МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ

Методические указания

по выполнению лабораторных работ
по дисциплине
«Тестирования и отладка программного обеспечения»
для направления подготовки **09.03.02 Информационные системы и технологии**
направленность (профиль)**Информационные системы и технологии обработки цифрового контента**

Пятигорск
2026

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

1. Цель и задачи освоения дисциплины

Целью изучения дисциплины является формирование набора универсальных и профессиональных компетенций будущего бакалавра по направлению 09.03.02 “Информационные системы и технологии”.

Задачи освоения дисциплины:

- знакомство с классификациями отказов информационных систем;
- определение показателей надежности информационных систем;
- изучение и моделирование факторов, влияющих на Тестирование и отладка программного обеспечения:
- изучение теории восстановления;
- знакомство с методиками проведения испытаний на надежность.

2. Место дисциплины в структуре основной образовательной программы

Дисциплина «Тестирования и отладка программного обеспечения» входит в вариативную часть блока Б1 ОПВО подготовки бакалавра направления 09.03.02 Информационные системы и технологии. Ее освоение происходит в 5 семестре.

3. Содержание лабораторных работ

№ Темы	Наименование работы	Объем часов
	5 семестр	
1.	**Лабораторная работа 1. Изучение принципов и стратегий тестирования программ.** Экономика тестирования. Тестирование программы методом черного ящика. Тестирование программы методом белого ящика. Принципы тестирования	6
2.	**Лабораторная работа 2. Типы ошибок и автоматизация тестирования программ.** Классификация типов ошибок. Уровни и виды автоматизации тестирования. Автоматизация тестирования и составления отчета	6
3.	**Лабораторная работа 3. Тестирования методом черного ящика.** Эквивалентное разбиение. Анализ граничных значений. Анализ причинно-следственных связей. Предположение об ошибке	6
4.	**Лабораторная работа 4. Тестирование по стратегии белого ящика.** Метод покрытия операторов. Метод покрытия решений (покрытия переходов). Метод покрытия условий. Метод комбинаторного покрытия условий	6
5.	**Лабораторная работа 5. Отладка программ в среде MS Visual Studio.** Точки останова и специальные окна. Отладка и локальные окна. Контрольные значения. Логические ошибки.	6
6.	**Лабораторная работа 6. Автоматизация тестирования. Модульное тестирование на Python.** Простейший пример модульного (Unit) тестирования. Составление модульных тестов.	6
	Итого за 5 семестр	**36**
	Итого	**36**

3. Содержание лабораторных работ

Лабораторная работа № 1. Изучение принципов и стратегий тестирования программ

Цель работы: Изучить понятия, основные принципы и стратегии тестирования программного обеспечения

Теоретическая часть

1. Определение термина «тестирование»

Тестирование – это процесс исполнения программы с целью обнаружения ошибок.

Из приведенного определения тестирования вытекает несколько следствий. Например, одно из них состоит в том, что **тестирование – процесс деструктивный** (т. е. обратный созидательному, конструктивному). Именно этим и объясняется, почему многие программисты и тестировщики считают его трудным. Большинство людей склонны к конструктивному процессу созидания объектов и в меньшей степени – к деструктивному процессу разделения на части. Из определения следует также, как нужно строить набор тестовых данных, и кто должен (а кто не должен) тестировать данную программу.

Для усиления определения тестирования проанализируем два понятия «удачный» и «неудачный» и, в частности, их использование руководителями проектов при оценке результатов тестирования. Некоторые руководители программных проектов называют тестовый прогон «неудачным» если обнаружена ошибка, и, наоборот, удачным, если он прошел без ошибок. Чаще всего это является следствием ошибочного понимания термина «тестирование», так как, по существу, слово «удачный» означает «результативный», а слово «неудачный» – «нежелательный», «нерезультативный». Но если тест не обнаружил ошибки, его выполнение связано с потерей времени и денег, и термин «удачный» никак не может быть применен к нему. Естественно, заранее неизвестно, будет ли тест удачным или неудачным, но построение удачных тестов – отдельная тема.

Тестовый прогон, приведший к обнаружению ошибки, нельзя назвать неудачным хотя бы потому, что, как отмечалось выше, это целесообразное вложение капитала. Отсюда следует, что в слова «удачный» и «неудачный» необходимо вкладывать смысл, обратный общепринятому. Поэтому в дальнейшем *будем называть тестовый прогон удачным, если в процессе его выполнения обнаружена ошибка, и неудачным, если получен корректный результат*.

Таким образом, можно сказать, что *тестирование представляется деструктивным процессом попыток обнаружения ошибок в программе* (наличие которых предполагается). Набор тестов, способствующий обнаружению ошибки, считается удачным. Естественно, в конечном счете, каждый с помощью тестирования хочет добиться определенной степени уверенности в том, что его программа соответствует своему назначению и не делает того, для чего она не предназначена, но лучшим средством для достижения этой цели является непосредственный поиск ошибок.

2. Экономика тестирования

Дав такое определение тестированию, необходимо на следующем шаге рассмотреть возможность создания теста, обнаруживающего все ошибки программы. Но ответ будет отрицательным даже для самых тривиальных программ. В общем случае невозможно обнаружить все ошибки программы. А это в свою очередь порождает экономические

проблемы, задачи, связанные с функциями человека в процессе отладки, способы построения тестов.

2.1 Тестирование программы как черного ящика

Одним из способов изучения поставленного вопроса является исследование стратегии тестирования, называемой *стратегией черного ящика*, тестированием с управлением по данным, или тестированием с управлением по входу-выходу. При использовании этой стратегии программа рассматривается как черный ящик. Иными словами, такое тестирование имеет целью выяснение обстоятельств, в которых поведение программы не соответствует ее спецификации. Тестовые же данные используются только в соответствии со спецификацией программы (т. е. без учета знаний о ее внутренней структуре).

При таком подходе обнаружение всех ошибок в программе является критерием исчерпывающего входного тестирования. Последнее может быть достигнуто, если в качестве тестовых наборов использовать все возможные наборы входных данных. Поскольку программа представляет собой черный ящик, единственный способ удовлетворения приведенному выше критерию – перебор всех возможных входных значений. Значит, требуется бесконечное число тестов.

Таким образом, построение исчерпывающего входного теста невозможно. Это подтверждается двумя аргументами: во-первых, **нельзя создать тест, гарантирующий отсутствие ошибок**; во-вторых, разработка таких тестов противоречит экономическим требованиям.

Поскольку исчерпывающее тестирование исключается, *целью тестирования должна стать* максимизация результативности капиталовложений в тестирование (иными словами, *максимизация числа ошибок, обнаруживаемых одним тестом*). Для этого мы можем рассматривать внутреннюю структуру программы и делать некоторые разумные, но, конечно, не обладающие полной гарантией достоверности предположения.

2.2. Тестирование программы как белого ящика

Стратегия белого ящика, или стратегия тестирования, управляемого логикой программы, позволяет исследовать внутреннюю структуру программы. В этом случае тестирующий получает тестовые данные путем анализа логики программы.

Сравним способ построения тестов при данной стратегии с исчерпывающим входным тестированием стратегии черного ящика. Непосвященному может показаться, что достаточно построить такой набор тестов, в котором каждый оператор исполняется хотя бы один раз; нетрудно показать, что это неверно. Не вдаваясь в детали, укажем лишь, что исчерпывающему входному тестированию может быть поставлено в соответствие исчерпывающее тестирование маршрутов. Подразумевается, что программа проверена полностью, если с помощью тестов удается осуществить выполнение этой программы по всем возможным маршрутам ее потока (графа) передач управления.

Последнее утверждение имеет два слабых пункта. Один из них состоит в том, что число не повторяющих друг друга маршрутов в программе – астрономическое. Чтобы убедиться в этом, рассмотрим представленный на рис. 1 граф передач управления в простейшей программе. Каждая вершина, или кружок, обозначают участок программы, содержащий последовательность линейных операторов, которая может заканчиваться

оператором ветвления. Дуги, оканчивающиеся стрелками, соответствуют передачам управления. По-видимому, граф описывает программу из 10–20 операторов, включая цикл **WHILE** (или DO WHILE), который исполняется не менее 20 раз (на рисунке показан темным цветом). Внутри цикла имеется несколько операторов **IF** (на рисунке соответствующие узлы графа изображены пустыми кружками). Для того чтобы определить число неповторяющихся маршрутов при исполнении программы, подсчитаем число неповторяющихся маршрутов из точки **A** в **B** в предположении, что все переходы взаимно независимы. Это число вычисляется как сумма $5^{20} + 5^{19} + ... + 5^1 = 10^{14}$, или 100 триллионов, где 5 – число путей внутри цикла. Поскольку большинству людей трудно оценить это число, приведем такой пример: если допустить, что на составление каждого теста мы тратим пять минут, то для построения набора тестов нам потребуется примерно один миллиард лет.

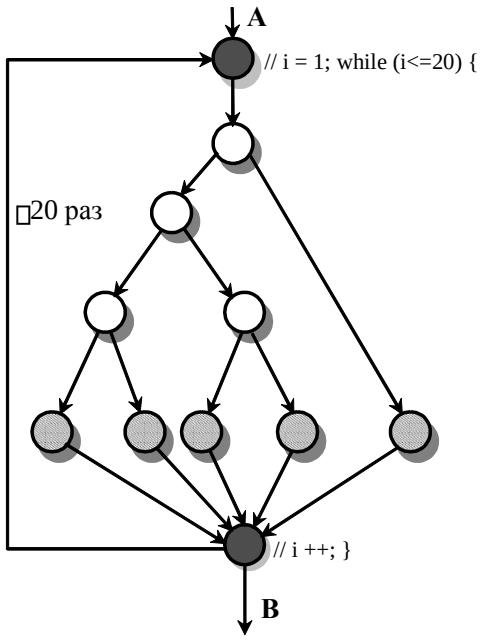

Рисунок 1 - Граф передач управления небольшой программы

Конечно, в реальных программах условные переходы не могут быть взаимно независимы, т. е. число маршрутов исполнения будет несколько меньше. С другой стороны, реальные программы значительно больше, чем простая программа, представленная на рис. 1. Следовательно, исчерпывающее тестирование маршрутов, как и исчерпывающее входное тестирование, не только невыполнимо, но и невозможно.

