТИРОВАННО приводит к дефектам в приложении. Также повторное успешное выполнение 100 % модульных тестов показывает, что в процессе доработок кода в нём не было ничего нарушено.

- Модульные тесты отделены от кода приложения на уровне структуры проекта.
- Модульные тесты независимы, отделены друг от друга, атомарны, просты.
- Модульные тесты часто пишутся программистами (т. к. часто требуется очень хорошо знать программирование, чтобы реализовать ту или иную логику).
- Модульные тесты могут писаться ДО кода приложения.

В рамках модульного тестирования проверкам подвергаются:

- метод;
- метод, который ничего не возвращает;
- класс;
- взаимодействие классов;
- геттеры и сеттеры;
- конструкторы;
- исключения.

Все эти варианты следует реализовать в данной лабораторной работе.

Порядок выполнения работы

Задание 1. В качестве примера рассмотрим задачу:

Дана окружность со следующими атрибутами: радиус, диаметр, длина окружности, площадь круга. Задано только одно значение этих элементов. Для остальных элементов окружности используются значения по умолчанию.

Разработаем класс окружность, содержащий:

1) поле радиус;

[13] Глава 4.4 «Разработка под управлением тестированием» (стр. 165) документа «Автоматизированное тестирование Святослав Куликов» материалов курса «Тестирования программного обеспечения» виртуальной образовательной среды ВГТУ: https://sdo.vstu.by/course/view.php?id=728.

2) конструктор, определяющий значение радиуса из заданного атрибута: радиус, диаметр, длина окружности, площадь круга;

3) геттер и сеттер для радиуса;

4) методы вычисления атрибута: радиус, диаметр, длина окружности, площадь круга;

5) методы заполнения поля радиус при вводе нового атрибута: радиус, диаметр, длина окружности, площадь круга.

Выполним тестирование разработанного класса. Подробно код тестов представлен в документе [14].

Задание 2. Разработать проект на языке Java в стиле ООП согласно варианту (табл. 3.1).

Протестировать разработанный проект. Поскольку модульные тесты покрывают атомарные участки кода, которыми в ООП являются методы, то в тестируемом проекте необходимо разработать не менее 5 методов.

В процессе разработки проекта необходимо покрыть модульными тестами все его компоненты.

Таблица 3.1 – Варианты задания

№ варианта	Задание для тестируемого проекта
1	2
1	Дан номер года (положительное целое число). Определить количество дней в этом году, учитывая, что обычный год насчитывает 365 дней, а високосный – 366 дней. Високосным считается год, делящийся на 4, за исключением тех годов, которые делятся на 100 и не делятся на 400 (например, годы 300, 1300 и 1900 не являются високосными, а 1200 и 2000 – являются).
2	Даны целочисленные координаты точки на плоскости. Если точка совпадает с началом координат, то вывести 0. Если точка не совпадает с началом координат, но лежит на оси OX или OY, то вывести соответственно 1 или 2. Если точка не лежит на координатных осях, то вывести 3.
3	Даны три переменные вещественного типа: A, B, C. Вывести их значения в порядке возрастания.
4	Даны координаты точки, не лежащей на координатных осях OX и OY. Определить номер координатной четверти, в которой находится данная точка.
5	Дано целое число. Вывести его строку-описание вида «отрицательное четное число», «нулевое число», «положительное нечетное число» и т. д.
6	Арифметические действия над числами пронумерованы следующим образом: 1 – сложение, 2 – вычитание, 3 – умножение, 4 – деление. Дан номер действия N (целое число в диапазоне 1–4) и вещественные числа A и B (B не равно 0). Выполнить над числами указанное действие и вывести результат.
7	Элементы равнобедренного прямоугольного треугольника пронумерованы следующим образом: 1 – катет a, 2 – гипотенуза $c = a \cdot 2^{1/2}$, 3 – высота h, опущенная на гипотенузу ($h = c/2$), 4 – площадь $S = c \cdot h/2$. Дан номер одного из этих элементов и его значение. Вывести значения остальных элементов данного треугольника (в том же порядке).

[14] Пример рассмотрен в документе «Тестирование с использованием Maven» в материалах лабораторной работы 3 курса «Тестирования программного обеспечения» виртуальной образовательной среды ВГТУ: https://sdo.vstu.by/course/view.php?id=728.

