МИНОБРНАУКИ РОССИИ

Орский гуманитарно-технологический институт (филиал)
федерального государственного бюджетного образовательного учреждения
высшего образования «Оренбургский государственный университет»
(Орский гуманитарно-технологический институт (филиал) ОГУ)

Кафедра программного обеспечения

Методические указания по выполнению и защите лабораторных работ по дисциплине «Б1.Д.В.18 Тестирование программного обеспечения»

Уровень высшего образования

БАКАЛАВРИАТ

Направление подготовки

09.03.01 Информатика и вычислительная техника

(код и наименование направления подготовки)

Программное обеспечение средств вычислительной техники и автоматизированных систем

(наименование направленности (профиля) образовательной программы)

Тип образовательной программы

Программа бакалавриата

Квалификация

Бакалавр

Форма обучения

Очная

Год начала реализации программы (набора)

2019

г. Орск 2018

Методические указания предназначены для обучающихся очной формы обучения направления подготовки 09.03.01 Информатика и вычислительная техника профилю Программное обеспечение средств вычислительной техники и автоматизированных систем по дисциплине «Б1.Д.В.18 Тестирование программного обеспечения»

Составитель _____ О.В. Подсобляева

Методические указания рассмотрены и одобрены на заседании кафедры программного обеспечения, протокол № 1 от «01» сентября 2018 г.

Заведующий кафедрой _____ Е.Е. Сурина

Согласовано:
Председатель методической комиссии по направлению подготовки 09.03.01 Информатика и вычислительная техника

_____ Е.Е.Сурина
«12» сентября 2018 г.

Пояснительная записка

В результате изучения дисциплины «Б1.Д.В.18 Тестирование программного обеспечения» у обучающихся должны быть сформированы знания, умения и навыки:

1. Дать представление о теоретических основах тестирования: фазы и технологии тестирования, критерии и метрики тестов, особенности процесса;
2. Научиться создавать собственные тест-кейсы;
3. Освоить современные системы отслеживания ошибок (issue tracker, bugtracker), познакомиться со стандартами использования таких трекеров;
4. Получить опыт тестирования задач из условно-реального проекта по разработке программного обеспечения;
5. Ознакомиться с внутренней организацией процесса тестирования и его включения в общие бизнес-процессы компании-разработчика ПО.

*о*дной из наиболее эффективных форм закрепления теоретических знаний и выработки навыков самостоятельной работы являются лабораторные занятия.

Целью проведения лабораторных занятий является:

- закрепление знаний студентов по основам проектной деятельности,
- формирование у студентов навыков использования современных технических средств и технологий для решения проектных и исследовательских задач.

Тематический план

Таблица 1 – Тематический план выполнения лабораторных работ по дисциплине «Б1.Д.В.18 Тестирование программного обеспечения» для обучающихся направления подготовки 09.03.01 Информатика и вычислительная техника профиль подготовки Программное обеспечение средств вычислительной техники и автоматизированных систем

Лабораторные работы

№ раздела	Наименование лабораторных работ	Кол-во часов
2	Методы тестирования. Подходы к тестированию. Организация процесса тестирования ПО.	4
3	Тестирование ПО по методу «белого ящика»	2
4	Тестирование ПО по методу «черного ящика»	2
5	Способы тестирования по методу «белого ящика»	4
5	Способы тестирования по методу «черного ящика»	4
	Итого:	16

Методические указания по выполнению и оформлению лабораторных работ

Лабораторные работы по дисциплине «Тестирование программного обеспечения» предполагают решение задач по темам, представленным в тематическом плане.

В практической работе должны быть выполнены все предусмотренные задания. В работе должна просматриваться логическая последовательность и взаимная увязка основных частей работы.

Рекомендуемая структура лабораторных работ:

1) цель практической работы;
2) задание в соответствии с выбранным вариантом;
3) теоретическая часть, включающая краткое изложение теоретических положений по

теме практической работы, формулы для решения задания;

4) практическая часть, включающая решение задания по теме практической работы. Дополнительно для наглядности расчетный материал может быть представлен в виде таблиц, графиков;

5) выводы по практической работе;

6) список использованной литературы.

Лабораторные работы могут быть оформлены:

- машинописным текстом на листах формата А4.

Титульный лист оформляется на основе СТО 02069024. 101 – 2014 «РАБОТЫ СТУДЕНЧЕСКИЕ. Общие требования и правила оформления».

Работа защищается устно и принимается к зачету, если нет замечаний по ее выполнению и оформлению. При отсутствии зачтенных лабораторных работ студент не допускается к зачету по дисциплине «Б1.Д.В.18 Тестирование программного обеспечения».

Лабораторная работа №1,2,3
Методы тестирования. Подходы к тестированию. Организация процесса тестирования ПО. Методы тестирования. Подходы к тестированию. Организация процесса тестирования ПО. Тестирование ПО по методу «белого ящика» Тестирование ПО по методу «черного ящика»

Цель работы: изучить классификацию видов тестирования, практически закрепить эти знания путем генерации тестов различных видов, научиться планировать тестовые активности в зависимости от специфики поставляемой на тестирование функциональности.

Теоретические сведения

Тестирование – процесс, направленный на оценку корректности, полноты и качества разработанного программного обеспечения.

Тестирование можно классифицировать по очень большому количеству признаков. Далее приведен обобщенный список видов тестирования по различным основаниям.

Типы тестов по покрытию (по глубине)

Smoke test – тестирование системы для определения корректной работы базовых функций программы в целом, без углубления в детали. При проведении теста определяется пригодность сборки для дальнейшего тестирования.

Minimal Acceptance Test (MAT, Positive test): тестирование системы или ее части только на валидных данных (валидные данные – это данные, которые необходимо использовать для корректной работы модуля/функции). При тестировании проверяется правильной работы всех функций и модулей с валидными данными.

