МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

Утверждаю:
Зав. кафедрой АОИ
профессор
_____Ю.П.Ехлаков
«___»_____2017 г.

Методические указания для выполнения лабораторных работ и организации самостоятельной работы по дисциплине «Тестирование программного обеспечения»

для студентов заочной формы обучения направления подготовки
09.03.04 «Программная инженерия»

Разработчик:
_____ Ю.В. Морозова

Томск – 2017

Содержание

Введение

Лабораторные работы предназначены для бакалавров заочной формы обучения направления подготовки 09.03.04 «Программная инженерия», изучающих дисциплину «Тестирование программного обеспечения».

Цели освоения дисциплины имеет своей целью обучение базовым знаниям по организации процесса тестирования и отладки программных продуктов с использованием современных технологий и подходов.

Для достижения поставленной цели выделяются задачи курса:

- Дать представление об основных понятиях тестирования: терминология тестирования, различия тестирования и отладки, фазы и технология тестирования, проблемы тестирования.
- Провести обзор современных критериев выбора тестов и оценки покрытия проекта.
- Обсудить разновидности тестирования: модульное, интеграционное, системное, регрессионное, автоматизация тестирования, издержки тестирования.
- Указать особенности процесса и технологии тестирования: планирование тестирования, подходы к разработке тестов, особенности ручной разработки и генерации тестов, автоматизация тестового цикла, документирование тестирования.
- Рассмотреть особенности и виды тестирования, методы отбора тестов, оценка эффективности.
- Дать представление о терминологии тестирования в соответствии с IEEE Standard Glossary of Software Engineering.

Пособие содержит 2 лабораторные работы с описанием выполнения типового задания.

Правила выполнения лабораторных работ (заданий)

В ходе выполнения лабораторной работы студент должен строго выполнять весь объем самостоятельной подготовки, указанный в описаниях соответствующих лабораторных работ. Выполнению каждой работы предшествует проверка готовности студента, которая проводится преподавателем.

Лабораторные занятия выполняются студентами самостоятельно, преподаватель в ходе занятия осуществляет научное и методическое руководство действиями студентов.

После выполнения работы студент должен представить отчет о проделанной работе с обсуждением полученных результатов и выводов.

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

Темы лабораторных работ

1. Функциональное тестирование.
2. Нефункциональное тестирование.

Отчет по лабораторной работе должен содержать:

1. Тему и цель лабораторной работы.
2. Вариант задания на лабораторную работу.
3. Краткие теоретические сведения и описание алгоритма работы программы.
4. Листинг разработанной программы с подробными комментариями.
5. Результаты работы программы.
6. Выводы.

Текст выполняется на листах формата А4 (210х297 мм) по ГОСТ 2.301 с применением печатающих устройств вывода ЭВМ (ГОСТ 2.004). На компьютере текст должен быть оформлен в текстовом редакторе Microsoft Word.

Текст работы выполняется на листах формата А4, без рамки, с соблюдением следующих размеров полей:

а) левое – не менее 30 мм;

б) правое – не менее 10 мм;

в) верхнее и нижнее – не менее 20 мм.

Страницы следует нумеровать арабскими цифрами, соблюдая сквозную нумерацию по всему тексту.

Номер страницы проставляют в центре нижней части листа без точки.

Тип шрифта: Times New Roman.

Шрифт основного текста – обычный, размер 14 пт.

Шрифт заголовков разделов, структурных элементов «Аннотация», «Содержание», «Введение», «Заключение», «Список использованных источников», «Приложение» – полужирный, размер 16 пт.

Шрифт заголовков подразделов – полужирный, размер 14 пт.

Межсимвольный интервал – обычный.

Межстрочный интервал – одинарный. Выравнивание текста по ширине.

Объем работы должен составлять не менее 25 страниц основной части. Изложение должно быть последовательным, логичным, конкретным.

Первая страница – титульный лист, вторая – задание, далее – аннотация, оглавление и текст (номера первых двух страниц не указываются). Оглавление создается автоматически средствами текстового редактора.

Документ может содержать таблицы, рисунки и формулы. На каждую таблицу, рисунок, формулу должна быть ссылка в тексте.

В текст пояснительно записки могут быть включены небольшие фрагменты программного кода, обязательно с комментариями. Рекомендуемый шрифт для выполнения фрагмента кода – Courier New, размер 12пт.

На материалы, взятые из литературы и других источников должны быть даны ссылки с указанием номера источника по списку использованной литературы.

Список составляется в порядке появления ссылок.

В приложениях размещаются листинг, схемы программы, скрины интерфейса. Приложения нумеруются русскими буквами в порядке появления ссылок на них в основном тесте документа.

Лабораторная работа № 1

Функциональное тестирование

Количество аудиторных часов – 4

Цель работы: Овладение навыками функционального тестирования.

Общие сведения:

Сегодня тестирование – это обязательная часть процесса разработки программного обеспечения (далее – ПО). Это связано с жесткими правилами конкуренции для компаний, производящих программные продукты (ПП).

Раньше таких компаний на рынке было мало и пользователи программных продуктов были продвинутыми и заменяли тестеров. Если в программе обнаруживались баги, то пользователь звонил или отправлял письмо в компанию, где ошибку исправляли и по почте отправляли дискетку со свежим релизом. Но начиная с 1990 года согласно статистики продажи персональных компьютеров с каждым годом удваивались. И появилась армия пользователей, которая не готова была что-то тестировать. Если что-то не устроило было проще обменять на другой софт, т.к. число компаний производящих ПО тоже увеличивалось с каждых готом. И у пользователей появился выбор что покупать и чем пользоваться.