Окончание таблицы 3.1

1	2
8	Единицы длины пронумерованы следующим образом: 1 – дециметр, 2 – километр, 3 – метр, 4 – миллиметр, 5 – сантиметр. Дан номер единицы длины (целое число в диапазоне 1–5) и длина отрезка в этих единицах (вещественное число). Найти длину отрезка в метрах.
9	Даны целочисленные координаты точки на плоскости. Если точка совпадает с началом координат, то вывести на экран 0, если точка не совпадает с началом координат, но лежит на оси OX или OY, то вывести на экран 1. Если точка не лежит на координатных осях, то вывести на экран 2.
10	Локатор ориентирован на одну из сторон света (1 – север, 2 – запад, 3 – юг, 4 – восток) и может принимать три цифровые команды поворота: 1 – поворот налево, –1 – поворот направо, 2 – поворот на 180°. Дан символ C – исходная ориентация локатора и целые числа N1 и N2 – две посланные команды. Вывести ориентацию локатора после выполнения этих команд.
11	Даны целые числа a, b, c. Вывести на экран 0, если каждое из них отлично от другого, 1 – если только два числа одинаковы и 3 – если одинаковы все три числа.
12	Даны три вещественных числа: A, B, C. Если их значения упорядочены по возрастанию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное по знаку. Вывести на экран новые значения переменных A, B, C.
13	Даны целые числа a, b, c, являющиеся сторонами некоторого треугольника. Вывести на экран заключение о том, каким является этот треугольник (равносторонним, равнобедренным или обычным).
14	Даны целые числа a, b, c. Вывести на экран 0, если каждое из них отлично от другого, 1 – если только два числа одинаковы и 3 – если одинаковы все три числа.
15	Тригонометрические функции над числами пронумерованы следующим образом: 1 – синус, 2 – косинус, 3 – тангенс, 4 – котангенс угла. Дан номер действия N (целое число в диапазоне 1–4) и вещественное число A – значение угла. Выполнить над числами указанное действие и вывести результат.

Содержание отчёта

Отчёт по лабораторной работе должен содержать:

- название лабораторной работы;
- ФИО студента и дату выполнения работы;
- цель работы;
- постановка задачи (в соответствии с вариантом).
- разработка проекта (исходный код проекта):
  - реализация классов и методов;
  - геттеры, сеттеры, конструкторы;
- разработка модульных тестов (исходный код модульных тестов):
  - покрытие всех компонентов проекта тестами;
  - тестирование всех проверок: атомарные методы, обработка исключений, геттеры и сеттеры.
- результаты тестирования (скриншоты выполнения тестов):
  - анализ успешных и неуспешных тестов;

- обоснование исправлений и корректировок кода.

В выводах необходимо отразить всё новое, что вы узнали, выполняя лабораторную работу. Например, что может подвергаться проверкам в рамках модульного тестирования, *приведя примеры из собственного варианта*. Перечислять, что вы сделали в самой лабораторной работе, в выводе нет необходимости.

Контрольные вопросы

1. Что необходимо сделать, если в рамках модульного тестирования необходимо проверить действие, опирающееся на внешние источники или приёмники данных?
2. Нужно ли проверять, что операция создания экземпляра класса вернула экземпляр нужного класса?
3. Для чего используются аннотации в JUnit?
4. Как проверку assertEquals можно заменить проверкой assertTrue?
5. Имеет ли смысл проверять модульными тестами геттеры и сеттеры?

ЛАБОРАТОРНАЯ РАБОТА 4. ТЕСТИРОВАНИЕ ВНЕШНИХ ЗАВИСИМОСТЕЙ, ИСПОЛЬЗОВАНИЕ JMOCK.

Цель работы: изучить возможности тестирования в модульных тестах внешних зависимостей, используя для этого Mock-объекты.

Краткие теоретические сведения

Внешняя зависимость (external dependency) – нечто, что нам неподконтрольно в процессе выполнения тестов, например:

- база данных;
- веб-сервис;
- сторонние библиотеки.

Не имея возможности контролировать внешнюю зависимость, мы не можем гарантированно проверить поведение тестируемого приложения. Решение: «подменить» зависимость объектом, который мы можем контролировать.

Например, имеется приложение, схема которого представлена на рисунке 4.1.

Рисунок 4.1 – Исходные зависимости

Как мы проверим запрос, не имея доступа к СУБД?

Рисунок 4.2 – Решение проблемы

Создадим специальный объект, у которого мы можем запросить ЛЮБЫЕ данные.

Mock-объекты (англ. mock – подделка) – это объекты, которые имеют тот же интерфейс, что и компоненты приложения, но полностью управляемые из теста.

Их использование позволяет избежать использования полной инфраструктуры, необходимой приложению для запуска.

Что ещё более важно, можно проконтролировать, что код вызывал те или иные методы у mock-объекта с теми или иными аргументами.

Для реализации тестирования внешних зависимостей на языке Java используется фреймворк модульного тестирования ПО jMock, расширение JUnit.