Для крупных и сложных приложений используется ограниченный набор сценариев и функций.

Acceptance Test (AT): полное тестирование системы или ее части как на корректных, так и на некорректных данных/сценариях. Вид теста, направленный на подтверждение того, что приложение может использоваться по назначению при любых условиях.

Тест на этом уровне покрывает все возможные сценарии тестирования: проверку работоспособности модулей при вводе корректных значений; проверку при вводе некорректных значений; использование форматов данных отличных от тех, которые указаны в требованиях; проверку исключительных ситуаций, сообщений об ошибках; тестирование на различных комбинациях входных параметров; проверку всех классов эквивалентности; тестирование граничных значений интервалов; сценарии не предусмотренные спецификацией и т.д.

Тестовые активности (типы тестов по покрытию (по ширине)):

Defect Validation – проверка результата исправления дефектов. Включает в себя проверку на воспроизводимость дефектов, которые были бисправлены в новой сборке продукта, а также проверку того, что исправление не повлияло на ранее работавшую

функциональность

New Feature Test (NFT, AT of NF) – определение качества поставленной на тестирование новой функциональности, которая ранее не тестировалась. Данный тип тестирования включает в себя: проведение полного теста (АТ) непосредственно новой функциональности; тестирование новой функциональности на соответствие документации; проверку всевозможных взаимодействий ранее реализованной функциональности с новыми модулями и функциями.

Regression testing (регрессионное тестирование) **–** проводится с целью оценки качества ранее реализованной функциональности. Включает в себя проверку стабильности ранее реализованной функциональности после внесения изменений, например добавления новой функциональности, исправление дефектов, оптимизация кода, разворачивание приложения на новом окружении. Регрессионное тестирование может быть проведено на уровне Smoke, MAT или AT.

Типы тестов по знанию коду

Черный ящик – тестирование системы, функциональное или нефункциональное, без знания внутренней структуры и компонентов системы. У тестировщика нет доступа к внутренней структуре и коду приложения либо в процессе тестирования он не обращается к ним.

Белый ящик – тестирование основанное на анализе внутренней структуры компонентов или системы. У тестировщика есть доступ к внутренней структуре и коду приложения.

Серый ящик – комбинация методов белого и черного ящика, состоящая в том, что к части кода архитектуры у тестировщика есть, а к части кода – нет.

Типы тестов по степени автоматизации

Ручное – тестирование, в котором тест-кейсы выполняются тестировщиком вручную без использования средств автоматизации.

Автоматизированное – набор техник, подходов и инструментальных средств, позволяющий исключить человека из выполнения некоторых задач в процессе тестирования. Тест-кейсы частично или полностью выполняет специальное инструментальное средство.

Типы тестов по изолированности компонентов

Unit/component (модульное) – тестирование отдельных компонентов (модулей) программного обеспечения.

Integration (интеграционное) – тестируется взаимодействие между интегрированными компонентами или системами.

System (системное) – тестируется работоспособность системы в целом с целью проверки того, что она соответствуе установленным требованиям.

Типы тестов по подготовленности.

Интуитивное тестирование выполняется без подготовки к тестам, без определения ожидаемых результатов, проектирования тестовых сценариев.

Исследовательское тестирование – метод проектирования тестовых сценариев во время выполнения этих сценариев. Тестировщик совершает проверки, продумывает их, придумывает новые проверки, часто использует для этого полученную информацию.

Тестирование по документации – тестирование по подготовленным тестовым сценариям, руководству по осуществлению тестов.

Титы тестов по месту и времени проведения

User Acceptance Testing (UAT) (приемочное тестирование) – 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 авторизованным лицам определить, принимать систему.

Alpha Testing (альфа-тестирование) **–** моделируемое или действительное

функциональное тестирование, выполняется в организации, разрабатывающей продукт, но не проектной командой (это может быть независимая команда тестировщиков, потенциальные пользователи, заказчики). Альфа тестирование часто применяется к коробочному программному обеспечению в качестве внутреннего приемочного тестирования.

Beta Testing (бета-тестирование) **–** эксплуатационное тестирование потенциальными или существующими клиентами/заказчиками на внешней стороне (в среде, где продукт будет использоваться) никак связанными с разработчиками, с целью определения действительно ли компонент или система удовлетворяет требованиям клиента/заказчика и вписывается в бизнес-процессы. Бета-тестирование часто проводится как форма внешнего приемочного тестирования готового программного обеспечения для того, чтобы получить отзывы рынка.

Типы тестов по объекту тестирования

Functional testing (функциональное тестирование) – это тестирование, основанное на анализе спецификации, функциональности компонента или системы. Функциональным можно назвать любой вид тестирования, который согласно требованиям проверяет правильную работу.

Safety testing (тестирование безопасности) – тестирование программного продукта с целью определить его безопасность (безопасность

– способность программного продукта при использовании оговоренным образом оставаться в рамках приемлемого риска причинения вреда здоровью, бизнесу, программам, собственности или окружающей среде.

Security testing (тестирование защищенности) – это тестирование с целью оценить защищенность программного продукта. Тестирование защищенности проверяет фактическую реакцию защитных механизмов, встроенных в систему, на проникновение.

Compatibility testing (тестирование совместимости) **–** процесс тестирования для определения возможности взаимодействия программного продукта, проверка работоспособности приложения в различных средах (браузеры и их версии, операционные системы, их типа, версии и разрядность)

Виды тестов:

- ✓ кросс-браузерное тестирование (различные браузеры или версии браузеров)
- ✓ кросс-платформенное тестирование (различные операционные системы или версии операционных систем)

Нефункциональное тестирование – это проверка характеристик программы. Иначе говоря, когда проверяется не именно правильность работы, а какие-либо свойства (внешний вид и удобство пользования, скорость работы и т.п.).