Таким образом, тестирование ушло внутрь компаний, и появилась профессия тестировщика.

Рассмотрим определение, которое записано в SWEBOK.

Тестирование ПО – это проверка соответствия между реальным поведением программы и ее ожидаемым поведением на конечном наборе тестов, выбранном определенным образом. [IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004].

Все виды тестирования можно условно разделить на две большие группы:

1. Статическое тестирование (*static testing*).
2. Динамическое тестирование (*dynamic testing*).

Статическое тестирование – это процесс анализа самой разработки программного обеспечения, т. е. тестирование без запуска программы.

К данной группе можно отнести анализ кода. Данный вид тестирования осуществляется в основном программистами. Проводят тестирование артефактов разработки программного обеспечения, таких как требования, дизайн или программный код, проводимое без исполнения этих артефактов. Например, с помощью рецензирования или статического анализа.

Статический анализ кода (static code analysis) – это анализ исходного кода, производимый без его исполнения.

Динамическое тестирование – это тестовая деятельность, предусматривающая эксплуатацию (запуск) программного продукта.

Динамическое тестирование предполагает запуск программы, выполнение всех ее функциональных модулей и сравнение фактического ее поведения с ожидаемым.

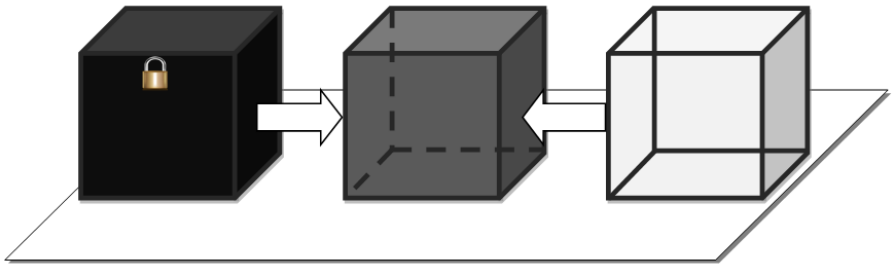
Статическое тестирование позволяет обнаружить дефекты, которые являются результатом ошибки и привести к сбоям в программном обеспечении. Динамическое тестирование позволяет продемонстрировать непосредственно сбои в программном обеспечении.

Существует несколько признаков, по которым принято производить классификацию видов тестирования.

По знанию системы выделяют:

- тестирование «черного ящика» (black box testing);
- тестирование «белого ящика» (white box testing);
- тестирование «серого ящика» (grey box testing).

Метод чёрного ящика (black box testing, closed box testing) – у тестировщика либо нет доступа к внутренней структуре и коду приложения, либо недостаточно знаний для их понимания, либо он сознательно не обращается к ним в процессе тестирования.



Разработка тестов методом черного ящика (black box test design technique)

Процедура создания и/или выбора тестовых сценариев, основанная на анализе функциональной или нефункциональной спецификации компонента или системы без знания внутренней структуры.

Техники разработки тестов на основе спецификаций, или методе черного ящика:

- эквивалентное разбиение;
- анализ граничных значений;
- тестирование таблицы решений;

Эквивалентное разбиение (equivalence partitioning)

Разработка тестов методом черного ящика, в которой тестовые сценарии создаются для проверки элементов эквивалентной области. Как правило, тестовые сценарии разрабатываются для покрытия каждой области как минимум один раз.

Входные данные для программного обеспечения или системы разбиваются на группы, от которых ожидается сходное поведение, то есть они должны обрабатываться аналогичным образом. Эквивалентные области (или классы) могут быть определены как для валидных, так и для невалидных данных, то есть тех значений, которые должны отвергаться.

Эквивалентное разбиение применимо на всех уровнях тестирования. Эквивалентное разбиение может быть использовано с целью покрытия входных и выходных данных. Оно может применяться при ручном вводе данных, при передаче данных через интерфейсы в систему, или при проверке параметров интерфейсов в интеграционном тестировании.

Анализ граничных значений (boundary value analysis): Разработка тестов методом черного ящика, при котором тестовые сценарии проектируются на основании граничных значений.

Граничное значение (boundary value): Входное значение или выходные данные, которое находится на грани эквивалентной области или на наименьшем расстоянии от обеих сторон грани, например, минимальное или максимальное значение области. Анализ граничных значений может применяться на всех уровнях тестирования.

Таблица решений (decision table): Таблица, отражающая комбинации входных данных и/или причин с соответствующими выходными данными и/или действиям (следствиям), которая может быть использована для проектирования тестовых сценариев.

Таблицы решений – хороший метод для сбора системных требований, содержащих логические условия и документирования внутреннего дизайна системы. Они могут использоваться для записи сложных бизнес-правил, которые должна реализовывать система. Анализируются спецификации и определяются условия и действия системы. Входные условия и действия чаще всего формулируются таким образом, чтобы они могли принимать логические значения «истина» или «ложь».

Сильной стороной тестирования таблицы решений является то, что она создает комбинации условий, которые могли бы быть не проверены в ходе тестирования иным способом. Этот метод может быть применен ко всем ситуациям, в которых действие программного продукта зависит от нескольких логических альтернатив.

Метод белого ящика (white box testing, open box testing, clear box testing, glass box testing) – у тестировщика есть доступ к внутренней структуре и коду приложения, а также есть достаточно знаний для понимания увиденного.