Порядок выполнения работы

Задание 1. В качестве примера рассмотрим проект, разработанный в стиле ООП, содержащий класс автомобиль:

```
public class Car {
    private String marka;
    private int year;
    private String owner;

    public Car(String marka, int year, String owner) {
        this. marka = marka;
        this.year = year;
        this.owner = owner;
    }

    public String getMarka () {
        return marka;
    }

    public int getYear() {
        return year;
    }

    private String requestOwnerToCar(String owner){
        //запрос к БД которая не существует
        String car=request(owner);
        return car;
    }
```

```
    public int calcPowerToSpped (int pow){
        //Запрос к ресурсу который вычисляет скорость авто
        int sppe d= request(pow);
        return spped;
    }
}
```

Создадим новый тестовый класс CarTest и напишем тест для проверки конструктора:

```
public class CarTest {
    @Test
    void testCreateCar(){
        Car car = new Car("", 2014, "");// создать объект типа карт
        Car newCar = Mockito.mock(Car.class);
```

Вызываем метод для эмуляции и указываем тот класс-объект, какого типа мы хотим создать.

Теперь, используя геттеры для получения значений полей объекта, выполним проверку. Для строк по умолчанию будет значение null, для целого числа 0:

```
        assertEquals(null, newCar. getMarka());
        assertEquals(0, newCar.getYear());
    }
```

В следующем тесте сгенерируем на выходе метода не значение по умолчанию, а заданное нами значение. Для этого применим метод when:

```
    @Test
    void remoteServissReturnValue(){
        Car newCar = Mockito.mock(Car.class);
        when(newCar. calcPowerToSpped (120)).thenReturn(250);
        assertEquals(120, newCar. calcPowerToSpped (250));
    }

    @Test
    void getOwner(){
        when(newCar.getOwner()).thenReturn("Mercedes");
        assertEquals("Mercedes", newCar.getOwner());
    }
```

Данный тест мы используем, когда общаемся с удалённым сервисом.

Теперь проверим не только конечный результат, но и то, как мы к нему пришли.

Метод verify позволяет нам проверить вызывались ли какие-то

конкретные методы нашего объекта:

```
    @Test
    void verificationTest(){
        //assertEquals(null, newCar. getMarka());
        String Marka= newCar. getMarka();
        verify(newCar, only()).getManufacturer();
    }
}
```

Задание 2. Создать модульные тесты для тестирования внешних зависимостей. Объект для тестирования взять из лабораторной работы 3.

Выполнить проверку:

- конструктора;
- геттеров / сеттеров;
- метода возвращающего заданное значение.

Каждую проверку оформить в отдельном тесте!!!

Содержание отчёта

Отчёт по лабораторной работе должен содержать:

- название лабораторной работы;
- ФИО студента и дату выполнения работы;
- цель работы;
- тестовые сценарии (в виде исходных кодов тестов):
  - создание Mock-объектов для внешних зависимостей;
  - модульные тесты для тестирования внешних зависимостей;
- результат выполнения тестов (в виде скриншотов выполнения тестов):
  - анализ корректности тестов.

В выводах необходимо отразить всё новое, что вы узнали, выполняя лабораторную работу. Например, какие проверки вы использовали в Selenium IDE. Перечислять, что вы сделали в самой лабораторной работе, в выводе нет необходимости.

Контрольные вопросы

1. Какими рекомендуется создавать модульные тесты?
2. Можно ли с помощью jMock проэмулировать поведение класса, реализующего взаимодействие с внешней средой?
3. Можно ли проэмулировать некоторую сущность с помощью jMock, если уже написан реально существующий класс для этой сущности?
4. В каких условиях применять mock-обекты и писать модульные тесты проще?

ЛАБОРАТОРНАЯ РАБОТА 5. ТЕСТИРОВАНИЕ WEB-ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ SELENIUM IDE

Цель работы: изучить возможности функционального тестирования на примере автоматизированных тестов, разработанных с использованием Selenium IDE.

Краткие теоретические сведения

Функциональное тестирование программного обеспечения (ПО) представляет собой комплекс ключевых мероприятий по проверке программного обеспечения, по результатам которых устанавливается соответствие этого ПО исходным требованиям заказчика. Иными словами, с помощью проведения данных тестов устанавливается способность информационных систем в конкретных условиях решать пользовательские задачи.

Таким образом, *функциональное тестирование* – это вид тестирования, направленный на проверку корректности работы функциональности приложения.

В рамках функционального тестирования программного продукта проверяются следующие компоненты и критерии:

- пользовательский интерфейс;
- API;
- базы данных;
- безопасность систем;
- работа клиентских и серверных приложений;
- общая функциональность.

Такое тестирование можно проводить как в ручном, так и в автоматическом режиме.

В лабораторной работе рассмотрим автоматизированное создание тестов с использованием Selenium IDE[15].

Selenium IDE – это расширение для браузера, которое записывает и воспроизводит действия пользователя. Тесты записываются, как запись видео, а воспроизведение записанных тестов аналогично воспроизведению видео. Является библиотекой Selenium с графическим интерфейсом и возможностями для работы со сценариями тестирования веб-страниц. IDE генерирует код Selenium WebDriver, который повторяет записанные действия пользователей.