Второй слабый пункт утверждения заключается в том, что, хотя исчерпывающее тестирование маршрутов является полным тестом, и хотя каждый маршрут программы может быть проверен, сама программа будет содержать ошибки. Это объясняется следующим образом.

Во-первых, исчерпывающее тестирование маршрутов не может дать гарантии того, что программа соответствует описанию. Например, вместо требуемой программы сортировки по возрастанию случайно была написана программа сортировки по убыванию. В этом случае ценность тестирования маршрутов невелика, поскольку после тестирования в программе окажется одна ошибка, т. е. программа неверна.

Во-вторых, программа может быть неверной в силу того, что пропущены некоторые маршруты. Исчерпывающее тестирование маршрутов не обнаружит их отсутствия.

В-третьих, исчерпывающее тестирование маршрутов не может обнаружить ошибок, появление которых зависит от обрабатываемых данных. Существует множество примеров таких ошибок. Приведем один из них. Допустим, в программе необходимо выполнить сравнение двух чисел на сходимость, т. е. определить, является ли разность между двумя числами меньше предварительно определенного числа. Может быть написано выражение

IF ((a – b) < epsilon)…

Безусловно, оно содержит ошибку, поскольку необходимо выполнить сравнение абсолютных величин. Однако обнаружение этой ошибки зависит от значений, использованных для a и b, и ошибка не обязательно будет обнаружена просто путем исполнения каждого маршрута программы.

В заключение отметим, что, хотя исчерпывающее входное тестирование предпочтительнее исчерпывающего тестирования маршрутов, ни то, ни другое не могут стать полезными стратегиями, потому что оба они нереализуемы. Возможно, поэтому реальным путем, который позволит создать хорошую, но, конечно, не абсолютную стратегию, является сочетание тестирования программы как черного и как белого ящиков. Вопрос выбора методов тестирования и их описание будет рассмотрен в дальнейшем.

3. Принципы тестирования

Сформулируем основные принципы тестирования.

1. Описание предполагаемых значений выходных данных или результатов должно быть необходимой частью тестового набора.
2. Следует избегать тестирования программы ее автором.
3. Программирующая организация не должна сама тестировать разработанные ею программы.
4. Необходимо досконально изучать результаты применения каждого теста.
5. Тесты для неправильных и непредусмотренных входных данных следует разрабатывать так же тщательно, как для правильных и предусмотренных.
6. Необходимо проверять не только, делает ли программа то, для чего она предназначена, но и не делает ли она то, что не должна делать.
7. Не следует выбрасывать тесты, даже если программа уже не нужна.
8. Нельзя планировать тестирование в предположении, что ошибки не будут обнаружены.
9. Вероятность наличия необнаруженных ошибок в части программы пропорциональна числу ошибок, уже обнаруженных в этой части.
10. Тестирование — процесс творческий.

Оборудование и материалы: для выполнения данной лабораторной работы необходим компьютер с установленной операционной системой Windows 7 или выше, одной из систем программирования Microsoft Visual Studio, IDLE (Python 3.*) или др., а также программой MS Word для подготовки отчета.

Указания по технике безопасности: к выполнению лабораторных работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Порядок выполнения работы:

1. Изучить теоретический материал.
2. В таблице 1 приведены варианты индивидуальных заданий, а в листинге 1 - программы на языке Python для решения этих задач, содержащие, по крайней мере, две логические ошибки.
3. Для обнаружения ошибок в программе составьте не менее 10 тестов в виде таблицы 2 с использованием стратегии черного ящика.
4. Наберите текст предложенной программы и выполните ее на каждом из подготовленных тестов.
5. Заполните таблицу с результатами тестирования (по образцу таблицы 3).
6. Найдите и исправьте обнаруженные в процессе тестирования ошибки (нужно именно внести исправления в предложенную программу, а не написать свой вариант ее решения).
7. Оформить отчет в электронном виде. Отчет по лабораторной работе должен быть выполнен в редакторе MS Word и содержать:
   - титульный лист с темой лабораторной работы;
   - цель работы;
   - вариант задания и исходную программу;
   - таблицу с составленными тестами;
   - результаты первого тестирования;
   - исправленный вариант программы;
   - результаты повторного тестирования;
   - выводы о проделанной работе.

Лабораторная работа № 2. Типы ошибок и автоматизация тестирования программ

Цели работы:

- изучить основные типы ошибок в программах;
- научиться автоматизировать прогон тестов и составление отчета о результатах тестирования.

Теоретическая часть

1. Классификация ошибок

Задача любого *тестировщика* заключается в нахождении наибольшего количества ошибок, поэтому он должен хорошо знать наиболее часто допускаемые ошибки и уметь находить их за минимально короткий период времени. Остальные ошибки, которые не являются типовыми, обнаруживаются только тщательно созданными наборами тестов. Однако из этого не следует, что для типовых ошибок не нужно составлять тесты.

Для классификации ошибок мы должны определить термин «ошибка».

Ошибка **–** это расхождение между вычисленным, наблюдаемым и истинным, заданным или теоретически правильным значением**.**

Итак, *по времени появления* ошибки можно разделить на три вида:

- ***структурные ошибки набора;***
- ***ошибки компиляции;***
- ***ошибки периода выполнения.***

Структурные ошибки возникают непосредственно при наборе программы.

Например, при работе в среде разработки Microsoft Visual Basic, если набрать оператор **If**, затем сравнение и нажать на клавишу **Enter**, не набрав слова **Then**, то Visual Basic укажет, что возникла ошибка компиляции. Это не совсем верно, так как компиляция в Visual Basic происходит только непосредственно при выполнении команды программы. В данном случае мы имеем дело именно со структурной ошибкой набора.

Данный тип ошибок определяется либо при наборе программы (самой IDE (**I**ntegrated **D**evelopment **E**nvironment) – интегрированной средой разработки) или при ее компиляции, если среда не разделяет первые два типа ошибок.

К данному типу ошибок относятся такие как: несоответствие числа открывающих скобок числу закрывающих, отсутствие парного оператора (например, **try** без **catch**), неправильное употребление синтаксических знаков и т. п.

Во многих средах разработки программного обеспечения данный тип ошибок объединяется со следующим типом, так как раннее определение ошибок вызывает некоторое неудобство при наборе программ (скажем, я задумал что-то написать, а потом вспомнил, что в начале пропустил оператор, тогда среда разработки может выдать мне ошибку при попытке перейти в другую строку).

Еще раз нужно отметить, что данный тип ошибок достаточно уникален и выделяется в отдельный тип только некоторыми средами разработки программного обеспечения.

Ошибки компиляции возникают из-за ошибок в тексте кода. Они включают ошибки в синтаксисе, неверное использование конструкций языка, использование несуществующих объектов или свойств, методов у объектов.

Среда разработки (компилятор) обнаружит эти ошибки при общей компиляции приложения и сообщит о **последствиях** этих ошибок. Необходимо подчеркнуть слово «последствия» – это очень важно. Дело в том, что часто, говоря об ошибках, мы не разделяем проявление ошибки и саму ошибку, хотя это и не одно и то же. Например, ошибка «неопределенный класс» не означает, что класс не определен. Он может быть неподключенным, так как не подключен пакет классов.

Ошибки периода выполнения возникают, когда программа выполняется и компилятор (или операционная система, виртуальная машина) обнаруживает, что оператор делает попытку выполнить недопустимое или невозможное действие. Например, деление на ноль. Предположим, имеется такое выражение:

```
ratio = firstValue / sum.
```

Если переменная sum содержит ноль, то деление – недопустимая операция, хотя сам оператор синтаксически правилен. Прежде, чем программа обнаружит эту ошибку, ее необходимо запустить на выполнение.

Хотя данный тип ошибок называется «ошибками периода выполнения», это не означает, что ошибки находятся только после запуска программы. Вы можете выполнять программу в уме и обнаружить ошибки данного типа, однако, понятно, что это крайне неэффективно.

Если проанализировать все типы ошибок согласно первой классификации, то можно прийти к заключению, что при тестировании приходится иметь дело с ошибками периода выполнения, так как первые два типа ошибок определяются на этапе кодирования.

В теоретической информатике программные ошибки классифицируют *по степени нарушения логики на*:

- *синтаксические;*
- *семантические;*
- *прагматические.*

Синтаксические ошибки заключаются в нарушении правописания или пунктуации в записи выражений, операторов и т. п., т. е. в нарушении грамматических правил языка.

В качестве примеров синтаксических ошибок можно назвать:

- пропуск необходимого знака пунктуации;
- несогласованность скобок;
- пропуск нужных скобок;
- неверное написание зарезервированных слов;
- отсутствие описания массива.

Все ошибки данного типа обнаруживаются компилятором.

Семантические ошибки заключаются в нарушении порядка операторов, параметров функций и употреблении выражений.

Надо отметить, что некоторые исследователи относят семантические ошибки к следующей группе ошибок.

Прагматические ошибки (или логические) заключаются в неправильной логике алгоритма, нарушении смысла вычислений и т. п. Они являются самыми сложными и крайне трудно обнаруживаются. Компилятор может выявить только следствие прагматической ошибки (см. выше пример с делением на ноль, компилятор обнаружит деление на ноль, но когда и почему переменная sum стала равна нулю – должен найти программист).

Таким образом, после рассмотрения двух классификаций ошибок можно прийти к выводу, что на ***этапе тестирования ищутся прагматические ошибки периода выполнения***, так как остальные выявляются в процессе программирования.

На этом можно было бы закончить рассмотрение классификаций, но с течением времени накапливался опыт обнаружения ошибок и сами ошибки, некоторые из которых образуют характерные группы, которые могут тоже служить характерной классификацией.

Ошибка адресации – ошибка, состоящая в неправильной адресации данных (например, выход за пределы участка памяти).

Ошибка ввода-вывода – ошибка, возникающая в процессе обмена данными между устройствами памяти, внешними устройствами.

Ошибка вычисления – ошибка, возникающая при выполнении арифметических операций (например, разнотипные данные, деление на нуль и др.).