Разработка тестов методом белого ящика (white-box test design technique): Процедура разработки или выбора тестовых сценариев на основании анализа внутренней структуры компонента или системы.

Техники, основанные на структуре, или методе белого ящика

- тестирование операторов;
- тестирование альтернатив.

Альтернатива (decision): Точка программы, в которой управление имеет два или более альтернативных путей. Узел с двумя или более связями для разделения ветвей.

Тестирование условий альтернатив (decision condition testing): Разработка тестов методом белого ящика, при котором тестовые сценарии проектируются для исходов условий и результатов альтернатив.

Покрытие (coverage): Уровень, выражаемый в процентах, на который определенный элемент покрытия был проверен набором тестов.

Покрытие альтернатив (decision coverage): Процент результатов альтернативы, который был проверен набором тестов. Стопроцентное покрытие решений подразумевает стопроцентное покрытие ветвей и стопроцентное покрытие операторов.

Покрытие кода (code coverage): Метод анализа, определяющий, какие части программного обеспечения были проверены (покрыты) набором тестов, а какие нет, например, покрытие операторов, покрытие альтернатив или покрытие условий.
Еще выделяют серый ящик.

Метод серого ящика сочетает преимущества и недостатки методов белого и чёрного ящика.

- *с одной стороны,* тестирование, ориентированное на пользователя, а значит, мы используем имитацию поведения пользователя, т.е. **применяем методику "Черного ящика";**
- *с другой* — **информированное тестирование,** т.е. мы знаем, как устроена хотя бы часть тестируемого ПО, и активно **используем** это знание.

По объекту тестирования различают:

- функциональное тестирование;
- нефункциональное тестирование.

Функциональное тестирование (functional testing) проводится с целью выявления ошибок в функционировании программы путем сопоставления фактического (actualt) и ожидаемого (expected) результатов. Для получения фактического результата программу надо выполнить. Основным источником ожидаемого результата являются спецификации функциональных требований (requirements) к программному продукту.

Цель модульного тестирования состоит в выявлении локализованных в модуле ошибок в реализации алгоритмов, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования. Функциональное тестирование проводится по принципу «белого ящика», то есть основывается на знании внутренней структуры программы, и часто включает те или иные методы анализа покрытия кода.

При функциональном тестировании выявляются следующие категории ошибок:

- некорректность или отсутствие функций;
- ошибки интерфейса;
- ошибки в структурах данных;
- ошибки машинных характеристик (нехватка памяти и др.);
- ошибки инициализации и завершения.

Функциональные тесты основываются на функциях, выполняемых системой, и могут проводиться на всех уровнях тестирования (*модульном, интеграционном, системном, приемочном*).

Каждому процессу разработки соответствует свой уровень тестирования.

Модульное тестирование

Компонентное тестирование (также известное как модульное) занимается поиском дефектов и верификацией функционирования программных модулей, программ, объектов, классов и т.п., которые можно протестировать изолированно.

Компонент (component) – это наименьший элемент программного обеспечения, который может быть протестирован отдельно.

Это может быть сделано изолированно от остальной части системы, в зависимости от контекста ЖЦ разработки и системы. В процессе могут быть использованы заглушки, драйвера и эмуляторы.

Компонентное тестирование может включать как тестирование функциональности и специфичных нефункциональных характеристик, таких как поведение ресурсов (например, поиск утечки памяти) или тестирование надежности, так и структурное тестирование или тестирование белым ящиком (например, покрытие кода).

Обычно, компонентное тестирование производится с доступом к тестируемому коду и с поддержкой рабочего окружения, такого как фреймворк модульного тестирования или утилиты отладки. На практике компонентное тестирование обычно производится разработчиками, которые пишут код. Дефекты обычно исправляются сразу после того, как становятся известны, без занесения их в базу дефектов.

Один из подходов к компонентному тестированию – составить автоматизированные тестовые сценарии до кодирования. Это называется разработкой через тестирование.

TDD (Test-driven development), разработка через тестирование – техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов разработки:

- сначала пишется тест, покрывающий желаемое изменение,
- затем пишется код, который позволит пройти тест,
- и под конец проводится рефакторинг нового кода к соответствующим стандартам.

Цикл разработки (кратко)

Красный – напишите небольшой тест, который не работает, а возможно, даже не компилируется.

Зеленый – заставьте тест работать как можно быстрее, при этом не думайте о правильности дизайна и чистоте кода. Напишите ровно столько кода, чтобы тест сработал.

Рефакторинг – удалите из написанного кода любое дублирование.

Разработка через тестирование ООП подразумевает следующие этапы, которые в 90% случаев идут строго по порядку(об оставшихся 10% скажу позднее):

1. пишется пустой класс A (релизный);
2. пишется класс ATest (тестовый) и в нем создаются заглушки всех тестовых методов(определяем весь функционал который нам нужен от класса A);
3. создаем заглушки методов в классе A, которые должны реализовывать весь необходимый функционал, который выявился на этапе 2;
4. если выявились недостающие тесты, то они декларируются, и реализуются все тестовые методы в классе ATest до конечной стадии. и прогоняем тесты. все тесты должны завершиться с ошибкой, за некоторым специфичным исключением(например, проверка наследования);
5. реализуем все методы в классе A, прогоняем тесты – все тесты должны завершиться успешно.

Тестовый метод должен быть коротким!