Порядок выполнения работы

Задание 1. Выполним тестирование сайта кафедры ИСиТ[16]:

[15] Подробное описание в главе 7.2.2 «Использование Selenium IDE» (стр. 252) документа «Автоматизированное тестирование. Святослав Куликов» материалов курса «Тестирования программного обеспечения» виртуальной образовательной среды ВГТУ: https://sdo.vstu.by/course/view.php?id=728.

[16] Пример использования Selenium IDE приведен в видео «Selenium IDE» в материалах лабораторной работы 5 курса «Тестирования программного обеспечения» виртуальной образовательной среды ВГТУ: https://sdo.vstu.by/course/view.php?id=728.

1. Запустим плагин Selenium IDE.
2. Создадим новый тест в новом проекте.
3. Зададим имя проекта «isit.vstu.by».
4. Зададим базовый URL «https://isit.vstu.by/».
5. Запустим запись теста.
6. Проверим, туда ли мы попали, проверяя заголовок страницы.
7. Протестируем поисковый сервис:
    a. Введём в строке поиска «Соколова».
    b. Запустим поиск.
    c. В открывшемся окне «Результаты поиска» найдем текст «Соколова Анна Сергеевна».
8. Остановим запись и откажемся создавать новый тест.

В окне плагина Selenium IDE, как показано на рисунке 5.1, представлены команды для тестирования[17], но прежде чем запустить тест на выполнение, уберем лишние команды. Selenium IDE записывает любое действие, в том числе и некоторые, не нужные нам.

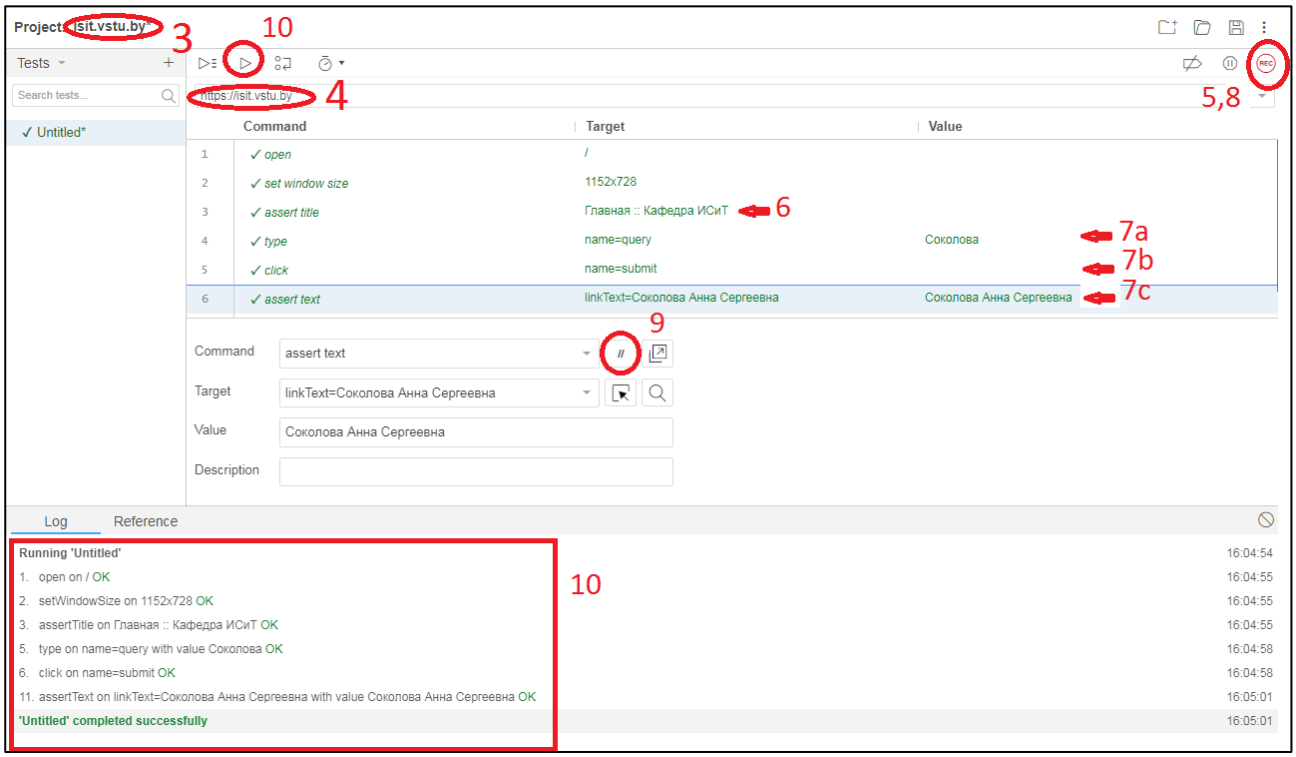

Рисунок 5.1 – Окно плагина Selenium IDE

9. Закомментируем лишние команды, а позже их удалим.
10. Запустим тест.
11. С результатами прохождения теста можно ознакомиться на вкладке «Log».