1. Тестирование пользовательского интерфейса (GUI) – тестирование, выполняемое путем взаимодействия с системой через графический интерфейс пользователя.

- ✓ навигация
- ✓ цвета, графика, оформление
- ✓ содержание выводимой информации
- ✓ поведение курсора и горячие клавиши
- ✓ отображение различного количества данных (нет данных, минимальное и максимальное количество)
- ✓ изменение размеров окна или разрешения экрана

2. Тестирование удобства использования (Usability Testing) – тестирование с целью определения степени понятности, легкости в изучении и использовании, привлекательности программного продукта для пользователя при условии использования в заданных условиях эксплуатации.

- ✓ визуальное оформление
- ✓ навигация

- ✓ логичность

3. Тестирование доступности (Accessibility testing) – тестирование, которое определяет степень легкости, с которой пользователи с ограниченными способностями могут использовать систему или ее компоненты.

4. Тестирование интернационализации – тестирование способности продукта работать в локализованных средах (способность изменять элементы интерфейса в зависимости от длины и направления текста, менять сортировки/форматы под различные локали и т.д.). (Максим Черняк).

Интернационализация – это процесс, упрощающий дальнейшую ьадаптацию продукта к языковым и культурным особенностям региона, отличного от того, в котором разрабатывался продукт. Это адаптация продукта для потенциального использования практически в любом месте, Интернационализация производится на начальных этапах разработки, в то время как локализация — для каждого целевого языка.

5. Тестирование локализации (Localization testing) – тестирование, проводимое с целью проверить качество перевода продукта с одного языка на другой.

6. Тестирование производительности или нагрузочное тестирование – процесс тестирования с целью определения производительности программного продукта.

Виды тестов:

- ✓ нагрузочное тестирование (Performance and Load testing) – вид тестирования производительности, проводимый с целью оценки поведения компонента или системы при возрастающей нагрузке, например количестве параллельных пользователей и/или операций, а также определения какую нагрузку может выдержать компонент или система;
- ✓ объемное тестирование (Volume testing) – позволяет получить оценку производительности при увеличении объемов данных в базе данных приложения;
- ✓ тестирование стабильности и надежности (Stability / Reliability testing) – позволяет проверять работоспособность приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки.
- ✓ стрессовое тестирование (Stress testing) – вид тестирования производительности, оценивающий систему или компонент на граничных значениях рабочих нагрузок или за их пределами, или же в состоянии ограниченных ресурсов, таких как память или доступ к серверу.

7. Тестирование требований (Requirements testing) – проверка требований на соответствие основным характеристикам качества.

8. Тестирование прототипа (Prototyte testing) – метод выявления структурных, логических ошибок и ошибок проектирования на ранней стадии развития продукта до начала фактической разработки.

9. Тестирование установки (Installability testing) и лицензирования – процесс тестирования устанавливаемости программного продукта.

Виды тестов:

- ✓ формальный тест программы установки приложения (проверка пользовательского интерфейса, навигации, удобства пользования, соответствия общепринятым стандартам оформления);
- ✓ функциональный тест программы установки;
- ✓ тестирование механизма лицензирования и функций защиты от пиратства;
- ✓ проверка стабильности приложения после установки.

10. Тестирование на отказ и восстановление (Failover and Recovery Testing) – тестирование при помощи эмуляции отказов системы или реально вызываемых отказов в управляемом окружении.

Тестирование программного продукта включает следующие этапы:

1. Изучение и анализ предмета тестирования.

2. Планирование тестирования.
3. Исполнение тестирования.

Изучение и анализ предмета тестирования начинается еще до утверждения спецификации и продолжается на стадии разработки (кодирования) программного обеспечения. Конечной целью этапа изучение и анализ предмета тестирования является получение ответов на два вопроса:

- какие функциональности предстоит протестировать,
- как эти функциональности работают.

Планирование тестирования происходит на стадии разработки (кодирования) программного обеспечения. На стадии планирования тестирования перед тестировщиком стоит задача поиска компромисса между объемом тестирования, который возможен в теории, и объемом тестирования, который возможен на практике. На данной стадии необходимо ответить на вопрос: как будем тестировать? Результатом планирования тестирования является тестовая документация.

Выполнение тестирования происходит на стадии тестирования и представляет собой практический поиск дефектов с использованием тестовой документации, составленной ранее.

Для всех программных продуктов выполняют следующие типы тестов и их композиции.

Для первого билда рекомендуется проводить Smoke+AT готовой функциональности: поверхностное тестирование (Smoke Test) выполняется для определения пригодности сборки для дальнейшего тестирования; полное тестирование системы или ее части как на корректных, так и на некорректных данных/сценариях (Acceptance Test, AT) позволяет обнаружить дефекты и внести запись о них в багтрэкинговую систему.

Для последующих билдов композиции тестов могут быть следующими:

- Если не была добавлена новая функциональность, то: DV+MAT. Т.е., выполняется проверка исправления дефектов программистом (Defect Validation, DV), а также проверка работоспособности остальной функциональности после исправления дефектов на позитивных сценариях (Minimal Acceptance Test, MAT).
- Если была добавлена новая функциональность, то: Smoke+DV+NFT+Regression Test. В частности, выполняется поверхностное тестирование (Smoke Test), проверка исправления дефектов программистом (Defect Validation, DV), тестирование новых функциональностей (New Feature Testing, NFT), проверка старых функциональностей, т.е. регрессионное тестирование (Regression Test).
- Если была добавлена новая функциональность, то возможен также вариант: DV+NFT+Resression test, т.е. без выполнения Smoke Test.

В зависимости от типа и специфики приложения (web, desktop, mobile) выполняют специализированные тесты (например, кроссбраузерное илиь кроссплатформенное тестирование, тестирование локализации и интернационализации и др.).