- Иногда полезно выносить дополнительные проверки во вспомогательные методы, чтобы улучшить читаемость теста
- Количество проверок (**assert**) должно быть минимальным. Иначе по падению теста сложно будет найти причину ошибки
- Каждый тест должен покрывать одну единицу логики.

Это может быть:

- Простой метод
- Один из исходов конструкции if..else
- Один из случаев (case) блока switch
- Исключение, обрабатываемое блоком try…catch
- Исключение, генерируемое (throw) в методе

Используйте аннотации:

- Аннотация *@Before* обозначает методы, которые будут вызваны до исполнения теста, методы должны быть *public void*. Здесь обычно размещаются предустановки для теста, в нашем случае это генерация тестовых данных (метод *setUpToHexStringData*).
- Аннотация *@BeforeClass* обозначает методы, которые будут вызваны до создания экземпляра тест-класса, методы должны быть *public static void*. Имеет смысл размещать предустановки для теста в случае, когда класс содержит несколько тестов использующих различные предустановки, либо когда несколько тестов используют одни и те же данные, чтобы не тратить время на их создание для каждого теста.
- Аннотация *@After* обозначает методы, которые будут вызваны после выполнения теста, методы должны быть *public void*. Здесь размещаются операции освобождения ресурсов после теста, в нашем случае — очистка тестовых данных (метод *tearDownToHexStringData*).
- Аннотация *@AfterClass* связана по смыслу с *@BeforeClass*, но выполняет методы после теста, как и в случае с *@BeforeClass*, методы должны быть *public static void*.
- Если вы хотите указать, что определенный тест необходимо пропустить, то пометьте его аннотацией *@Ignore*. Хотя можно просто удалить аннотацию *@Test*.
- Аннотация *@Test* обозначает тестовые методы. Как и ранее, эти методы должны быть *public void*. Здесь размещаются сами проверки. Кроме того, у данной аннотации есть два параметра, *expected* — задает ожидаемое исключение и *timeout* – задает время, по истечению которого тест считается провалившимся.

Таблица – Тестовые методы

Сигнатура	**Описание**
fail(String)	Вызывает сбой. Может быть использован, чтобы удостовериться, что определенная часть кода не достигнута или пока тестовый метод не реализован.
assertTrue(true) / assertTrue(false)	Постоянно будет true / false. Может быть использован,

	чтобы предопределить результат теста, пока он не реализован.
assertsEquals([String message], expected, actual)	Проверяет, равны ли две величины. Важно: для массивов проверяется ссылка а не содержимое.
assertsEquals([String message], expected, actual, tolerance)	Проверяет, совпадают ли величины float и double
assertNull([message], object)	Проверяет, что объект null.
assertNotNull([message], object)	Проверяет, что объект не null.
assertSame([String], expected, actual)	Проверяет, что обе переменные ссылаются на один объект.
assertNotSame([String], expected, actual)	Проверяет, что переменные ссылаются на разные объекты.
assertTrue([message], boolean condition)	Проверяет условие на истинность.

Пример:

Тестируемый метод:

```
/**
 * Реализация функций
 */
public class CustomMath {
public static int division(int x, int y){
        if (y == 0) { //если делитель равен нулю
                throw new IllegalArgumentException("divider is 0 ");
                }  //бросается исключение
        return x/y;  //возвращает результат деления
}
}
```

Модульные тесты:

```
//подключение библиотеки
import junit.framework.*;
//Тест должен быть наследником класса
TestCase
public class CustomMathTest extends TestCase {
//тестирующие методы начинаются с префикса “test”
        public void testDivision() {
        System.out.println("division");
        int x = 0; int y = 0; int expResult = 0;
        int result = CustomMath.division(x, y);
//проверка условия на совпадение
        assertEquals(expResult, result);
```

```
        fail("The test case is a prototype.");
} }
```

Запуск теста для метода division(x, y) выдал ошибку:

```
Testcase: testDivision(simplemathapp.CustomMathTest):
Caused an ERROR divider is null
java.lang.IllegalArgumentException: divider is null
at implemathapp.CustomMath.division(CustomMath.java:42)
at simplemathapp.CustomMathTest.testDivision(CustomMathTest.java:38)
```

Причина в том, что тест некорректно обрабатывает деление на ноль:

```
public void testDivision() {
System.out.println("division");
int x = 0; int y = 0; int expResult = 0;
int result = CustomMath.division(x, y);
assertEquals(expResult, result);
}
```

Вместо сравнения числовых результатов нужно ожидать исключение.

Обработка исключений

- В случае исключения мы не делаем ничего.
- Если исключения нет – сообщаем об ошибке.

```
public void testDivision() {
int x = 0; int y = 0; int expResult = 0;
        try {
                expResult = CustomMath.division(x, y);
                fail("Exception expected");
        } catch (IllegalArgumentException e){
System.out.println("pass");
}
}
```

Ключевые моменты

- Из-за того, что методы тестируются в изоляции друг от друга, связи между классами становятся более слабыми, что положительно влияет на архитектурный дизайн приложения и масштабируемость.
- Приложения разработанные через TDD обладают большей стабильностью.
- Тест-кэйсы могут предотвратить написание ненужного функционала.
- Начинающим разработчикам может показаться, что временные затраты на написание тестов слишком высоки и гораздо быстрее разработать фунционал не прибегая к TDD, но на практике — это не так. Все потому что процесс тестирования ПО неизбежен и гораздо проще его автоматизировать, нежели каждую итерацию приложения проверять вручную.