Задание 2. Выполнить функциональное тестирование объектов согласно варианту (табл. 5.1).

[17] Информацию о команде можно почитать на вкладке «Reference».

Выполнить проверку:

- работы ссылок;
- полей ввода и поиска;
- наличие различных элементов на странице.

Каждую проверку оформить в отдельном тесте!!!

Количество тестов не менее десяти.

Таблица 5.1 – Варианты задания

№ варианта	Объект для тестирования 1	Объект для тестирования 2
1	at.vstu.by	brsm.vstu.by
2	tm.vstu.by	cit.vstu.by
3	te.vstu.by	uo.vstu.by
4	miit.vstu.by	sport.vstu.by
5	mif.vstu.by	abiturient.vstu.by
6	eeb.vstu.by	ovr.vstu.by
7	kito.vstu.by	nic.vstu.by
8	trt.vstu.by	vestnik.vstu.by
9	eicht.vstu.by	stud.vstu.by
10	design.vstu.by/	library.vstu.by
11	forlang.vstu.by	zf.vstu.by
12	fis.vstu.by	fd.vstu.by
13	sgd.vstu.by/	fpt.vstu.by
14	fitr.vstu.by	mat-tech.vstu.by
15	ef.vstu.by	ic.vstu.by

Содержание отчёта

Отчёт по лабораторной работе должен содержать:

- название лабораторной работы;
- ФИО студента и дату выполнения работы;
- цель работы;
- запись созданных автоматизированных тестов (в виде скриншотов):
    - настройка параметров тестов;
    - редактирование и повторное использование тестов;
    - проверка работы тестов.

В выводах необходимо отразить всё новое, что вы узнали, выполняя лабораторную работу. Например, какие проверки вы использовали в Selenium IDE. Перечислять, что вы сделали в самой лабораторной работе, в выводе нет необходимости.

Контрольные вопросы

1. Как работает Selenium IDE?
2. Что можно записать с помощью технологии Record & Playback в Selenium IDE?
3. В каком поле записывается указание на объект, с которым выполняется действие в Selenium IDE?
4. В чём заключается разница между проверками assert и verify в Selenium IDE?

ЛАБОРАТОРНАЯ РАБОТА 6. АВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ SELENIUM WEBDRIVER НА ЯЗЫКЕ JAVA

Цель работы: изучить возможности функционального тестирования на примере автоматизированных тестов, разработанных с использованием Selenium WebDriver.

Краткие теоретические сведения

Selenium WebDriver – популярный инструмент для управления реальным браузером, который можно использовать как для автоматизации тестирования веб-приложений, так и для выполнения других рутинных задач, связанных с работой в вебе.

Это фреймворк для управления браузерами, основной продукт комплекта Selenium. Представляет из себя семейство драйверов для разных браузеров (Firefox, Edge, Google Chrome/Chromium, Internet Explorer, Safari, Opera) и набор клиентских библиотек на разных языках программирования для работы с драйверами. WebDriver поддерживает работу с языками Java, .Net (C#), Python, Ruby, JavaScript.

WebDriver напрямую отправляет команды браузеру, используя его API и получает результаты тестирования. WebDriver же использует способ взаимодействия с браузером, максимально близкий к действиям обычного пользователя.

Поскольку Selenium WebDriver представляет собой фреймворк, для его использования нужен язык программирования, на котором будем писать тесты, и среда разработки для этого языка. В качестве языка программирования выберем язык Java, а среду разработки будем использовать IntelliJ IDE.

Порядок выполнения работы

Задание 1. Управление браузером Google Chrome[18].

Решение этой задачи будет состоять из следующих шагов:

1. Создадим консольный проект.
2. Подключим необходимый фреймворк (selenium-server-4.15.0.jar).
3. Текст программы возьмём из файла, полученного в результате экспорта из Selenium IDE.
4. Из файла экспорта возьмём только команды открытия браузера и вывод заголовка страницы на консоль.

Листинг программы управления браузером Google Chrome:

```
//драйвер браузера Google Chrome
import org.openqa.selenium.chrome.ChromeDriver;
// экземпляр браузера Google Chrome
import org.openqa.selenium.WebDriver;
```

[18] Пример представлен в документе «Автотест Selenium Webdriver на Java» в материалах лабораторной работы 6 курса «Тестирования программного обеспечения» виртуальной образовательной среды ВГТУ: https://sdo.vstu.by/course/view.php?id=728.

```
public class Main {
    public static void main(String[] args) {
        WebDriver driver;
        // создаем экземпляр браузера
        driver = new ChromeDriver();
        // открываем заданную страницу
        driver.get("https://isit.vstu.by/");
        // выводим в консоль заголовок
        System.out.println(driver.getTitle());
    }
}
```

Данная задача доказала, что мы можем управлять браузером!!!