Ошибка интерфейса – программная ошибка, вызванная несовпадением характеристик фактических и формальных параметров (как правило, семантическая ошибка периода компиляции, но может быть и логической ошибкой периода выполнения).

Ошибка обращения к данным – ошибка, возникающая при обращении программы к данным (например, выход индекса за пределы массива, не инициализированные значения переменных и др.). **Ошибка описания данных** – ошибка, допущенная в ходе описания данных.

2. Автоматизация тестирования и составления отчета

Если разработанные тесты записать в текстовый файл и написать программу, которая будет последовательно считывать очередной тест из файла, прогонять на нем программу и записывать ожидаемый и фактический результаты в файл отчета, то процесс тестирования и документирования будет выполняться автоматически.

Напомним, что в языке Python чтение из открытого для чтения файла выполняется с помощью методов **read(), readline()** и **readlines().** Запись в открытый для записи файл выполняется с помощью методов **write()** и **writeline().**

Оборудование и материалы: для выполнения данной лабораторной работы необходим компьютер с установленной операционной системой Windows 7 или выше, одной из систем программирования Microsoft Visual Studio, IDLE (Python 3.*) или др., а также программой MS Word для подготовки отчета.

Указания по технике безопасности: к выполнению лабораторных работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Порядок выполнения работы:

1. Записать построчно в текстовый файл tests.txt тесты, составленные в лабораторной работе 1. Значения входных переменных и ожидаемый результат должны быть заданы в одной строке файла через пробел.
2. Оформить первоначальный вариант программы из лабораторной 1 в виде функции **program1()**, а исправленный – в виде функции **program2().**
3. Написать функцию **auto_test()** на языке Python, которая будет считывать из текстового файла **tests.txt** подготовленные тесты, выполнять программу **program1()** на каждом из тестов, сравнивать полученный результат с ожидаемым, и записывать выводы по каждому тесту в файл **report1.txt**.
4. Выполнить функцию **auto_test** для исходного и исправленного вариантов программы. Результаты тестирования исправленного варианта записать в файл **report2.txt.**
5. Оформить отчет в электронном виде. Отчет по лабораторной работе должен быть выполнен в редакторе MS Word и содержать:
   - титульный лист с темой лабораторной работы;
   - цель работы;
   - вариант задания и исходную программу;
   - скриншоты функций **program1()** и **auto_test;**
   - скриншот файла **tests.txt** с тестами;
   - скриншот файла **report1.txt**;

- скриншоты функций **program2()** и **auto_test;**;
- скриншот файла **report2.txt**;
- выводы о проделанной работе.

Лабораторная работа № 3. Тестирования методом черного ящика.

Цель работы: Получить навыки использования методов тестирования программного обеспечения с использованием стратегии черного ящика

Теоретическая часть

Одним из способов проверки программ является стратегия тестирования, называемая стратегией "черного ящика" или тестированием с управлением по данным. В этом случае программа рассматривается как "черный ящик", и такое тестирование имеет целью выяснение обстоятельств, в которых поведение программы не соответствует спецификации.

Для обнаружения всех ошибок в программе необходимо выполнить *исчерпывающее тестирование*, т.е. тестирование на всех возможных наборах данных. Для тех же программ, где исполнение команды зависит от предшествующих ей событий, необходимо проверить и все возможные последовательности.

Очевидно, что построение исчерпывающего входного теста для большинства случаев невозможно. Поэтому, обычно выполняется *"разумное" тестирование*, при котором тестирование программы ограничивается прогонами на небольшом подмножестве всех возможных входных данных. Естественно при этом целесообразно выбрать наиболее подходящее подмножество (подмножество с наивысшей вероятностью обнаружения ошибок).

Правильно выбранный тест подмножества должен обладать следующими свойствами:

1) уменьшать, причем более чем на единицу число других тестов, которые должны быть разработаны для достижения заранее определенной цели «приемлемого» тестирования:

2) покрывать значительную часть других возможных тестов, что в некоторой степени свидетельствует о наличии или отсутствии ошибок до и после применения, этого ограниченного множества значений входных данных.

Стратегия "черного ящика" включает в себя следующие методы формирования тестовых наборов:

- эквивалентное разбиение;
- анализ граничных значений;
- анализ причинно-следственных связей;
- предположение об ошибке.

Эквивалентное разбиение

Основу метода составляют два положения:

1. Исходные данные программы необходимо разбить на конечное число классов эквивалентности так, чтобы можно было предположить, что каждый тест, являющийся представителем некоторого класса, эквивалентен любому другому тесту этого класса. Иными словами, если тест какого-либо класса обнаруживает ошибку, то

предполагается, что все другие тесты этого класса эквивалентности тоже обнаружат эту ошибку и наоборот.

2. Каждый тест должен покрывать по возможности максимальное количество различных входных условий, что позволяет минимизировать общее число необходимых тестов. Первое положение используется для разработки набора "интересных" условий, которые должны быть протестированы, а второе - для разработки минимального набора тестов.

Разработка тестов методом эквивалентного разбиения осуществляется в два этапа:

- выделение классов эквивалентности;
- построение тестов.

Выделение классов эквивалентности

Классы эквивалентности выделяются путем выбора каждого входного условия (обычно это предложение или фраза из спецификации) и разбиением его на две или более групп. Для этого используется таблица следующего вида:

Входное условие	Правильные классы эквивалентности	Неправильные классы эквивалентности

Правильные классы включают правильные данные, неправильные классы - неправильные данные. Выделение классов эквивалентности является эвристическим процессом, однако при этом существует ряд правил:

Если входные условия описывают *область* значений (например, «целое данное может принимать значения от 1 до 999»), то выделяют один правильный класс $1 \le X \le 999$ и два неправильных $X<1$ и $X>999$.

Если входное условие описывает *число* значений (например, «в автомобиле могут ехать от одного до шести человек»), то определяется один правильный класс эквивалентности и два неправильных (ни одного и более шести человек).

Если входное условие описывает множество входных значений и есть основания полагать, что каждое значение программист трактует особо (например, «известные способы передвижения на АВТОБУСЕ, ГРУЗОВИКЕ, ТАКСИ, МОТОЦИКЛЕ или ПЕШКОМ»), то определяется правильный класс эквивалентности для каждого значения и один неправильный класс (например «на ПРИЦЕПЕ»).

Если входное условие описывает ситуацию «должно быть» (например, «первым символом идентификатора должна быть буква»), то определяется один правильный класс эквивалентности (первый символ - буква) и один неправильный (первый символ - не буква).

Если есть любое основание считать, что различные элементы класса эквивалентности трактуются программой неодинаково, то данный класс разбивается на меньшие классы эквивалентности.

Построение тестов.

Этот шаг заключается в использовании классов эквивалентности для построения тестов. Этот процесс включает в себя:

- назначение каждому классу эквивалентности уникального номера;

– проектирование новых тестов, каждый из которых покрывает как можно большее число непокрытых классов эквивалентности, до тех пор, пока все правильные классы не будут покрыты (только не общими) тестами;

– запись тестов, каждый из которых покрывает один и только один из непокрытых неправильных классов эквивалентности, до тех пор, пока все неправильные классы не будут покрыты тестами.

Разработка индивидуальных тестов для неправильных классов эквивалентности обусловлена тем, что определенные проверки с ошибочными входами скрывают или заменяют другие проверки с ошибочными входами. Недостаток метода эквивалентных разбиений в том, что он не исследует комбинации входных условий.

Анализ граничных значений

Граничные условия - это ситуации, возникающие на границе, выше или ниже границ входных классов эквивалентности. Анализ граничных значений отличается от эквивалентного разбиения следующим:

выбор любого элемента в классе эквивалентности в качестве представительного при анализе граничных условий осуществляется таким образом, чтобы проверить тестом каждую границу этого класса;

при разработке тестов рассматриваются не только входные условия (пространство входов), но и пространство результатов.

Применение метода анализа граничных условий требует определенной степени творчества и специализации в рассматриваемой проблеме. Тем не менее, существует несколько общих правил этого метода:

Построить тесты для границ области и тесты с неправильными входными данными для ситуаций незначительного выхода за границы области, если входное условие описывает область значений (например, для области входных значений от -1.0 до +1.0 необходимо написать тесты для ситуаций -1.0, +1.0, -1.001 и +1.001).

Построить тесты для минимального и максимального значений условий и тесты, большие и меньшие этих двух значений, если входное условие удовлетворяет дискретному ряду значений. Например, если входной файл может содержать от 1 до 255 записей, то проверить 0, 1, 255 и 256 записей.

Использовать правило 1 для каждого выходного условия. Причем, важно проверить границы пространства результатов, поскольку не всегда границы входных областей представляют такой же набор условий, как и границы выходных областей. Не всегда также можно получить результат вне выходной области, но, тем не менее, стоит рассмотреть эту возможность.

Использовать правило 2 для каждого выходного условия.

Если вход или выход программы есть упорядоченное множество (например, последовательный файл, линейный список, таблица), то сосредоточить внимание на первом и последнем элементах этого множества.

Попробовать свои силы в поиске других граничных условий.

Анализ граничных условий, если он применен правильно, является одним из наиболее полезных методов проектирования тестов. Однако следует помнить, что граничные условия могут быть едва уловимы и определение их связано с большими

трудностями, что является недостатком этого метода. Второй недостаток связан с тем, что метод анализа граничных условий не позволяет проверять различные сочетания исходных данных.

Анализ причинно-следственных связей

Метод анализа причинно-следственных связей помогает системно выбирать высоко результативные тесты. Он дает полезный побочный эффект, позволяя обнаруживать неполноту и неоднозначность исходных спецификаций.

Для использования метода необходимо понимание булевской логики (логических операторов - и, или, не). Построение тестов осуществляется в несколько этапов.

1) Спецификация разбивается на «рабочие» участки, так как таблицы причинноследственных связей становятся громоздкими при применении метода к большим спецификациям. Например, при тестировании компилятора в качестве рабочего участка можно рассматривать отдельный оператор языка.

2) В спецификации определяются множество причин и множество следствий. *Причина* есть отдельное входное условие или класс эквивалентности входных условий. *Следствие* есть выходное условие или преобразование системы. Каждым причине и следствию приписывается отдельный номер.