Пример выполнения лабораторной работы

Необходимо составить тестовый план для объекта «Карандаш».
Пример тестового плана для объекта карандаш представлен на рисунке 1.1.

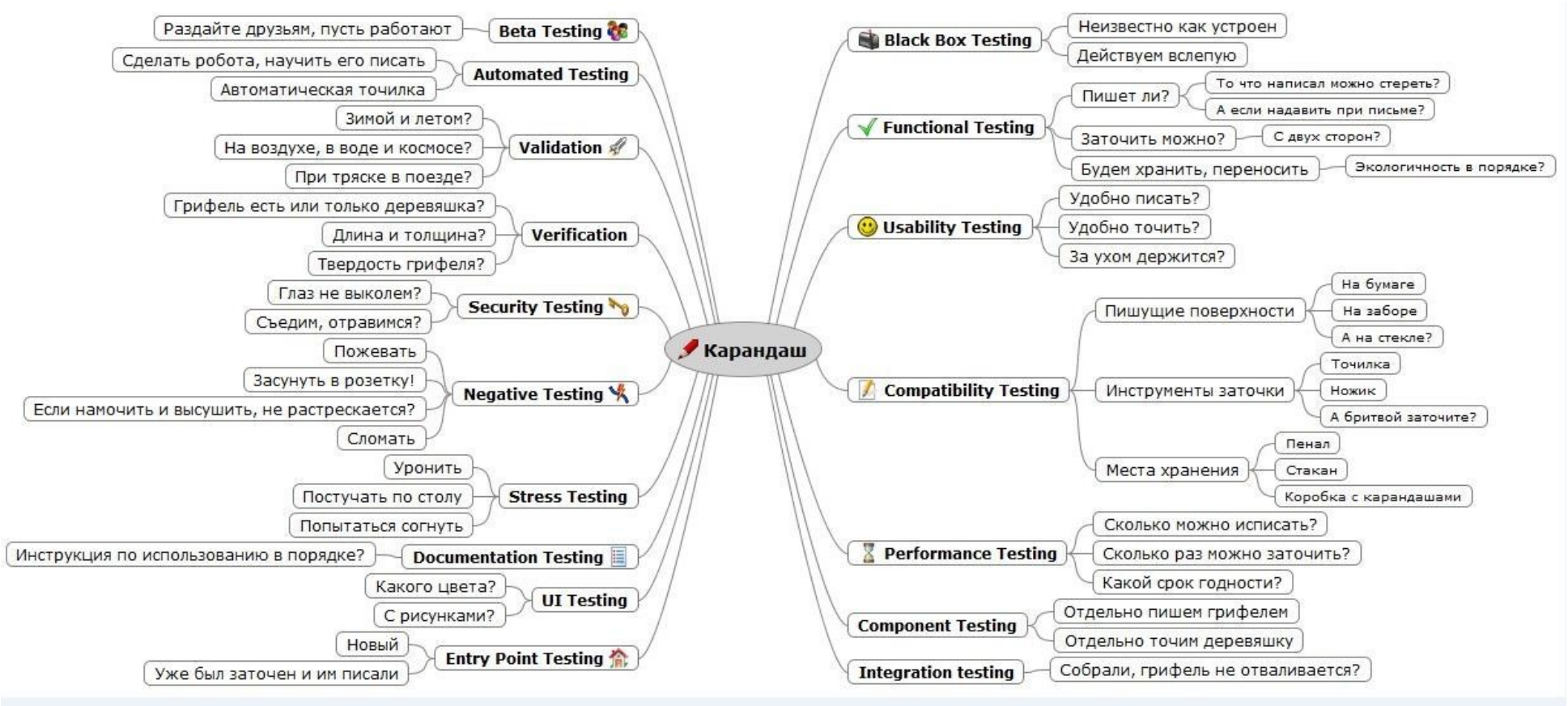


Рисунок 1.1 – Пример генерации тестов различных видов для объекта «Карандаш»

Теоретическая часть. Виды тестирования

Тестирование программного обеспечения включает в себя целый комплекс действий, аналогичных последовательности процессов разработки программного обеспечения. В него входят [7]:

- • постановка задачи для теста;
- • проектирование теста;
- • написание тестов;
- • тестирование тестов;
- • выполнение тестов;
- • изучение результатов тестирования.

Наиболее важным является проектирование тестов. Существуют разные подходы к проектированию тестов.

Первый состоит в том, что тесты проектируются на основе внешних спецификаций программ и модулей либо спецификаций сопряжения модуля с другими модулями, программа при

этом рассматривается как «черный ящик». Смысл теста заключается в том, чтобы проверить, соответствует ли программа внешним спецификациям. При этом содержание модуля не имеет значения. Такой подход получил название — стратегия «черного ящика».

Второй подход — стратегия «белого ящика», основан на анализе логики программы. При таком подходе тестирование заключается в проверке каждого пути, каждой ветви алгоритма. При этом внешняя спецификация во внимание не принимается.

Ни один из этих подходов не является оптимальным. Реализация тестирования методом «черного ящика» сводится к проверке всех возможных комбинаций входных данных. Невозможно протестировать программу, подавая на вход бесконечное множество значений, поэтому ограничиваются определенным набором данных. При этом исходят из максимальной отдачи теста по сравнению с затратами на его создание. Она измеряется вероятностью того, что тест выявит ошибки, если они имеются в программе. Затраты измеряются временем и стоимостью подготовки, выполнения и проверки результатов теста.