Задание 2. Предоставить приложение, выполняющее с помощью фреймворка Selenium WebDriver открытие в браузере веб-страницы согласно варианту (таблица 5.1) и вывести в консоль её заголовок.

Задание 3. Предоставить приложение, выполняющее функциональные автотесты с использованием библиотеки JUnion и фреймворка Selenium WebDriver.

Решение этой задачи должно состоять из следующих шагов:

1. Создадим консольный проект.
2. Подключим необходимый (selenium-server-4.15.0.jar, junit-4.12.jar, hamcrest-core-1.3.jar).
3. Текст программы возьмём из файлов, полученных в результате экспорта из Selenium IDE.
4. Берем только те строки, которые будут задействованы в том или ином тесте. В нашем случае это проверка поиска и проверка ссылки «сотрудники».

Пример листинга тестового класса для функциональных автотестов проверки поиска и ссылки[19]:

```
public class isitTest {
    private WebDriver driver;

    // запускается только один раз при запуске класса:
    @BeforeClass
    public void setUpClass() {
        driver = new ChromeDriver();
        driver.get("https://isit.vstu.by/");
    }

    // запускается после того, как отработали все тестовые методы:
    @AfterClass
    public void tearDownClass() {
        driver.quit();
```

[19] Более подробно об использовании JUnit можно прочитать в документе «Учебник JUnit» в материалах лабораторной работы 6 курса «Тестирования программного обеспечения» виртуальной образовательной среды ВГТУ: https://sdo.vstu.by/course/view.php?id=728.

```
    @Test
    public void searchTest() {
        // проверка заголовка страницы
        assertThat(driver.getTitle(), is("Главная :: Кафедра ИСиТ"));
        //ввод строки поиска
        driver.findElement(By.name("query")).sendKeys("Соколова");
        //нажатие кнопки поиск
        driver.findElement(By.name("submit")).click();
        //проверка заголовка страницы
        assertThat(driver.getTitle(), is("Результаты поиска :: Кафедра
ИСиТ"));
        //проверка текста
        assertThat(driver.findElement(By.linkText("Соколова Анна
Сергеевна")).getText(), is("Соколова Анна Сергеевна"));
    }

    @Test
    public void linkKafedraTest() {
        // проверка заголовка страницы
        assertThat(driver.getTitle(), is("Главная :: Кафедра ИСиТ"));
        // нажатие ссылки
        driver.findElement(By.linkText("Сотрудники")).click();
        // проверка текста
        assertThat(driver.findElement(By.cssSelector(".container:nth-
child(5) .ccm-block-testimonial-name")).getText(), is("Соколова Анна
Сергеевна"));
    }
}
```

Каждую проверку оформить в отдельном тесте!!!

Количество тестов не менее десяти.

Содержание отчета

Отчет по лабораторной работе должен содержать:

- название лабораторной работы;
- ФИО студента и дату выполнения работы;
- цель работы;
- описание добавления пакетов Selenium в проект;
- тестовый класс для проверки веб-ресурса согласно варианту:
    - исходный код автоматизированных тестов при взаимодействии с элементами веб-страниц (поиск, ввод текста, клики);
- отчет о проведенных тестах (скриншоты и примеры логов).

В выводах необходимо отразить всё новое, что вы узнали, выполняя лабораторную работу. Например, как выполняется поиск элементов на странице, какие методы используются для проверки и т.д. Перечислять, что вы сделали в самой лабораторной работе, в выводе нет необходимости.

Контрольные вопросы

1. Как перенести тесты из Selenium IDE в Selenium Web Driver?
2. Можно ли с помощью Selenium Web Driver провести тестирование приложения без использования настоящего браузера?
3. Что рекомендуется использовать в качестве-локаторов в Selenium Web Driver?
4. Позволяет ли Selenium Web Driver выполнять тесты Selenium IDE?
5. Гарантирует ли Selenium Web Driver то, что тесты будут одинаково выполняться в разных браузерах?

ЛАБОРАТОРНАЯ РАБОТА 7. АВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ SELENIUM WEBDRIVER НА ЯЗЫКЕ C#

Цель работы: изучить возможности функционального тестирования на примере автоматизированных тестов, разработанных с использованием Selenium WebDriver.

Краткие теоретические сведения

Фреймворк Selenium WebDriver поддерживает разные языки программирования такие как Java, C#, PHP, Ruby, Perl, Python.

C# – объектно-ориентированный язык программирования, созданный на основе C++ и Java. C# позволяет разработчикам создавать приложения с использованием Visual Studio на платформе .Net.

Visual Studio – это интегрированная среда разработки (IDE), которая используется разработчиками для создания приложений на различных платформах, таких как Windows, Android, iOS и облачные приложения.