3) На основе анализа семантического (смыслового) содержания спецификации строится таблица истинности, в которой последовательно перебираются все возможные комбинации причин и определяются следствия каждой комбинации причин. Таблица снабжается примечаниями, задающими ограничения и описывающими комбинации причин и/или следствий, которые являются невозможными из-за синтаксических или внешних ограничений. Аналогично, при необходимости строится таблица истинности для класса эквивалентности.

Примечание. При этом можно использовать следующие приемы:

1. по возможности выделять независимые группы причинно-следственных связей в отдельные таблицы;
2. истина обозначается "1", ложь обозначается "0", для обозначения безразличных состояний условий применять обозначение "x", которое предполагает произвольное значение условия (0 или 1).
3. Каждая строка таблицы истинности преобразуется в тест. При этом по возможности следует совмещать тесты из независимых таблиц. Недостаток метода - неадекватно исследует граничные условия.

Предположение об ошибке

Часто программист с большим опытом выискивает ошибки "без всяких методов". При этом он подсознательно использует метод "предположение об ошибке". Процедура метода предположения об ошибке в значительной степени основана на интуиции. Основная идея метода состоит в том, чтобы перечислить в некотором списке возможные ошибки или ситуации, в которых они могут появиться, а затем на основе этого списка составить тесты. Другими словами, требуется перечислить те специальные случаи, которые могут быть не учтены при проектировании.

Оборудование и материалы: для выполнения данной лабораторной работы необходим компьютер с установленной операционной системой Windows 7 или выше, одной из систем программирования Microsoft Visual Studio, Pascal ABC или др., а также программой MS Word для подготовки отчета.

Указания по технике безопасности: к выполнению лабораторных работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Порядок выполнения работы:

1. Ознакомьтесь с теоретическими сведениями по стратегиям тестирования.
2. Подготовьте тесты по методикам стратегии "черного ящика" для программы, решающей задачу, определяемую вариантом задания. Предлагаемые тесты сведите в следующую таблицу.

Номер теста	Назначение теста	Значения исходных данных	Ожидаемый результат	Реакция программы	Вывод (успешно/ неуспешно)

3. Напишите программу, реализующую заданный вариантом задания алгоритм обработки данных.
4. Сохраните в отчете первоначальный вариант программы.
5. Проведите тестирование программы и заполните подготовленную ранее таблицу с тестами. Сохраните таблицу в отчете.
6. Устраните в программе ошибки, выявленные по результатам тестирования. После этого повторно проведите тестирование по тем тестам, которые ранее были не пройдены. Приведите таблицу с новыми результатами тестирования.
7. Сделайте вывод о роли тестирования с использованием стратегии "черного ящика" и возможностях его применения. Сформулируйте его достоинства и недостатки.
8. Оформите отчет по лабораторной работе.
9. Ответьте на контрольные вопросы.

Лабораторная работа № 4. Тестирование по стратегии белого ящика.

Цель работы: Получить навыки использования методов тестирования программного обеспечения с использованием стратегии белого ящика

Теоретическая часть

Тестирование программного обеспечения охватывает целый ряд видов деятельности, аналогичных последовательности процессов разработки программного обеспечения. В него входят:

а) постановка задачи для теста,

б) проектирование теста,

в) написание тестов,

г) тестирование тестов,

д) выполнение тестов,

е) изучение результатов тестирования.

Решающую роль играет проектирование тестов. Возможны разные подходы к проектированию тестов. Первый состоит в том, что тесты проектируются на основе внешних спецификаций программ и модулей, либо спецификаций сопряжения программы или модуля. Программа при этом рассматривается как черный ящик (стратегия ‘черного ящика’). Существо такого подхода - проверить соответствует ли программа внешним спецификациям. При этом логика модуля совершенно не принимается во внимание.

Второй подход основан на анализе логики программы (стратегия ‘белого ящика’). Существо подхода - в проверке каждого пути, каждой ветви алгоритма. При этом внешняя спецификация во внимание не принимается.

Ни один из этих подходов не является оптимальным. Из анализа существа первого подхода ясно, что его реализация сводится к проверке всех возможных комбинаций значений на входе программы. Тестирование любой программы для всех значений входных данных невозможно, так как их бесконечное множество, поэтому ограничиваются меньшим. 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.

Проанализируем теперь второй подход к тестированию. Даже если предположить, что выполнены тесты для всех путей программы, нельзя с полной уверенностью утверждать, что модуль не содержит ошибок.

Очевидное основание этого утверждения состоит в том, что выполнение всех путей не гарантирует соответствия программы ее спецификациям. Допустим, если требовалось написать программу для вычисления кубического корня, а программа фактически вычисляет корень квадратный, то программа 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. Путь может правильно выполняться для одних данных и неправильно для других. Например, если для определения равенства трех чисел программируется выражение вида:

if (A+B+C)/3==A,

то оно будет верным не для всех значений A, B и C (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. Тест для любой программы будет обязательно неполным, то есть тестирование не гарантирует полное отсутствие ошибок в программе. Стратегия проектирования тестов заключается в том, чтобы попытаться уменьшить эту неполноту насколько это возможно.

Тестирование по принципу белого ящика характеризуется степенью, какой тесты выполняют или покрывают логику (исходный текст программы).

Стратегия белого ящика включает в себя пять методов:

- 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:

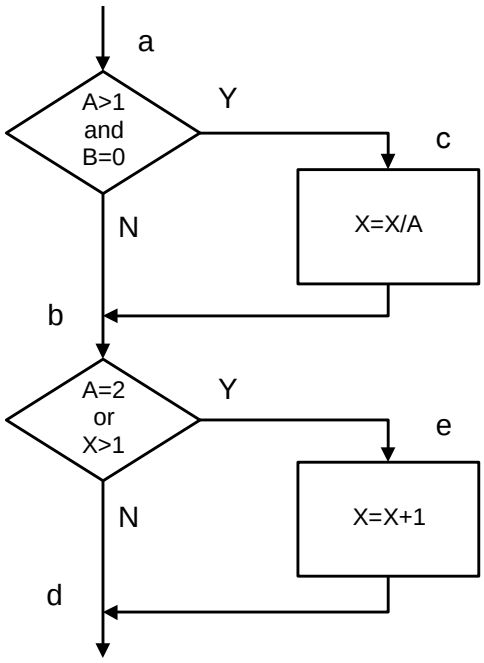

Рисунок 1

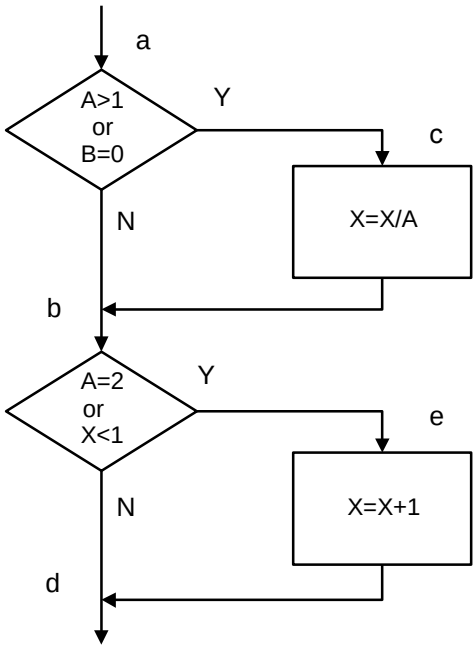

Рисунок 2

В этой программе можно выполнить каждый оператор, записав один единственный тест, который реализовал бы путь *ace.* Т.е., если бы на входе было: A=2, B=0, X=3, каждый оператор выполнился бы один раз. Но этот критерий на самом деле хуже, чем он кажется на первый взгляд. Пусть в первом условии вместо "*and"* будет*"or"* и во втором вместо *"x>1"* будет *"x<1"* (блок-схема правильной программы приведена на рисунке 1, а неправильной - на рисунке 2).

Результаты тестирования приведены в таблице 1. Обратите внимание: ожидаемый результат определяется по алгоритму на рисунке 1, а фактический - по алгоритму рисунка 2, поскольку определяется чувствительность метода тестирования к ошибкам программирования. Как видно из этой таблицы, ни одна из внесенных в алгоритм ошибок не будет обнаружена.

Таблица 1 - Результат тестирования методом покрытия операторов

Тест	Путь	Ожидаемый результат	Фактический результат	Результат тестирования

A=2, B=0, X=3	*ace*	X=2,5	X=2,5	неуспешно

Метод покрытия решений (покрытия переходов)

Более сильный метод тестирования известен как покрытие решений (покрытие переходов). Согласно данному методу каждое направление перехода должно быть реализовано, по крайней мере, один раз.

Покрытие решений обычно удовлетворяет критерию покрытия операторов. Поскольку каждый оператор лежит на некотором пути, исходящем либо из оператора перехода, либо из точки входа программы, при выполнении каждого направления перехода каждый оператор должен быть выполнен.

Для программы, приведенной на рисунке 2, покрытие решений может быть выполнено двумя тестами, покрывающими пути {*ace, abd*}, либо {*acd,abe*}. Пути {*acd,abe*} покроим, выбрав следующие исходные данные: {A=3, B=0, X=3} и {A=2, B=1, X=1}
(результаты тестирования - в таблице 2).

Таблица 2 - Результат тестирования методом покрытия решений

Тест	Пути	Ожидаемый результат	Фактический результат	Результат тестирования
A=3, B=0, X=3	*acd*	X=1	X=1	неуспешно
A=2, B=1, X=1	*abe*	X=2	X=1,5	успешно

Метод покрытия условий

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

В предыдущем примере имеем четыре условия: {A>1, B=0}, {A=2, X>1}. Следовательно, требуется достаточное число тестов, такое, чтобы реализовать ситуации, где A>1, A□1, B=0 и B□0 в точке *a* и A=2, A□2, X>1 и X□1 в точке b. Тесты, удовлетворяющие критерию покрытия условий и соответствующие им пути, приведены в таблице 3.