Рекомендуется использовать следующие инструменты:

- Среда разработки (IDE) – **Visual Studio (C#)**
- Фреймворк для модульного тестирования – **MSTest**

- IDE **NetBeans** для JUnit (https://netbeans.org/kb/docs/java/junit-intro_ru.html) и PHPUnit (https://netbeans.org/kb/docs/php/phpunit.html#installing-phpunit).
- Версия **JUnit 4.0** в IDE Eclipse.

Ссылки

- Сайт проекта **JUnit**: http://junit.sourceforge.net
- Сайт проекта **PHPUnit**: https://phpunit.de/
- Страничка модуля **NetBeans**: http://junit.netbeans.org, https://netbeans.org/index_ru.html
- Оболочка для модульного тестирования программ на C: http://cunit.sourceforge.net/

Интеграционное тестирование

Следующий уровень интеграционное тестирование:

Интеграционное тестирование – проводится после компонентного тестирования и направлено на выявление дефектов взаимодействия различных подсистем на уровне потоков управления и обмена данными.

Интеграционное тестирование – это тестирование части системы, состоящей из двух и более модулей. Основная задача интеграционного тестирования – поиск дефектов, связанных с ошибками в реализации и интерпретации интерфейсного взаимодействия между модулями. С технологической точки зрения интеграционное тестирование является количественным развитием модульного, поскольку так же, как и модульное тестирование, оперирует интерфейсами модулей и подсистем и требует создания тестового окружения, включая заглушки (**Stub**) на месте отсутствующих модулей.

Заглушка – это имитация вызываемой функции, возвращающая те же данные, но ничего больше не делающая.

Системное тестирование

Системное тестирование (system testing): Процесс тестирования системы в целом с целью проверки того, что она соответствует установленным требованиям.

Системное тестирование производится над проектом в целом с помощью метода «черного ящика». Структура программы не имеет никакого значения, для проверки доступны только входы и выходы, видимые пользователю. Тестированию подлежат коды и пользовательская документация. Системное тестирование чаще всего выполняет независимая тестовая команда.

Еще можно выделить **приёмочное тестирование (acceptance testing):** Формальное тестирование по отношению к потребностям, требованиям и бизнес процессам пользователя, проводимое с целью определения соответствия системы критериям приёмки и дать возможность пользователям, заказчикам или иным авторизированым лицам определить, принимать систему или нет. [Согласно IEEE 610]

Порядок выполнения работы

1) Разработать функцию в соответствии со своим вариантом.
2) Реализовать полученное задание, согласно технологии TDD.

3) Добиться 100% прохождения этих тестов. Описать принципы выбора тестов

4) Провести тестирование программы и представить результаты в виде таблицы.

5) Выработать рекомендации для корректировки тестируемой программы.

6) Представить отчет по лабораторной работе для защиты.

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

Лабораторная работа № 2

Нефункциональное тестирование

Количество аудиторных часов – 4.

Цель работы: провести нефункциональное тестирование и написать тестовую документацию.

Общие сведения: В отличии от **функционального** тестирования, целью которого является проверка соответствия реальных функций продукта с функциональными требованиями, как Вы уже наверное догадались, целью нефункционального тестирования является проверка соответствия свойств приложения с его нефункциональными требованиями.

Соответственно: **нефункциональное тестирование** – тестирование свойств, которые не относятся к функциональности системы. Данные свойства определяются нефункциональными требованиями, которые характеризуют продукт с таких сторон, как:

Надежность (реакция системы на непредвиденные ситуации).

Производительность (Работоспособность системы под разными нагрузками).

Удобство (Исследование удобности работы с приложением с точки зрения пользователя).

Масштабируемость (Требования к горизонтальному или вертикальному масштабированию приложения).

Безопасность (Защищенность пользовательских данных).

Портируемость (Переносимость приложения на различные платформы).

Данные свойства системы можно исследовать, используя следующие виды тестирования:

Тестирование установки (Installation testing) – проверка успешности установки приложения, его настройки и удаления. Снижает риски потери пользовательских данных, потери работоспособности приложения и пр.

Тестирование удобства использования (Usability testing) – характеризует систему с точки зрения удобства использования конечного пользователя.

Конфигурационное тестирование (или тестирование портируемости) – исследование работоспособности программной системы в условиях различных программных конфигураций.

Тестирование с разными браузерами (**кросс-браузерным тестированием** (cross-browser testing)).

Тестирование на отказ и восстановление (Failover and Recovery Testing) – исследование программной системы на предмет восстановления после ошибок, сбоев. Оценивание реакции защитных свойств приложения.

Необходимо учитывать особенности тестирования на мобильных устройств:

- Размер экрана и touch-интерфейс.
- Ресурсы устройства.
- Различные разрешения экрана и версии ОС.
- Реакция приложения на внешние прерывания.
- Удобство навигации по приложению.
- Тестирование вёрстки сайта.
- Отсутствие пустых экранов.
- Одновременное нажатие на все клавиши («защита от дурака»).
- Жесты предусмотренные функционалом.

Подготовка и порядок выполнения работы

1. Выбрать любой интернет-сайт и эмулятор (симулятор) мобильного устройства.
2. Провести нефункциональное тестирование этого сайта.
3. Найти дефекты в работе сайта, сравнив отображение на эмуляторе устройства и на ПК в различных браузерах.
4. Заполните тест-план и 10 тест-кейсов для тестирования сайта.