NUnit – это фреймворк для модульного тестирования, поддерживаемый Visual Studio и Selenium WebDriver. NUnit – наиболее широко используемый фреймворк модульного тестирования для приложений .Net. NUnit представляет результаты тестирования в удобном для чтения формате и позволяет тестировщику отлаживать автоматизированные тесты.

Интеграция Selenium с платформой NUnit позволяет тестировщику различать различные классы тестов. NUnit также позволяет тестировщикам использовать аннотации, такие как SetUp, Test и TearDown, для выполнения действий до и после запуска теста.

Фреймворк NUnit можно интегрировать с Selenium, создав тестовый класс NUnit и запустив его с использованием платформы NUnit.

Порядок выполнения работы

Работа состоит в управлении браузером FireFox[20] и состоит из следующих

[20] Пример представлен в документе «Автотест Selenium Webdriver на C#» в материалах лабораторной работы 7 курса «Тестирования программного обеспечения» виртуальной образовательной среды ВГТУ: https://sdo.vstu.by/course/view.php?id=728.

шагов:

1. Создадим консольное приложение (.NET Framework).
2. Текст программы возьмём из файла, полученного в результате экспорта из Selenium IDE.
3. Выполним экспорт тестов, разработанных в предыдущей лабораторной работе, выполняющих проверки:

- ссылок;
- полей ввода и поиска;
- наличие элементов на странице.

Содержание отчёта

Отчет по лабораторной работе должен содержать:

- название лабораторной работы;
- ФИО студента и дату выполнения работы;
- цель работы;
- тестовый класс для проверки веб-ресурса согласно варианту:
  - исходный код автоматизированных тестов при взаимодействии с элементами веб-страниц (поиск, ввод текста, клики);
- отчет о проведенных тестах (скриншоты и примеры логов).

В выводах необходимо отразить всё новое, что вы узнали, выполняя лабораторную работу. Например, какие отличия в тестах, написанных на Java и на C#. Перечислять что вы сделали в самой лабораторной работе в выводе нет необходимости.

Контрольные вопросы

1. Как и для чего нужна конструкция using?
2. Какие библиотеки нужны для модульного тестирования на C#?
3. Какие команды используются для проверок в модульных тестах?
4. Назовите порядок команд перехода по ссылкам.

ЛАБОРАТОРНАЯ РАБОТА 8. ПОИСК И ДОКУМЕНТИРОВАНИЕ ДЕФЕКТОВ

Цель работы: освоить навыки выявления дефектов программного обеспечения и их систематического документирования. Освоить оформление отчётов о дефектах таким образом, чтобы они были полезны для разработчиков при исправлении ошибок.

Краткие теоретические сведения

Дефекты, обнаруженные в процессе тестирования, должны быть корректно и понятно документированы, так чтобы разработчик мог быстро воспроизвести и устранить данный дефект. Описание дефекта (баг-репорт) можно сохранить в специализированной багтрэкинговой системе (например,

JIRA, Bugzilla, Mantis, Redmine и др.) или в табличном виде[21].

Упрощённые определения:

Дефект – расхождение ожидаемого и фактического результата.

Ожидаемый результат – поведение системы, описанное в требованиях.

Фактический результат – поведение системы, наблюдаемое в процессе тестирования.

Дефекты могут встречаться в любой документации, в архитектуре и дизайне, в коде программы и т. д.

Отчёт о дефекте – документ, описывающий и определяющий важность обнаруженного дефекта, а также содействующий его устранению.

Цели создания отчёта о дефекте:

- предоставить информацию о проблеме;
- определить важность проблемы;
- содействовать устранению проблемы.

Основная цель написания отчёта о дефекте – устранение дефекта. Поэтому стоит помнить, что хороший тестировщик – не тот, кто написал за день 1000 бесполезных и бессмысленных отчётов, а тот, по чьим отчётам (вне зависимости от их количества) было исправлено большое количество дефектов.

Универсальная формула отчёта о дефекте состоит из трёх простых пунктов:

1. Что сделали (шаги воспроизведения).
2. Что получили (фактический результат).
3. Что ожидали получить (ожидаемый результат).

Формула «Что сделали? Что получили? Что ожидали?» хороша по многим причинам:

- прозрачность: нельзя отклониться в повествовательный стиль и/или написать что-то, отличное от отчёта о дефекте;
- легко верифицировать дефект;
- чётко (ещё до воспроизведения) видно, является ли описанное дефектом;
- избавление от лишней бессмысленной коммуникации;
- этой форме легко обучить «несмышлёных пользователей».

Порядок выполнения работы

Необходимо на заданном преподавателем ресурсе найти не менее 5 дефектов и оформить отчеты о них.