Тестирование методом «белого ящика» также не дает 100%-ной гарантии того, что модуль не содержит ошибок. Даже если предположить, что выполнены тесты для всех ветвей алгоритма, нельзя с полной уверенностью утверждать, что программа соответствует ее спецификациям. Например, если требовалось написать программу для вычисления кубического корня, а программа фактически вычисляет корень квадратный, то реализация будет совершенно неправильной, даже если проверить все пути. Вторая проблема — отсутствующие пути. Если программа реализует спецификации не полностью (например, отсутствует такая специализированная функция, как проверка на отрицательное значение входных данных программы вычисления квадратного корня), никакое тестирование существующих путей не выявит такой ошибки. И наконец, проблема зависимости результатов тестирования от входных данных. Одни данные будут давать правильные результаты, а другие нет. Например, если для определения равенства трех чисел программируется выражение вида:

? (A + B+ C)/Ъ=A,

то оно будет верным не для всех значений *A, B* и С (ошибка возникает в том случае, когда из двух значений *B* или *C* одно больше, а другое на столько же меньше *A).* Если концентрировать внимание только на тестировании путей, нет гарантии, что эта ошибка будет выявлена.

Таким образом, полное тестирование программы невозможно, т. е. никакое тестирование не гарантирует полное отсутствие ошибок в программе. Поэтому необходимо проектировать тесты таким образом, чтобы увеличить вероятность обнаружения ошибки в программе.

Стратегия «белого ящика»

Существуют следующие методы тестирования по принципу «белого ящика»:

- • покрытие операторов;
- • покрытие решений;
- • покрытие условий;

- • покрытие решений/условий;
- • комбинаторное покрытие условий.

Метод покрытия операторов

Целью этого метода тестирования является выполнение каждого оператора программы хотя бы один раз.

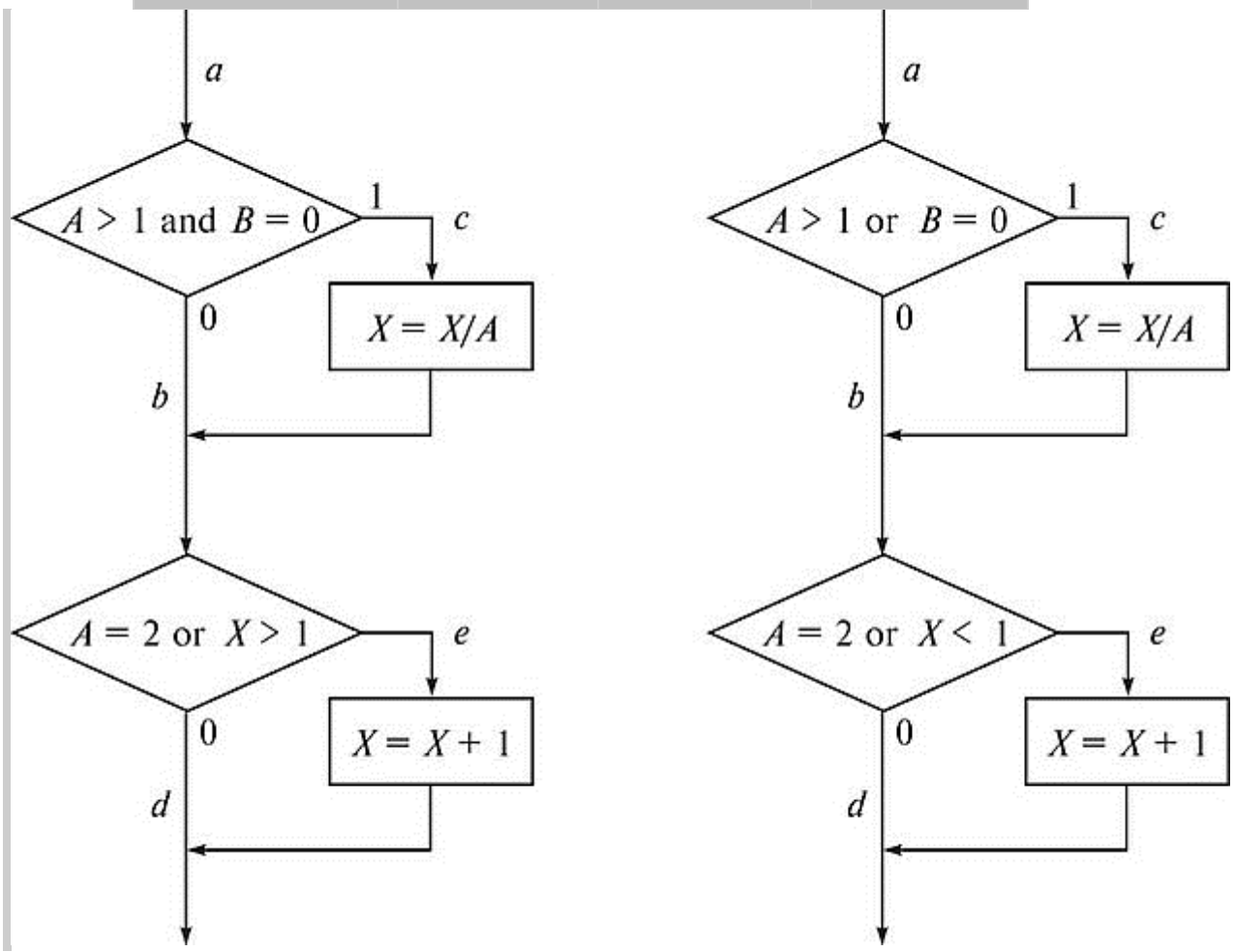
Пример.

Если для тестирования задать значения переменных $A = 2$, $B = 0$, $X = 3$, будет реализован путь *ace,* т. е. каждый оператор программы выполнится один раз (рис. Л5.1, *а).* Но если внести в алгоритм ошибки — заменить в первом условии *and* на *or,* а во втором $X > 1$ на $X < 1$ (рис. Л5.1, *б),* ни одна ошибка не будет обнаружена (табл. Л5.1). Кроме того, путь *abd* вообще не будет охвачен тестом, и если в нем есть ошибка, она также не будет обнаружена. В табл. Л5.1 ожидаемый результат определяется по блок-схеме на рис. Л5.1, *а,* а фактический — по рис. Л5.1, *б.*

Как видно из этой таблицы, ни одна из внесенных в алгоритм ошибок не будет обнаружена.