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

Руководство к выполнению самостоятельной работы

Согласно рабочей программе самостоятельная работа студентов заключается в следующем:

№	Наименование работы	Форма контроля
1	Подготовка к контрольным работам	Контрольный опрос на лекции
2	Проработка лекционного материала	Тестовый опрос на лекции
4	Изучение тем теоретической части дисциплины, вынесенных для самостоятельной проработки	Опрос на лекции
5	Подготовка к лабораторным работам	Программирование

Темы опросов на занятиях

- Исследовательское тестирование.
- Виды отчетностей и показателей.
- Гибкое тестирование. Нагрузочные испытания.
- Тестирование «серого» ящика.
- Минусы/плюсы автоматизации.
- Правила проведения регрессионного тестирования.
- Позитивное негативное и дымовое тестирование.

Зачёт

1. Каковы цели тестирования?
2. Почему нежелательно, чтобы тестированием занимались программисты?
3. Назовите 7 принципов тестирования и расшифруйте их значение.
4. Назовите и опишите уровни тестирования.
5. Перечислите известные вам виды и стратегии тестирования, опишите их (стратегий) основные характеристики.
6. Что такое дефект? Какие существуют виды дефектов (определения)?
7. Перечислите и поясните основные характеристики общих требований к качеству ПО.
8. Опишите ЖЦ дефекта.
9. Опишите схему, по которой должен быть описан дефект.
10. Перечислите и охарактеризуйте способы поиска дефектов.
11. Что такое функциональное тестирование?
12. Охарактеризуйте позитивное негативное и дымовое тестирование.
13. Какова схема действий в процессе тестирования? Опишите каждый этап.
14. Перечислите и охарактеризуйте известные вам методы проектирования тестов.
15. Что такое покрытие кода?
16. Что оценивает нефункциональное тестирование? Примеры (виды нефункционального тестирования).
17. Что такое регрессионное тестирование?
18. Причины возникновения повторных ошибок.
19. Типичные ошибки при проведении регрессионного тестирования.
20. Перечислите виды регрессионного тестирования.
21. Правила проведения регрессионного тестирования.
22. Что такое автоматизированное тестирование?
23. Минусы/плюсы автоматизации.
24. Цели автоматизации.

25. Каким проектам противопоказана автоматизация?
26. Порядок действий при проведении автоматизации.
27. Какие тесты – лучшие претенденты на автоматизацию?
28. Как выбирать инструменты на автоматизацию?
29. Что такое тест план, для чего пишется, назовите хотя бы некоторые пункты тест плана?
30. Что такое тест-кейсы, для чего пишутся?
31. Что такое чек-лист, для чего пишется?
32. Назовите наиболее распространенные тесты мобильной разработки.
33. Назовите ошибки при адаптации сайтов для мобильных устройств.

Темы контрольных работ

Особенности процесса и технологии тестирования.

Учебно-методическое и информационное обеспечение дисциплины

Основная литература

1. Михеева Е.Н. Управление качеством [Текст]: учебник для вузов. – М.: Дашков и К°, 2012. – 532 с. – Библиогр.: с. 481–487. Гриф. (наличие в библиотеке ТУСУР – 15 экз.)
2. Липаев В.В. Тестирование компонентов и комплексов программ. – М.: Синтег, 2010. – 399 с. (наличие в библиотеке ТУСУР – 9 экз.)

Дополнительная литература

1. Майерс Гленфорд Дж. Искусство тестирования программ. – М.: Финансы и статистика, 1982. – 176 с. (наличие в библиотеке ТУСУР – 3 экз.)
2. Бек К. Экстремальное программирование: разработка через тестирование. – СПб. : Питер, 2003. – 224 с. (наличие в библиотеке ТУСУР – 1 экз.)
3. Амблер С. Гибкие технологии: экстремальное программирование и унифицированный процесс разработки. – СПб.: Питер, 2005. – 411 с. (наличие в библиотеке ТУСУР – 5 экз.)
4. Басовский Л.Е., Протасьев В.Б. Управление качеством: Учебник для вузов. – М.: Инфра-М, 2008. – 211 с. Гриф МО РФ. (наличие в библиотеке ТУСУР – 10 экз.)

Базы данных, информационно справочные и поисковые системы

Научно-образовательный портал университета (http://edu.tusur.ru); электронные информационно-справочные ресурсы вычислительных залов кафедры АОИ.

Материально-техническое обеспечение дисциплин кафедры АОИ.

Для проведения **занятий лекционного типа, групповых и индивидуальных консультаций, текущего контроля и промежуточной аттестации** используются аудитории, расположенные по адресу 634034, Томская область, г. Томск, ул. Вершинина, д. 74, 4 этаж:

– **ауд. 412**. Состав оборудования:

Компьютер для преподавателя на базе Intel Celeron 2.53 Ггц, ОЗУ – 1 Гб, жесткий диск – 80 Гб. Видеопроектор BENQ, экран, магнитно-маркерная доска, стандартная учебная мебель. Количество посадочных мест -99.

Используется лицензионное программное обеспечение: Windows XP Professional SP 3, MS Office 2003 SP3, Антивирус Касперского 6.0.

Свободно распространяемое программное обеспечение: Developer C++, Adobe Reader X.

Компьютер подключен к сети ИНТЕРНЕТ и обеспечивает доступ в электронную информационно-образовательную среду университета.