Таблица 3 - Результаты тестирования методом покрытия условий

Тест	Пути	Ожидаемый результат	Фактический результат	Результат тестирования
A=2, B=0, X=4	*ace*	X=3	X=3	неуспешно
A=1, B=1, X=0	*abd*	X=0	X=1	успешно

Метод покрытия решений / условий

Метод покрытия решений/условий требует такого достаточного набора тестов, чтобы все возможные результаты каждого условия в решении 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. Два теста метода покрытия условий

а) A=2, B=0, X=4 *ace*

б) A=1, B=1, X=0 *abd*

отвечают и критерию покрытия решений/условий. Это является следствием того, что одни условия приведенных решений скрывают другие условия в этих решениях. Так, если условие A>1 будет ложным, транслятор может не проверять условия B=0, поскольку при любом результате условия B=0, результат решения (A>1) and (B==0) примет значение *ложь*. Следовательно, недостатком критерия покрытия решений/условий является невозможность его применения для выполнения всех результатов всех условий.

Другая реализация рассматриваемого примера приведена на рисунке 3. Многоусловные решения исходной программы разбиты на отдельные решения и переходы.

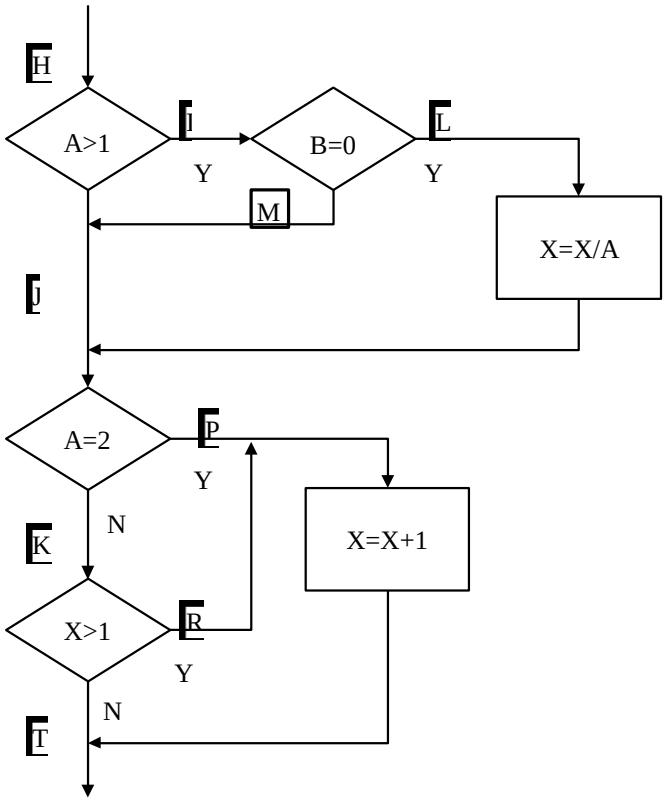

Рисунок 3

Наиболее полное покрытие тестами в этом случае выполняется так, чтобы выполнялись все возможные результаты каждого простого решения. Для этого нужно покрыть пути HILP (тест A=2,B=0,X=4), HIMKT (тест A=3, B=1, X=0), HJKT (тест A=0, B=0, X=0), HJKR (тест A=0, B=0, X=2).

Протестировав алгоритм на рисунке 3, нетрудно убедиться в том, что критерии покрытия условий и критерии покрытия решений/условий недостаточно чувствительны к ошибкам в логических выражениях.

Метод комбинаторного покрытия условий

Критерием, который решает эти и некоторые другие проблемы, является комбинаторное покрытие условий. Он требует создания такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении выполнялись, по крайней мере, один раз. Набор тестов, удовлетворяющих критерию комбинаторного покрытия условий, удовлетворяет также и критериям покрытия решений, покрытия условий и покрытия решений/условий.

По этому критерию в рассматриваемом примере должны быть покрыты тестами следующие восемь комбинаций:

а) A>1, B=0;

б)A>1, B□0;

в) A□1, B=0;

г) A□1, B□0;

д) A=2, X>1;

е) A=2, X□1;

ж) A□2, X>1;

з) A□2, X□1;

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

- A=2, B=0, X=4 {покрывает c, e};

- A=2, B=1, X=1 {покрывает b, e};

- A=0,5, B=0, X=2 {покрывает b, e}; - A=1, B=0, X=1 {покрывает c, d}.

Таблица 4 - Результаты тестирования методом комбинаторного покрытия условий

Тест	Пути	Ожидаемый результат	Фактический результат	Результат тестирования
A=2, B=0, X=4	*ace*	X=3	X=3	неуспешно
A=2, B=1, X=1	*abe*	X=2	X=1,5	успешно
A=0,5 B=0, X=2	*abe*	X=3	X=4	успешно
A=1, B=0, X=1	*acd*	X=1	X=1	неуспешно

Таким образом, для программ, содержащих только одно условие на каждое решение, минимальным является критерий, набор тестов которого:

1) вызывает выполнение всех результатов каждого решения, по крайней мере, один

раз;

2) передает управление каждой точке входа (например, точке входа, **case**-единице) по крайней мере, один раз (чтобы обеспечить выполнение каждого оператора программы, по крайней мере, один раз).

Для программ, содержащих решения, каждое из которых имеет более одного условия, минимальный критерий состоит из набора тестов, вызывающих выполнение всех возможных комбинаций результатов условий в каждом решении и передающих управление каждой точке входа программы, по крайней мере, один раз. Слово «возможных» употреблено здесь потому, что некоторые комбинации условий могут быть нереализуемыми; например, в выражении (a > 2) and (a < 10) могут быть реализованы только три комбинации условий.

Оборудование и материалы: для выполнения данной лабораторной работы необходим компьютер с установленной операционной системой Windows 7 или выше, одной из систем программирования Microsoft Visual Studio, Pascal ABC или др., а также программой MS Word для подготовки отчета.

Указания по технике безопасности: к выполнению лабораторных работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Порядок выполнения работы:

1. Для программы, составленной в лабораторной работе 3, представить алгоритм решения задачи в виде блок-схемы. Обозначить буквами или цифрами ветви алгоритма. Вместо блок-схемы разрешается просто включить нумерацию строк кода.
2. Выбрать метод тестирования по стратегии белого ящика, который, по Вашему мнению, может дать наибольшую вероятность обнаружения ошибок в программе.
3. Выписать пути алгоритма, которые должны быть проверены тестами для выбранного метода тестирования.
4. Записать тесты, которые позволят пройти по путям алгоритма, выбранным Вами в п.3.
5. Протестировать разработанную Вами программу. Результаты оформить в виде таблицы (см. таблицы 1-4).
6. Оформить отчет по лабораторной работе.
7. Ответить на контрольные вопросы.

Замечание. Перед проверкой логических условий рекомендуется проверять корректность вводимых данных (например, могут ли введенные три числа быть сторонами треугольника).

Содержание отчета: отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям.

Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25.

Текст должен быть выровнен по ширине.

Отчет должен содержать:

– титульный лист с темой лабораторной работы;

– цель работы;

– программу для решения задачи;

– блок-схему алгоритма решения задачи (по желанию);

– обоснование выбора метода тестирования по стратегии белого ящика;

– перечисление реализующих выбранный метод тестирования путей алгоритма и тесты для прохождения этих путей;

– таблицу с результатами тестирования программы;

– выводы по результатам тестирования (не забывайте, что целью тестирования является обнаружение ошибок в программе).

Лабораторная работа № 5. Отладка программ в среде MS Visual Studio

Цель работы: познакомиться с инструментами отладки программы в среде Microsoft Visual Studio.

Теоретическая часть

Отладка – этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки. Чтобы понять, где возникла ошибка, приходится узнавать текущие значения переменных и выяснять, по какому пути выполнялась программа.

В большинстве сред разработки содержатся средства, позволяющие проанализировать процесс выполнения программы, получить информацию о текущих значениях переменных, об имеющихся синтаксических ошибках и их детализации.

В процессе написания программы допускаются синтаксические ошибки (часто по причине невнимательности), которые легко отслеживаются в редакторе кода и в процессе отладки. Намного сложнее выявлять логические ошибки при написании алгоритма решения той или иной задачи.

Microsoft Visual Studio позволяет контролировать значения переменных в процессе выполнения программы с использованием инструментов для отладки: **точек останова и специальных окон** для проверки значений переменных во время выполнения программы. Одним из удобных инструментов Microsoft Visual Studio является использование точек останова – намеренного прерывания исполнения программы. В режиме отладки программа будет исполняться до такой точки, после чего ее выполнение будет приостановлено и возобновлено нажатием клавиши F5. Чтобы поставить точку останова, следует щелкнуть левой кнопкой мыши по серой области слева от кода программы (рис. 1).

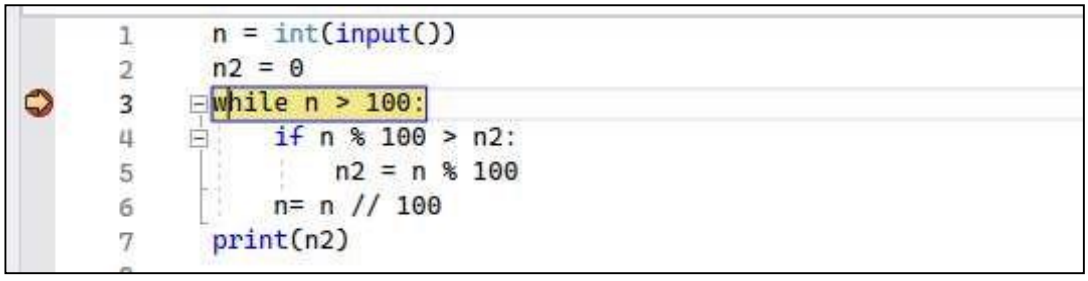

Рис. 1. Использование точек останова

В режиме отладки на строку, код которой в данный момент выполняется, указывает желтая стрелка (рис. 2).