Таблица 1. Результат тестирования методом покрытия операторов

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
$A = 2, B = 0, X = Ъ$	$X = 2{,}5$	$X = 2{,}5$	Неуспешно



а б

Рис. Л5.1. Пример алгоритма программы: *а* — правильный; *б* — с ошибкой

Метод покрытия решений (покрытия переходов)

Согласно методу покрытия решений каждое направление перехода должно быть реализовано, по крайней мере, один раз. Этот метод включает в себя критерий покрытия

операторов, так как при выполнении всех направлений переходов выполнятся все операторы, находящиеся на этих направлениях.

Для программы, приведенной на рис. Л5.1, покрытие решений может быть выполнено двумя тестами, покрывающими пути *{ace, aЬc*!}, либо *{acc1, aЬe*}. Для этого выберем следующие исходные данные; *{A* = 3, *B*= О, *X=3}* — в первом случае и *{A = 2, B*= 1, *X*= 1} — во втором. Однако путь, где *X* не меняется, будет проверен с вероятностью 50 %: если во втором условии вместо условия *X*> 1 записано *X*<, то ошибка не будет обнаружена двумя тестами.

Результаты тестирования приведены в табл. Л5.2.

Таблица 2. Результат тестирования методом покрытия решений

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
/1 = 3, *B*= 0, *X=2*	*X* = 1	*X* = 1	Неуспешно
A = 2, B = 1, *X*= 1	*X=2*	* II	Успешно

Метод покрытия условий

Этот метод может дать лучшие результаты по сравнению с предыдущими. В соответствии с методом покрытия условий записывается число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении выполнялись, по крайней мере, один раз.

В рассматриваемом примере имеем четыре условия: *{A*> 1, 5=0}, *{A* = 2, *X*> 1}. Следовательно, требуется достаточное число тестов, такое, чтобы реализовать ситуации, где *A* > 1, *A* < 1, 5 = 0 и й#0 в точке а и /1 = 2, *A *2, X*> и Т< 1 в точке *Ь.* Тесты, удовлетворяющие критерию покрытия условий (табл. Л5.3), и соответствующие им пути:

- а) *A = 2,* 5 = 0, *X=4 ace;*
- б) *A* = 1, 5= 1, *X=0 aЬc!.*

Таблица 3. Результаты тестирования методом покрытия условий

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
/1 = 2, *B*= 0, *X= 4*	*=з	*X= 3*	Неуспешно
A= 1, *B*= 1, *X=0*	*=0	*X*= 1	Успешно

Метод покрытия решений/условий

Критерий покрытия решений/условий требует такого достаточного набора тестов, чтобы все возможные результаты каждого условия выполнялись по крайней мере один раз, все результаты каждого решения выполнялись по крайней мере один раз и, кроме того, каждой точке входа передавалось управление по крайней мере один раз.

Недостатки метода:

- • не всегда можно проверить все условия;
- • невозможно проверить условия, которые скрыты другими условиями;
- • недостаточная чувствительность к ошибкам в логических выражениях.

Так, в рассматриваемом примере два теста метода покрытия условий

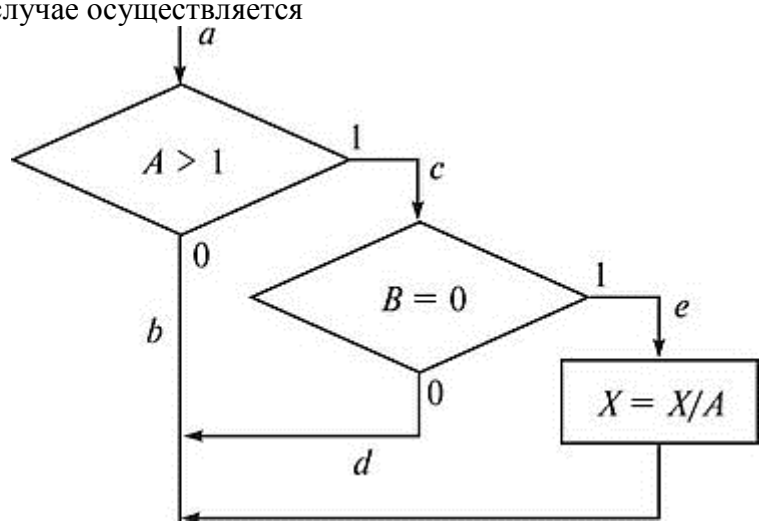
а) *A* = 2, *B* = О, *X= 4 ace;*

б*) A*= 1, *B*= 1, *X=0 aЬc!*

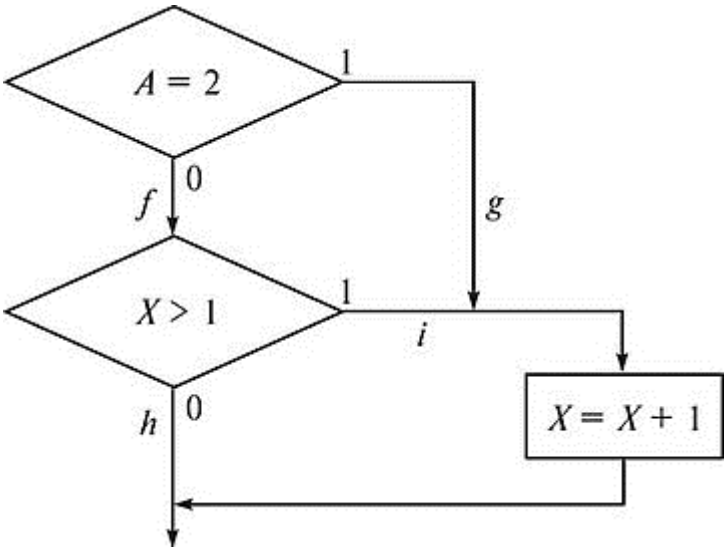
отвечают и критерию покрытия решений/условий. Это является следствием того, что одни условия приведенных решении скрывают другие условия в этих решениях. Так, если условие *A* > 1