– **ауд. 421.** Состав оборудования:

Компьютер для преподавателя на базе Intel Celeron 2.93 Ггц, ОЗУ – 512 Мб, жесткий диск – 30 Гб. Видеопроектор BENQ MX 501, экран, магнитно-маркерная доска, стандартная учебная мебель. Количество посадочных мест - 99.

Используется лицензионное программное обеспечение: Windows XP Professional SP 3, MS Office 2003 SP3, Антивирус Касперского 6.0.

Свободно распространяемое программное обеспечение: Developer C++, Adobe Reader X.

Компьютер подключен к сети ИНТЕРНЕТ и обеспечивает доступ в электронную информационно-образовательную среду университета.

– **ауд. 418.** Состав оборудования:

Компьютер для преподавателя на базе Intel Celeron 2.53 Ггц, ОЗУ – 1.25 Гб, жесткий диск – 80 Гб. Широкоформатный телевизор для презентаций , экран, магнитно-маркерная доска, стандартная учебная мебель. Количество посадочных мест - 50.

Используется лицензионное программное обеспечение: Windows XP Professional SP 3, MS Office 2003 SP3, Антивирус Касперского 6.0.

Свободно распространяемое программное обеспечение: Developer C++, Adobe Reader X.

Компьютер подключен к сети ИНТЕРНЕТ и обеспечивает доступ в электронную информационно-образовательную среду университета.

Для проведения **практических и лабораторных занятий** используются вычислительные классы, расположенные по адресу 634034, Томская область, г. Томск, ул. Вершинина, д. 74, 4 этаж:

– **ауд. 407.** Состав оборудования:

Видеопроектор Optoma Ex632.DLP, экран Lumian Mas+Er, магнитно-маркерная доска, стандартная учебная мебель.

Компьютеры – 12 шт. Дополнительные посадочные места – 13 шт.

Компьютеры Intel Core i5-2320 3.0 Ггц, ОЗУ – 4 Гб, жесткий диск – 500 Гб.

Используется лицензионное программное обеспечение: Windows 7 Enterprise N (Windows 7 Professional), 1С:Предприятие 8.3, Mathcad 13, MS Office 2003, Пакет совместимости для выпуска 2007 MS Office, MS Project профессиональный 2010, MS Visual Studio Professional, Антивирус Касперского 6.0.

Свободно распространяемое программное обеспечение:Far file manager, GIMP 2.8.8, Google Earth, Java 8, QGIS Wien 2.8.1, Adobe Reader X, Mozilla Firefox, Google Chrome, Eclipse IDE for Java Developers 4.2.1, Dev-C++, FreePascal, IntelliJ IDEA 15.0.3, ARIS Express, Open Office, MS Silverlight, Pyton 2.5, MS SQL Server 2008 Express.

Компьютеры подключен к сети ИНТЕРНЕТ и обеспечивает доступ в электронную информационно-образовательную среду университета.

– **ауд. 409.** Состав оборудования:

Видеопроектор Optoma Ex632.DLP, экран Lumian Mas+Er, магнитно-маркерная доска, стандартная учебная мебель.

Компьютеры – 9 шт. Дополнительные посадочные места – 16 шт.

Компьютеры Intel Core 2 6300 1.86 Ггц, ОЗУ – 2 Гб, жесткий диск – 150 Гб.

Используется лицензионное программное обеспечение: Windows XP Professional SP 3, 1С:Предприятие 8.3, Mathcad 13, MS Office 2003, Пакет совместимости для выпуска 2007 MS Office, MS Project профессиональный 2010, MS Visual Studio Professional, Антивирус Касперского 6.0

Свободно распространяемое программное обеспечение:Far file manager, GIMP 2.8.8, Google Earth, Java 8, QGIS Wien 2.8.1, Adobe Reader X, Mozilla Firefox, Google Chrome, Eclipse IDE for Java Developers 4.2.1, Dev-C++, FreePascal, IntelliJ IDEA 15.0.3., ARIS Express, Open Office, MS Silverlight, Pyton 2.5, MS SQL Server 2008 Express.

Компьютеры подключены к сети ИНТЕРНЕТ и обеспечивает доступ в электронную информационно-образовательную среду университета.

– **ауд. 428.** Состав оборудования:

Доска меловая, стандартная учебная мебель.

Компьютеры – 14 шт. Дополнительные посадочные места – 11 шт.

Компьютеры Intel Core 2 Duo E6550 2.33 Ггц, ОЗУ – 2 Гб, жесткий диск – 250 Гб.

Используется лицензионное программное обеспечение: Windows XP Professional SP 3, 1С:Предприятие 8.3, Mathcad 13, MS Office 2003, Пакет совместимости для выпуска 2007 MS Office, MS Project профессиональный 2010, MS Visual Studio Professional, Антивирус Касперского 6.0

Свободно распространяемое программное обеспечение:Far file manager, GIMP 2.8.8, Google Earth, Java 8, QGIS Wien 2.8.1, Adobe Reader X, Mozilla Firefox, Google Chrome, Eclipse IDE for Java Developers 4.2.1, Dev-C++, FreePascal, IntelliJ IDEA 15.0.3, ARIS Express, Open Office, MS Silverlight, Pyton 2.5, MS SQL Server 2008 Express.

Компьютеры подключены к сети ИНТЕРНЕТ и обеспечивает доступ в электронную информационно-образовательную среду университета.

– **ауд. 430.** Состав оборудования:

Магнитно-маркерная доска, стандартная учебная мебель.