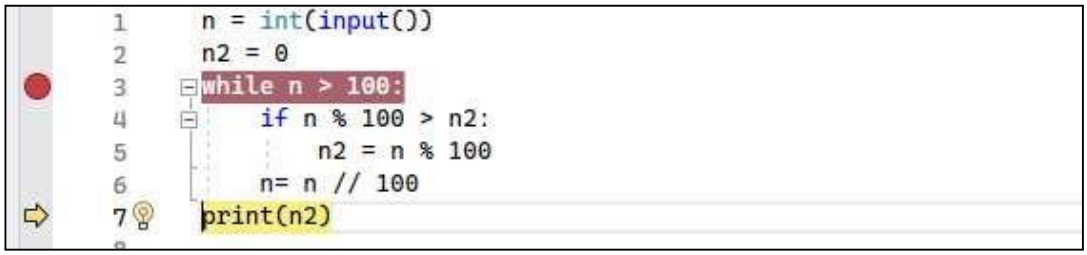

Рис. 2. Пауза выполнения программы

В режиме останова можно отображать значения переменных или свойств, просто удерживая мышь над названием переменной в коде программы (рис. 3).

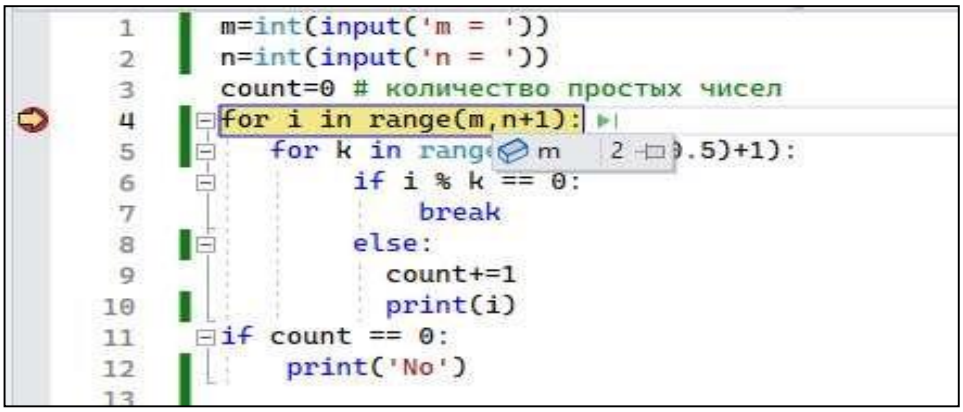

Рис. 3. Возврат текущих значений переменных

Таким образом, мы имеем возможность контролировать значения переменных на каждом шаге выполнения программы, а также следить за путем выполнения программы в случае наличия ветвлений и циклов.

Чтобы выполнить следующий оператор программы, щелкните **F8** или на кнопке **Шаг с заходом**, расположенной на панели инструментов **Отладка**. Затем в меню **Отладка \ Окна** выбрать **Локальные**.

Окно «**Локальные**» показывает состояние переменных и свойств, которые используются на протяжении всего выполнения программы (рис. 4).

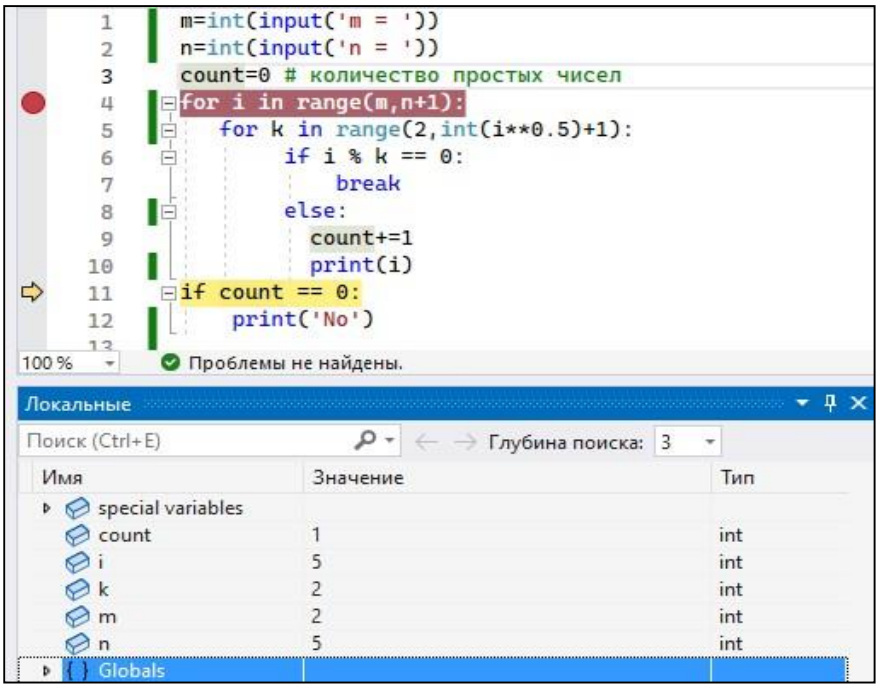

Рис. 4. Состояние переменных в окне «Локальные»

Окно **«Контрольные значения»** позволяет видеть изменение значений не только переменных, но и любых выражений и свойств объектов в процедуре, в которой установлены точки останова. Данные выражения нужно просто выделить в окне **Код** и перетащить в окно **Контрольные значения**.

Таким образом, использование **точек останова** и **окон отладки** позволяет эффективно обнаруживать **логические ошибки** в программах без использования отладочных печатей.

Оборудование и материалы: для выполнения данной лабораторной работы необходим компьютер с установленной операционной системой Windows 7 или выше, системой программирования Microsoft Visual Studio, а также программой MS Word для подготовки отчета.

Указания по технике безопасности: к выполнению лабораторных работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Порядок выполнения работы:

1. Составьте не менее 10 тестов с использованием методов стратегии черного ящика для тестирования программы, заданной вариантом задания.
2. Напишите программу, заданную вариантом задания.
3. Заполните таблицу с результатами тестирования (по образцу таблицы 1 в примере выполнения задания).
4. Используя средства отладки Visual Studio (точки останова и окно отладки «Контрольные значения»), выполните пошаговое выполнение программы на одном из удачных тестов (дающих ошибочный ответ). Поместите в окно «Контрольные

значения» значения всех используемых в программе переменных и логических выражений.

5. Заполните таблицу промежуточных значений используемых переменных и логических выражений в процессе пошагового выполнения программы (по образцу таблицы 2 в примере выполнения задания).
6. Найдите и исправьте в программе все ошибки.
7. Повторно заполните таблицу с результатами тестирования.
8. Оформите отчет по лабораторной работе.
9. Подготовьтесь к ответам на контрольные вопросы.

Способы соединения элементов информационных систем. Расчет показателей надежности информационных систем при параллельном соединении элементов системы.

Лабораторная работа № 6. Автоматизация тестирования. Модульное тестирование на Python

Цель работы: изучить технологии модульного тестирования. Получить практические навыки по работе с модулем Python unittest.

Краткие теоретические сведения

Расширение возможностей программного обеспечения и как следствие усложнение его архитектуры привело к тому, что возросли также сложность и объём работы по его тестированию. В результате этого возникла необходимость автоматизации процесса тестирования, чтобы программисту или тестировщику при каждой итерации не приходилось в очередной раз выполнять одни и те же действия по проверке правильности работы программы.

В качестве одного из вариантов решения данной задачи можно рассматривать *модульное* или *Unit тестирование*.

Идея модульного тестирования состоит в том, что параллельно основному компоненту программы, который включает непосредственно алгоритмы её работы, создаётся дополнительный «тестовый», в котором имитируется работа основного компонента в тех или иных условиях. По результатам выполнения «тестового» компонента судят о правильности работы основного.

При этом «тестовый» компонент можно запускать на выполнение, не компилируя программу целиком, в том числе в автоматическом режиме. Это обеспечивает достаточно высокую степень автоматизации процесса тестирования и значительно сокращает время на его выполнение.

Важно отметить, что задача автоматизации тестирования в принципе не может быть решена полностью. В частности невозможно автоматизировать исследовательское тестирование. Однако автоматизировать рутинные операции, например, интеграционное и регрессионное тестирование можно вполне. Последнее особенно важно, так как при создании новой версии программного обеспечения значительный объём работ по тестированию состоит именно в том, чтобы убедиться, что новый функционал не привёл к ошибкам в работе уже существующего.

Что собой представляет модульный (Unit) тест? Как уже говорилось выше, модульный тест - это вспомогательный компонент программы, предназначенный для имитации её работы в целях тестирования. По сути, это не что иное, как *программный сценарий,*

который вызывает те или иные функции тестируемой программы и анализирует результаты их работы.
В настоящее время для создания подобных сценариев нет необходимости разрабатывать какие-либо сложные технические решения. Существует масса готовых фреймворков, которые не только облегчают разработку тестов, но и берут на себя значительную часть работы по анализу и представлению их результатов.
Подобные фреймворки часто входят в состав интегрированных сред разработки (IDE). Например, Visual Studio имеет собственный фреймворк **Unit Testing Framework** для модульных тестов.
Для тестирования модулей на языке Python можно использовать стандартную библиотеку **unittest.**
Пусть тестируемый модуль содержит несколько функций, и для их тестирования составлен набор тестов. Для автоматизации прогона этих тестов и составления отчета по результатам тестирования нужно создать специальный класс, и в этом классе для каждого теста создать тестовый метод, который будет сравнивать ожидаемый результат функции с фактическим.
Тест считается не пройденным (или проваленным, т.е. в работе программы присутствует ошибка), если в ходе выполнения тестового метода ожидаемый результат не совпал с фактическим, или возникло исключение.

Простейший пример модульного (Unit) тестирования

В качестве примера для наглядности возьмём простейший модуль calc.py, содержащий всего две функции (рис. 1).

```
# файл calc.py

def add(x, y):
    return x + y

def is_positive(x):
    return x > 0
```

Рисунок 1 –Тестируемый модуль

Требуется написать тесты для этого файла (модуля), который проверит правильность разработанных функций. Для этого создадим файл **tests.py**, в котором будет класс с методами для тестирования. Важно, чтобы все методы с тестами начинались с *test_<что-то>,* иначе Python не поймет, что тестировать.
После наследования от класса *TestCase* из *unittest* будут доступны методы, которые проверяют на соответствие ожидаемому результату.
Напишем несколько проверок для функций вышеприведённого модуля (рис. 2).

```python
import unittest
import calc # тестируемый модуль

class TestCalc(unittest.TestCase):
    # начинается с test_
    def test_add(self):
        self.assertEqual(calc.add(3, 6), 9)

    # начинается с test_
    def test_is_positive(self):
        self.assertTrue(calc.is_positive(1))

if __name__ == "__main__":
    unittest.main()
```

Рисунок 2 – Класс с методами для тестирования

Здесь два теста: один проверяет на равенство, другой на истину.