В качестве примера составим баг-репорт описывающий дефект, позволяющий выполнить регистрацию при загрузке в поле «Фотография» файлов недопустимого формата (.exe, .zip и другие).

Атрибуты отчета о дефекте:

- *Идентификатор*: уникальное значение с осмысленным

[21] Подробнее в главе 2.5.3 «Атрибуты (поля) отчёта о дефекте» (стр. 175) книги:

Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – 3-е изд. – Минск: Четыре четверти, 2020. – 312 с.

компонентом.

- *Краткое описание:* состоит из ответов на вопросы «Что произошло?», «Где это произошло?», «При каких условиях это произошло?».
- *Подробное описание:* информация о дефекте в развёрнутом виде.
- *Шаги по воспроизведению*: описывают действия, которые необходимо выполнить для воспроизведения дефекта.
- *Воспроизводимость:* показывает, при каждом ли прохождении по шагам воспроизведения дефекта удаётся вызвать его проявление.
- *Важность:* показывает степень ущерба, который наносится проекту существованием дефекта.
- *Срочность:* показывает, как быстро дефект должен быть устранён.
- *Симптом:* позволяет классифицировать дефекты по их типичному проявлению.
- *Возможность обойти:* показывает, существует ли альтернативная последовательность действий.
- *Комментарий:* может содержать любые полезные для понимания.
- *Приложения:* список прикреплённых к отчёту о дефекте приложений.

Результаты работы сведём в таблицу 8.1.

Таблица 8.1 – Отчёт о дефекте

Атрибут отчёта о дефекте	Содержание
Идентификатор	megaform_1
Краткое описание	Регистрация проходит успешно, когда в поле «Фотография» формы есть возможность загружать файлы разного формата
Подробное описание	Когда все необходимые поля в форме регистрации заполнены, есть возможность загружать файлы разного формата (.exe, .zip и другие) в поле «Фотография». При этом регистрация проходит успешно. *Ожидаемый результат*: сообщение или ошибка. *Фактический результат*: загрузка файлов. *Требование*: –
Шаги по воспроизведению	1. Открыть страницу http://svyatoslav.biz/testlab/megaform.php. 2. Заполните все необходимые поля. 3. Загрузить файл с расширением exe в поле «Фотография». 4. Нажать кнопку «Регистрация». *Дефект*: ожидалось, что система отреагирует на файл, не содержащий фотографию сообщением или ошибкой
Воспроизводимость	Всегда
Важность	Высокая
Срочность	Обычная
Симптом	Неожиданное поведение
Возможность обойти	Нет
Комментарий	Это рискованно для безопасности и бизнеса, как упущенная выгода
Приложения	C:\Temp\ Screenshot 2024-09-26.png

Содержание отчета

Отчет по лабораторной работе должен содержать:

- название лабораторной работы;
- ФИО студента и дату выполнения работы;
- цель работы;
- пошаговое описание процесса обнаружения дефектов;
- зафиксированные в виде скриншотов найденные дефекты;
- отчёты о дефектах (Bug Report) в форме таблицы.

В выводах необходимо отразить всё новое, что вы узнали, выполняя лабораторную работу. Например, что порядок заполнения полей отчета о дефектах не совпадает с последовательностью из заполнения. Перечислять, что вы сделали в самой лабораторной работе, в выводе нет необходимости.

Контрольные вопросы

1. Что такое дефект?
2. Для чего служит отчёт о дефекте?
3. В каком поле отчёта о дефекте располагается ответ на вопросы «Что? Где? При каких условиях?»?
4. Что такое важность и срочность дефекта?
5. Что и где нужно указать в отчёте о дефекте, если некорректная операция не позволяет продолжить дальнейшее тестирование?

ЛИТЕРАТУРА

1. Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – 3-е изд. – Минск: Четыре четверти, 2020. – 312 с.

2. Лабун, Б. Дружеское знакомство с тестированием программ: пер. с англ. – СПб.: БХВ-Петербург, 2022. – 288 с.

3. Назина, О. Е. Что такое тестирование. Курс молодого бойца. – СПб.: БХВ-Петербург, 2022. – 592 с.

4. Савин, Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах / Р. Савин. – М.: Дело, 2007. – 312 с.

Учебное издание

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические указания
по выполнению лабораторных работ

Составители:
Черненко Дмитрий Владимирович
Соколова Анна Сергеевна

Редактор *Р.А. Никифорова*
Корректор *А.С. Прокопюк*
Компьютерная верстка *А.С. Соколова*

Подписано к печати 12.05.2025. Усл. печ. листов 1,9.
Уч.-изд. листов 2,3. Заказ № 97.

Учреждение образования «Витебский государственный технологический университет»
210038, г. Витебск, Московский пр., 72.
Отпечатано на ризографе учреждения образования
«Витебский государственный технологический университет».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/172 от 12 февраля 2014 г.
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 3/1497 от 30 мая 2017 г.