будет ложным, транслятор может не проверять условия *B*= О, поскольку при любом результате условия *B*= 0 результат решения *((A >)&(B=0))* примет значение *ложь*. То есть в варианте на рис. Л5.1 не все результаты всех условий выполнятся в процессе тестирования.

Рассмотрим реализацию того же примера на рис. Л5.2. Наиболее полное покрытие тестами в этом случае осуществляется



' ,



так, чтобы выполнялись все возможные результаты каждого простого решения. Для этого нужно покрыть пути *aceg* (тест *A = 2, B*= О, *X*= 4), *acё/к* (тест *A* = 3, *B* = 1, *X*= 0), *aЬ/к* (тест *A* = 0, *B*= 0, *X*= 0), *aЬр* (тест Л = 0, *B* = 0, *X*= *2).*

Протестировав алгоритм на рис. Л5.2, нетрудно убедиться в том, что критерии покрытия условий и критерии покрытия решений/условий недостаточно чувствительны к ошибкам в логических выражениях.

Метод комбинаторного покрытия условий

Критерий комбинаторного покрытия условий удовлетворяет также и критериям покрытия решений, покрытия условий и покрытия решений/условий.

Этот метод требует создания такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении выполнялись по крайней мере один раз. По этому критерию в рассматриваемом примере должны быть покрыты тестами следующие восемь комбинаций:

- 1. *A*> 1, *B*= 0. *5. A = 2, X*> 1.
- 2. *A* > 1, *Bф* 0.
- 3. *A*< 1, *B*= 0.
- 4. *A*< 1, Вф 0.
- 6. *A* = 2, 1.

- 7. Аф2, *X*> 1.
- 8. Аф2, *X*< 1.

Для того чтобы протестировать эти комбинации, необязательно использовать все 8 тестов. Фактически они могут быть покрыты четырьмя тестами (табл. Л5.4):

- • *A* = 2, *B = 0, X=4* (покрывает 1,5};
- • *A = 2, B = 1, X= 1* (покрывает 2,6};
- • *A* = 0,5, *B*= 0, *X= 2* (покрывает 3, 7};
- • *A = I, B = 0, X*= (покрывает 4, 8}.

Таблица 4. Результаты тестирования методом комбинаторного покрытия условий

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A-2, B=0, X- 4	*=з	*=з	Неуспешно
A = 2, B= 1, *X*= 1	*X= 2*	1 *Г)* СЧ II	Успешно
A = 0,5, *B*= 0, *X=2*	*X=3*	*X= 4*	Успешно
A= 1, ?=0, *= 1	*X*= 1	*X*= 1	Неуспешно

Лабораторная работа №4
Способы тестирования по методу «белого ящика»

Цель: закрепить теоретические знания и получить лабораторные навыки в разработке программы тестирования методами по стратегии «белого ящика».

Используемые приемы и технологии: технология визуального проектирования и событийного программирования, среда программирования Visual Studio 2010 Professional, язык программирования Visual C++ 2010 Professional.

Ключевые термины: логика программы, критерии структурного тестирования, покрытие, методы структурного тестирования.

Варианты заданий

Вариант 1. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить сумму элементов массива, расположенных после минимального элемента.

Вариант 2. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить сумму элементов массива, расположенных между минимальным и максимальным элементами.

Вариант 3. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить среднеарифметическое значение элементов массива.

Вариант 4. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить среднеквадратическое значение элементов массива.

Вариант 5. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить среднегеометрическое значение ненулевых элементов массива.

Вариант 6. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить среднегармоническое значение положительных элементов массива.

Вариант 7. В одномерном динамическом массиве, состоящем из *n* элементов, поменять местами максимальный и минимальный элементы.

Вариант 8. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить сумму элементов массива, расположенных между первым и последним отрицательными элементами.

Вариант 9. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить сумму элементов массива, расположенных до последнего положительного элемента.

Вариант 10. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить сумму элементов массива, расположенных между первым и последним нулевыми элементами.

Вариант 11. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить сумму модулей элементов массива, расположенных после первого элемента, равного нулю.

Вариант 12. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить сумму положительных элементов массива, расположенных до максимального элемента.

Вариант 13. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить сумму элементов массива, расположенных до минимального элемента.

Вариант 14. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить сумму элементов массива, расположенных между первым и последним положительными элементами.

Вариант 15. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить среднее значение элементов, расположенных в массиве между первым последним нулевыми элементами.

Вариант 16. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить произведение элементов массива, расположенных между первым и вторым нулевыми элементами.

Вариант 17. В одномерном динамическом массиве, состоящем из *n* элементов, определить номер минимального и максимального элементов массива.

Вариант 18. В одномерном динамическом массиве, состоящем из *n* элементов, определить сумму модулей элементов массива, расположенных после первого отрицательного элемента.

Вариант 19. В одномерном динамическом массиве, состоящем из *n* элементов, определить количество элементов массива, больших С.

Вариант 20. В одномерном динамическом массиве, состоящем из *n* элементов, определить количество элементов массива, меньших С.

Вариант 21. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить сумму отрицательных элементов массива.

Вариант 22. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить сумму положительных элементов массива.