Компьютеры – 12 шт. Дополнительные посадочные места – 13 шт.

Компьютеры Intel Core 2 Duo E6550 2.33 Ггц, ОЗУ – 2 Гб, жесткий диск – 250 Гб.

Используется лицензионное программное обеспечение: Windows XP Professional SP 3, 1С:Предприятие 8.3, Mathcad 13, MS Office 2003, Пакет совместимости для выпуска 2007 MS Office, MS Project профессиональный 2010, MS Visual Studio Professional, Антивирус Касперского 6.0

Свободно распространяемое программное обеспечение:Far file manager, GIMP 2.8.8, Google Earth, Java 8, QGIS Wien 2.8.1, Adobe Reader X, Mozilla Firefox, Google Chrome, Eclipse IDE for Java Developers 4.2.1, Dev-C++, FreePascal, IntelliJ IDEA 15.0.3, ARIS Express, Open Office, MS Silverlight, Pyton 2.5, MS SQL Server 2008 Express.

Компьютеры подключены к сети ИНТЕРНЕТ и обеспечивает доступ в электронную информационно-образовательную среду университета.

– **ауд. 432а.** Состав оборудования:

Доска меловая, стандартная учебная мебель.

Компьютеры – 12 шт. Дополнительные посадочные места – 13 шт.

Компьютеры Intel Core i5-3330 3.0 Ггц, ОЗУ – 4 Гб, жесткий диск – 500 Гб.

Используется лицензионное программное обеспечение: Windows 7 Enterprise N (Windows 7 Professional), 1С:Предприятие 8.3, Mathcad 13, MS Office 2003, Пакет совместимости для выпуска 2007 MS Office, MS Project профессиональный 2010, MS Visual Studio Professional, Антивирус Касперского 6.0

Свободно распространяемое программное обеспечение:Far file manager, GIMP 2.8.8, Google Earth, Java 8, QGIS Wien 2.8.1, Adobe Reader X, Mozilla Firefox, Google Chrome, Eclipse IDE for Java Developers 4.2.1, Dev-C++, FreePascal, IntelliJ IDEA 15.0.3, ARIS Express, Open Office, MS Silverlight, Pyton 2.5, MS SQL Server 2008 Express.

Компьютеры подключены к сети ИНТЕРНЕТ и обеспечивает доступ в электронную информационно-образовательную среду университета.

– **ауд. 432б.** Состав оборудования:

Магнитно-маркерная доска, стандартная учебная мебель.

Компьютеры – 12 шт. Дополнительные посадочные места – 13 шт.

Компьютеры Intel Core i5-2320 3.0 Ггц, ОЗУ – 4 Гб, жесткий диск – 500 Гб.

Используется лицензионное программное обеспечение: Windows 7 Enterprise N (Windows 7 Professional), 1С:Предприятие 8.3, Mathcad 13, MS Office 2003, Пакет совместимости для выпуска 2007 MS Office, MS Project профессиональный 2010, MS Visual Studio Professional, Антивирус Касперского 6.0

Свободно распространяемое программное обеспечение:Far file manager, GIMP 2.8.8, Google Earth, Java 8, QGIS Wien 2.8.1, Adobe Reader X, Mozilla Firefox, Google Chrome, Eclipse IDE for Java Developers 4.2.1, Dev-C++, FreePascal, IntelliJ IDEA 15.0.3, ARIS Express, Open Office, MS Silverlight, Pyton 2.5, MS SQL Server 2008 Express.

Компьютеры подключены к сети ИНТЕРНЕТ и обеспечивает доступ в электронную информационно-образовательную среду университета.

Для **самостоятельной работы и занятий ГПО** используется аудитория, расположенная по адресу 634034, Томская область, г. Томск, ул. Вершинина, д. 74, 4 этаж, ауд 431.

Состав оборудования:

Видеопроектор Infocus LP540, магнитно-маркерная доска, стандартная учебная мебель. Компьютеры – 5 шт. Количество посадочных мест -10.

Компьютеры Intel Core 2 Duo E6550 2.33 Ггц, ОЗУ – 2 Гб, жесткий диск – 250 Гб.

Используется лицензионное программное обеспечение: Windows XP Professional SP 3, 1С:Предприятие 8.3, Mathcad 13, MS Office 2003, Пакет совместимости для выпуска 2007 MS Office, MS Project профессиональный 2010, MS Visual Studio Professional, Антивирус Касперского 6.0

Свободно распространяемое программное обеспечение:Far file manager, GIMP 2.8.8, Google Earth, Java 8, QGIS Wien 2.8.1, Adobe Reader X, Mozilla Firefox, Google Chrome, Eclipse IDE for Java Developers 4.2.1, Dev-C++, FreePascal, IntelliJ IDEA 15.0.3, ARIS Express, Open Office, MS Silverlight, Pyton 2.5, MS SQL Server 2008 Express.

Компьютеры подключены к сети ИНТЕРНЕТ и обеспечивает доступ в электронную информационно-образовательную среду университета.

Приложение А

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)
Кафедра автоматизации обработки информации (АОИ)

Отчет
по лабораторной работе № __
по дисциплине «Тестирование программного обеспечения»

Студент гр. ___
_____ И. О. Фамилия
«___»_____201_ г.

Руководитель
Ст. преподаватель. каф. АОИ,
канд. техн. наук
_____ Ю. В. Морозова
«___»_____201_ г.

Томск 201_