Кроме того, имеются и другие виды проверок, которые показаны на рисунке 3. Так, например, для сравнения чисел с плавающей точкой (float) рекомендуется использовать assertAlomstEqual.

Method	Checks that
assertEqual(a, b)	a == b
assertNotEqual(a, b)	a != b
assertTrue(x)	bool(x) is True
assertFalse(x)	bool(x) is False
assertIs(a, b)	a is b
assertIsNot(a, b)	a is not b
assertIsNone(x)	x is None
assertIsNotNone(x)	x is not None
assertIn(a, b)	a in b
assertNotIn(a, b)	a not in b
assertIsInstance(a, b)	isinstance(a, b)
assertNotIsInstance(a, b)	not isinstance(a, b)

Рисунок 3 - Список методов для проверки на ожидаемое соответствие

После запуска файла *tests.py* на экране выведется сообщение о проведении тестов и их статус. Каждый пройденный тест обозначается точкой, а проваленный - буквой F. В нашем случае получили две точки и статус OK (рис. 4).

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.006s

OK
```

Рисунок 4 – Результаты первого тестирования

Попробуем заменить некоторые значения так, что ожидаемый результат не будет сходиться с вычислениями, например, изменим тест с проверкой на положительное число (рис. 5):

```
    # начинается с test_
    def test_is_positive(self):
        self.assertTrue(calc.is_positive(-1))
```

Рисунок 5 – Пример тестового метода с ошибочным результатом

На рисунке 6 показано сообщение, которое появится после запуска файла *tests.py*. Оно означает, что мы провалили один тест и получили .F и статус Failed.

```
.F
======================================================================
FAIL: test_is_positive (__main__.TestCalc.test_is_positive)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\NATA\METOD\Тестирование_и_отладкаПО\2024\Программы\tests.py", line 11
, in test_is_positive
    self.assertTrue(calc.is_positive(-1))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 2 tests in 0.007s

FAILED (failures=1)
```

Рисунок 6 – Результаты второго тестирования

Если мы заменим в первом методе с проверкой на равенство ожидаемый результат на другое число (рис. 7), то провалим оба тести и получим FF. На рисунке 8 показано, как это выглядит.

```
import unittest
import calc # тестируемый модуль

class TestCalc(unittest.TestCase):
    # начинается с test_
    def test_add(self):
        self.assertEqual(calc.add(3, 6), 10)

    # начинается с test_
    def test_is_positive(self):
        self.assertTrue(calc.is_positive(-1))

if __name__ == "__main__":
    unittest.main()
```

Рисунок 7 – Пример тестового метода с двумя ошибочными результатами

```
FF
======================================================================
FAIL: test_add (__main__.TestCalc.test_add)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\NATA\METOD\Тестирование_и_отладкаПО\2024\Программы\tests.py", line 7,
 in test_add
    self.assertEqual(calc.add(3, 6), 10)
AssertionError: 9 != 10

======================================================================
FAIL: test_is_positive (__main__.TestCalc.test_is_positive)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\NATA\METOD\Тестирование_и_отладкаПО\2024\Программы\tests.py", line 11
, in test_is_positive
    self.assertTrue(calc.is_positive(-1))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 2 tests in 0.014s