Вариант 23. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить произведение элементов массива с четными номерами.

Вариант 24. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить сумму элементов массива, расположенных после минимального элемента.

Вариант 25. В одномерном динамическом массиве, состоящем из *n* элементов, вычислить сумму элементов массива, расположенных до максимального элемента.

Лабораторная работа №5
Способы тестирования по методу «черного ящика»

Цель: закрепить теоретические знания и получить лабораторные навыки в разработке программы тестирования методом эквивалентного разбиения классов.

Используемые приемы и технологии: технология визуального проектирования и событийного программирования, среда программирования Visual Studio 2010 Professional, язык программирования Visual C++ 2010 Professional.

Ключевые термины: класс эквивалентности, правильный класс, неправильный класс, предусловия, постусловия, дерево разбиения области данных, анализ граничных значений.

Варианты заданий

Протестировать методом классов эквивалентности с построением дерева разбиения области данных программу, формализующую алгоритм варианта задачи.

Вариант 1. Решить алгебраическое уравнение 2-й степени (квадратное уравнение) $a * x^2 + b * x + c = 0$.

Вариант 2. Решить алгебраическое уравнение 3-й степени (кубическое уравнение) $a*x^3 + b*x^2 + c*x + d = 0.$ Корни приведенного уравнения рассчитать по формулам Кардано.

Вариант 3. Решить алгебраическое уравнение 3-й степени (кубическое уравнение) $a*x^3 + b*x^2 + c*x + d = 0.$ Корни рассчитать по тригонометрической формуле Виета.

$$a * x^4 + b * x^2 + c = 0.$$

Вариант 4. Решить биквадратное уравнение

Вариант 5. В одномерном динамическом массиве, состоящем из *n* элементов, выполнить поиск элемента по заданному ключу последовательным методом поиска.

Вариант 6. В одномерном динамическом массиве, состоящем из *n* элементов, выполнить поиск элемента по заданному ключу бинарным методом поиска.

Вариант 7. В одномерном динамическом массиве, состоящем из *n* элементов, выполнить поиск элемента по заданному ключу интерполяционным методом поиска.

Вариант 8. В одномерном динамическом массиве, состоящем из *n* элементов, определить количество и индексы элементов массива, больших C.

Вариант 9. В одномерном динамическом массиве, состоящем из *n* элементов, определить количество и индексы элементов массива, меньших C.

Вариант 10. В одномерном динамическом массиве, состоящем из *n* элементов, определить количество и индексы элементов массива, больших A и меньших B (A<B).

Вариант 11. В одномерном динамическом массиве, состоящем из *n* элементов, определить количество и индексы элементов массива, меньших A и больших B (A<B).

Вариант 12. Дано действительное положительное число ε . Методом деления отрезка пополам найти приближённое значение корня уравнения

Рекомендуемая литература

Основная литература

1. Антамошкин, О.А. Программная инженерия. Теория и практика : учебник / О.А. Антамошкин ; Министерство образования и науки Российской Федерации, Сибирский Федеральный университет. – Красноярск : Сибирский федеральный университет, 2012. – 247 с. : ил., табл., схем. – Режим доступа: по подписке. – URL: http://biblioclub.ru/index.php?page=book&id=363975 – Библиогр.: с. 240. – ISBN 978-5-7638-2511-4.

2. Зубкова, Т.М. Технология разработки программного обеспечения : учебное пособие / Т.М. Зубкова ; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего образования «Оренбургский государственный университет», Кафедра программного обеспечения вычислительной техники и автоматизированных систем. – Оренбург : ОГУ, 2017. – 469 с. : ил. – Режим доступа: по подписке. – URL: http://biblioclub.ru/index.php?page=book&id=485553 – Библиогр.: с. 454-459. – ISBN 978-5-7410-1785-2

3. Извозчикова, В.В. Эксплуатация и диагностирование технических и программных средств информационных систем : учебное пособие / В.В. Извозчикова ; Министерство образования и науки Российской Федерации, Оренбургский Государственный Университет, Кафедра программного обеспечения вычислительной техники и автоматизированных систем. – Оренбург : Оренбургский государственный университет, 2017. – 137 с. : ил. – Режим доступа: по подписке. – URL: http://biblioclub.ru/index.php?page=book&id=481761 – Библиогр. в кн. – ISBN 978-5-7410-1746-3.

Дополнительная литература

1. Ехлаков, Ю.П. Управление программными проектами : учебное пособие / Ю.П. Ехлаков ; Министерство образования и науки Российской Федерации, Томский Государственный Университет Систем Управления и Радиоэлектроники (ТУСУР). – Томск : Эль Контент, 2014. – 140 с. : схем., табл. – Режим доступа: по подписке. – URL: http://biblioclub.ru/index.php?page=book&id=480462 – Библиогр.: с. 128-130. – ISBN 978-5-4332-0163-7.

2. Гагарина, Л. Г. Технология разработки программного обеспечения[Текст] : учебное пособие для вузов / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Виснадул. - Москва : Форум : ИНФРА-М, 2015. - 400 с. : ил. - (Высшее образование). - Библиогр. : с. 388-391. (20 экз)

3. Орлов, С. А. Технологии разработки программного обеспечения. Современный курс по программной инженерии [Текст] : учебник для вузов по специальности "Программное обеспечение вычислительной техники и автоматизированных систем" / С. А. Орлов, Б. Я. Цилькер.- 4-е изд. - Москва : Питер, 2012. - 608 с. : ил. - (Учебник для вузов. Стандарт третьего поколения) - (40 экз)

Периодические издания

1. Журнал «Вестник компьютерных и информационных технологий »
2. Журнал «Информационные технологии и вычислительные системы»
3. Журнал «Стандарты и качество»
4. Журнал «Прикладная информатика»