FAILED (failures=2)
```

Рисунок 8 – Результаты третьего тестирования

Как уже говорилось в самом начале, всё это лишь самые простейшие примеры. На самом деле при помощи модульных тестов можно выполнять тестирование практически любой сложности в соответствии с решаемой задачей.

Рассмотрим достоинства и недостатки модульного (Unit) тестирования.

Достоинства

1. Модульные тесты значительно оптимизируют сам процесс тестирования. Тестовые случаи обрабатываются в автоматическом режиме.

   Тестировщику остаётся только анализировать результат.
2. Более высокий уровень контроля качества по сравнению с «обычным» ручным тестированием. При автоматизированном тестировании практически исключается возможность пропуска тех или иных тестов. Кроме того решается проблема с описанием тестовых случаев. При написании теста они естественным образом документируются в его коде.

Недостатки

Недостатки модульных тестов, по сути, являются продолжением их достоинств.

1. Модульные тесты - это программные сценарии. То есть, данные тесты сами являются программами и, как следствие, не защищены от возможных ошибок, свойственных программам.
2. Написание модульных тестов требует времени. Наиболее оптимальным считается покрытие тестами 70-80% кода (для 70-80% кода программы должны быть написаны модульные тесты). Таким образом при разработке нового программного обеспечения или

его компонентов, затраты времени на написание тестов становятся сопоставимы с самим процессом разработки. Экономия времени проявляется только при регрессионном и интеграционном тестировании. Данное обстоятельство неизбежно сказывается на сроках выполнения проекта.

3. Модульные тесты не способны полностью заменить ручное тестирование, и тем более исключить необходимость владения теорией и технологиями тестирования. Модульные тесты не в состоянии заменить такие виды тестирования как, например, исследовательское или тестирование юзабилити. Кроме того, если внимательно присмотреться даже к тем элементарным примерам, которые приведены выше, в них можно увидеть реализацию простейших функциональных тестов. Единственная особенность – эти тесты записаны в виде программного сценария, и выполняться они будут не человеком, а программой. Поэтому, также как и при ручном тестировании, для правильного написания модульных тестов владение соответствующими знаниями и навыками в области тестирования обязательны.
4. Применение модульного тестирования требует не только определённой квалификации, но и высокой культуры разработки. Модульные тесты на самом деле довольно тонкий и сложный инструмент. При неправильном использовании они могут не только оказаться бесполезными, но и навредить процессу разработки. При этом, как показывает практика, зачастую даже специальные рекомендации для подобных случаев не в силах исправить ситуацию.

   Не стоит надеяться на то, что модульные тесты сами по себе улучшат качество готового программного продукта и решат вопросы, связанные с организационными и кадровыми издержками. Это не панацея, а инструмент в помощь разработчикам, не более. Модульные тесты при правильном применении способны принести большую пользу разработчикам, которая нивелирует указанные недостатки.

Задание для лабораторной работы

Для программы, заданной в лабораторной работе 5 (в соответствии с вариантом задания), необходимо выполнить тестирование модуля (метода), решающего поставленную задачу, на всех тестах, составленных в работе 5.

Порядок выполнения работы

1. Решение задачи в соответствии с вариантом задания в работе 5 оформите в виде функции (метода) (см. пример выполнения задания). Первоначальный вариант программы должен содержать логические ошибки.
2. Создайте класс модульного тестирования, в котором для каждого теста, составленного в работе 5, должен быть свой тестовый метод.
3. Используя средства автоматизированного тестирования модуля unittest, выполните все тесты.
4. Внесите исправления в тестируемую программу так, чтобы все тесты были пройдены.

5. Оформите отчет по лабораторной работе.
6. Подготовьте ответы на контрольные вопросы.

6.1. Рекомендуемая литература

6.1.1. Основная литература

1. Сенченко, П.В. Надежность, эргономика и качество АСОИУ: учебное пособие / П.В. Сенченко; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования Томский государственный университет систем управления и радиоэлектроники. - Томск: ТУСУР, 2016. - 189 с.: [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=480960.

2. Надёжность информационных систем: лабораторный практикум / Ю.Ю. Громов, И.В. Дидрих, О.Г. Иванова и др.; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Тамбовский государственный технический университет». - Тамбов: Издательство ФГБОУ ВПО «ТГТУ», 2015. - 113 с. [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=444906

6.1.2. Дополнительная литература:

1. Антонов В.Ф. Методы и средства проектирования информационных систем: учебное пособие / В.Ф. Антонов, А.А. Москвитин; Министерство образования и науки Российской Федерации, Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Северо-Кавказский федеральный университет». - Ставрополь: СКФУ, 2016. - 342 с.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=458663

6.1.3. Методическая литература:

1. Методические рекомендации для студентов по организации самостоятельной работы по дисциплине «Тестирование и отладка программного обеспечения».

6.1.4. Интернет-ресурсы:

1. http://www.intuit.ru – сайт дистанционного образования в области информационных технологий
2. http://www.iprbookshop.ru – ЭБС «IPRbooks».
3. http://www.biblioclub.ru – университетская библиотека онлайн
4. http://window.edu.ru – система федеральных образовательных порталов. Каталоги, библиотеки, форумы, законы, документы, стандарты
5. http://www.iqlib.ru - интернет библиотека образовательных изданий, в которой собраны электронные учебники, справочные и учебные пособия.

6.1.5. Программное обеспечение

MathCad 14, Microsoft Excel.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ

Методические указания

для обучающихся по организации и проведению самостоятельной работы
по дисциплине «Тестирования и отладка программного обеспечения»
для студентов направления подготовки 09.03.02 Информационные системы и технологии
направленность (профиль) Информационные системы и технологии обработки цифрового контент

Пятигорск, 2026

СОДЕРЖАНИЕ

1. Общие положения

Самостоятельная работа - планируемая учебная, учебно-исследовательская, научно-исследовательская работа студентов, выполняемая во внеаудиторное (аудиторное) время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Самостоятельная работа студентов (СРС) в ВУЗе является важным видом учебной и научной деятельности студента. Самостоятельная работа студентов играет значительную роль в рейтинговой технологии обучения.

К основным видам самостоятельной работы студентов относятся:

− формирование и усвоение содержания конспекта лекций на базе рекомендованной лектором учебной литературы, включая информационные образовательные ресурсы (электронные учебники, электронные библиотеки и др.);

− написание докладов;

− подготовка к семинарам, практическим и лабораторным работам, их оформление;

− составление аннотированного списка статей из соответствующих журналов по отраслям знаний (педагогических, психологических, методических и др.);

− выполнение учебно-исследовательских работ, проектная деятельность;

− подготовка лабораторных разработок и рекомендаций по решению проблемной ситуации;

− выполнение домашних заданий в виде решения отдельных задач, проведения типовых расчетов, расчетно-компьютерных и индивидуальных работ по отдельным разделам содержания дисциплин и т.д.;

− компьютерный текущий самоконтроль и контроль успеваемости на базе электронных обучающих и аттестующих тестов;

− выполнение выпускной квалификационной работы и др.

Методика организации самостоятельной работы студентов зависит от структуры, характера и особенностей изучаемой дисциплины, объема часов на ее изучение, вида заданий для самостоятельной работы студентов, индивидуальных качеств студентов и условий учебной деятельности.

Процесс организации самостоятельной работы студентов включает в себя следующие этапы:

- подготовительный (определение целей, составление программы, подготовка методического обеспечения, подготовка оборудования);
- основной (реализация программы, использование приемов поиска информации, усвоения, переработки, применения, передачи знаний, фиксирование результатов, самоорганизация процесса работы);
- заключительный (оценка значимости и анализ результатов, их систематизация, оценка эффективности программы и приемов работы, выводы о направлениях оптимизации труда).

Самостоятельная работа по дисциплине «Тестирования и отладка программного обеспечения» направлена на формирование следующих **компетенций**:

Код	Формулировка:
ПК-6	Способность оценки качества разрабатываемого обеспечения, включая разработку тестов, проведение тестирования и исследование результатов
ПК-8	Способность обеспечивать требуемый качественный бесперебойный режим работы инфокоммуникационной системы
ПК-9	Способность разработки, отладки, модификации и поддержки системного программного обеспечения

2. Цель и задачи самостоятельной работы

Ведущая цель организации и осуществления СРС совпадает с целью обучения студента – формирование набора общенаучных, профессиональных и специальных компетенций будущего бакалавра по соответствующему направлению подготовки

При организации СРС важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности. Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

Задачами СРС являются:

- систематизация и закрепление полученных теоретических знаний и лабораторных умений студентов;
- углубление и расширение теоретических знаний;
- формирование умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развитие познавательных способностей и активности студентов: творческой инициативы, самостоятельности, ответственности и организованности;
- формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развитие исследовательских умений;

3. Технологическая карта самостоятельной работы студента

Коды реализуемых компетенций	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе (акад.)		
			СРС	Контактная работа с преподавателем	Всего
5 семестр					
ПК-6 ПК-8 ПК-9	Самостоятельное изучение литературы и источников	Собеседование	49,32	5,48	54,8
ПК-6 ПК-8 ПК-9	Подготовка к лабораторным занятиям	Защита ЛР	6,48	0,72	7,2
ПК-6 ПК-8 ПК-9	Написание реферата/доклада	Защита доклада	9	1	10
	Итого за 5 семестр		**64,8**	**7,2**	**72**

4. Порядок выполнения самостоятельной работы студентом

4.1. Методические рекомендации по работе с учебной литературой

При работе с книгой необходимо подобрать литературу, научиться правильно ее читать, вести записи. Для подбора литературы в библиотеке используются алфавитный и систематический каталоги.

Важно помнить, что рациональные навыки работы с книгой - это всегда большая экономия времени и сил.

Правильный подбор учебников рекомендуется преподавателем, читающим лекционный курс. Необходимая литература может быть также указана в методических разработках по данному курсу.

Изучая материал по учебнику, следует переходить к следующему вопросу только после правильного уяснения предыдущего, описывая на бумаге все выкладки и вычисления (в том числе те, которые в учебнике опущены или на лекции даны для самостоятельного вывода).

При изучении любой дисциплины большую и важную роль играет самостоятельная индивидуальная работа.

Особое внимание следует обратить на определение основных понятий курса. Студент должен подробно разбирать примеры, которые поясняют такие определения, и уметь строить аналогичные примеры самостоятельно. Нужно добиваться точного представления о том, что изучаешь. Полезно составлять опорные конспекты. При изучении материала по учебнику полезно в тетради (на специально отведенных полях) дополнять конспект лекций. Там же следует отмечать вопросы, выделенные студентом для консультации с преподавателем.

Выводы, полученные в результате изучения, рекомендуется в конспекте выделять, чтобы они при перечитывании записей лучше запоминались.

Опыт показывает, что многим студентам помогает составление листа опорных сигналов, содержащего важнейшие и наиболее часто употребляемые формулы и понятия. Такой лист помогает запомнить формулы, основные положения лекции, а также может служить постоянным справочником для студента.

Чтение научного текста является частью познавательной деятельности. Ее цель – извлечение из текста необходимой информации. От того на сколько осознанна читающим собственная внутренняя установка при обращении к печатному слову (найти нужные сведения, усвоить информацию полностью или частично, критически проанализировать материал и т.п.) во многом зависит эффективность осуществляемого действия.

Выделяют ***четыре основные установки в чтении научного текста*:**

информационно-поисковый (задача – найти, выделить искомую информацию)

усваивающая (усилия читателя направлены на то, чтобы как можно полнее осознать и запомнить, как сами сведения, излагаемые автором, так и всю логику его рассуждений)

аналитико-критическая (читатель стремится критически осмыслить материал, проанализировав его, определив свое отношение к нему)

творческая (создает у читателя готовность в том или ином виде – как отправной пункт для своих рассуждений, как образ для действия по аналогии и т.п. – использовать суждения автора, ход его мыслей, результат наблюдения, разработанную методику, дополнить их, подвергнуть новой проверке).

Основные виды систематизированной записи прочитанного:

Аннотирование – предельно краткое связное описание просмотренной или прочитанной книги (статьи), ее содержания, источников, характера и назначения;

Планирование – краткая логическая организация текста, раскрывающая содержание и структуру изучаемого материала;

Тезирование – лаконичное воспроизведение основных утверждений автора без привлечения фактического материала;

Цитирование – дословное выписывание из текста выдержек, извлечений, наиболее существенно отражающих ту или иную мысль автора;

Конспектирование – краткое и последовательное изложение содержания прочитанного.

Конспект – сложный способ изложения содержания книги или статьи в логической последовательности. Конспект аккумулирует в себе предыдущие виды записи, позволяет всесторонне охватить содержание книги, статьи. Поэтому умение составлять план, тезисы, делать выписки и другие записи определяет и технологию составления конспекта.

Методические рекомендации по составлению конспекта:

1. Внимательно прочитайте текст. Уточните в справочной литературе непонятные слова. При записи не забудьте вынести справочные данные на поля конспекта;
2. Выделите главное, составьте план;
3. Кратко сформулируйте основные положения текста, отметьте аргументацию автора;
4. Законспектируйте материал, четко следуя пунктам плана. При конспектировании старайтесь выразить мысль своими словами. Записи следует вести четко, ясно.
5. Грамотно записывайте цитаты. Цитируя, учитывайте лаконичность, значимость мысли.

В тексте конспекта желательно приводить не только тезисные положения, но и их доказательства. При оформлении конспекта необходимо стремиться к емкости каждого предложения. Мысли автора книги следует излагать кратко, заботясь о стиле и выразительности написанного. Число дополнительных элементов конспекта должно быть логически обоснованным, записи должны распределяться в определенной последовательности, отвечающей логической структуре произведения. Для уточнения и дополнения необходимо оставлять поля.

Овладение навыками конспектирования требует от студента целеустремленности, повседневной самостоятельной работы.

4.2. Методические рекомендации по подготовке к практическим и лабораторным занятиям

Для того чтобы лабораторные и практические занятия приносили максимальную пользу, необходимо помнить, что упражнение и решение задач проводятся по вычитанному на лекциях материалу и связаны, как правило, с детальным разбором отдельных вопросов лекционного курса. Следует подчеркнуть, что только после усвоения лекционного материала с определенной точки зрения (а именно с той, с которой он излагается на лекциях) он будет закрепляться на лабораторных занятиях как в результате обсуждения и анализа лекционного материала, так и с помощью решения проблемных ситуаций, задач. При этих условиях студент не только хорошо усвоит материал, но и научится применять его на практике, а также получит дополнительный стимул (и это очень важно) для активной проработки лекции.

При самостоятельном решении задач нужно обосновывать каждый этап решения, исходя из теоретических положений курса. Если студент видит несколько путей решения проблемы (задачи), то нужно сравнить их и выбрать самый рациональный. Полезно до начала вычислений составить краткий план решения проблемы (задачи). Решение проблемных задач или примеров следует излагать подробно, вычисления располагать в строгом порядке, отделяя вспомогательные вычисления от основных. Решения при необходимости нужно сопровождать комментариями, схемами, чертежами и рисунками.

Следует помнить, что решение каждой учебной задачи должно доводиться до окончательного логического ответа, которого требует условие, и по возможности с выводом. Полученный ответ следует проверить способами, вытекающими из существа данной задачи. Полезно также (если возможно) решать несколькими способами и сравнить полученные результаты. Решение задач данного типа нужно продолжать до приобретения твердых навыков в их решении.

4.3. Методические рекомендации по самопроверке знаний

После изучения определенной темы по записям в конспекте и учебнику, а также решения достаточного количества соответствующих задач на лабораторных занятиях и самостоятельно студенту рекомендуется, провести самопроверку усвоенных знаний, ответив на контрольные вопросы по изученной теме.

В случае необходимости нужно еще раз внимательно разобраться в материале.

Иногда недостаточность усвоения того или иного вопроса выясняется только при изучении дальнейшего материала. В этом случае надо вернуться назад и повторить плохо усвоенный материал. Важный критерий усвоения теоретического материала - умение решать задачи или пройти тестирование по пройденному материалу. Однако следует помнить, что правильное решение задачи может получиться в результате применения механически заученных формул без понимания сущности теоретических положений.

4.4. Методические рекомендации по написанию научных текстов (докладов, докладовов, эссе, научных статей и т.д.)

Перед тем, как приступить к написанию научного текста, важно разобраться, какова истинная цель вашего научного текста - это поможет вам разумно распределить свои силы и время.

Во-первых, сначала нужно определиться с идеей научного текста, а для этого необходимо научиться либо относиться к разным явлениям и фактам несколько критически (своя идея – как иная точка зрения), либо научиться увлекаться какими-то известными идеями, которые нуждаются в доработке (идея – как оптимистическая позиция и направленность на дальнейшее совершенствование уже известного). Во-вторых, научиться организовывать свое время, ведь, как известно, свободное (от всяких глупостей) время –

важнейшее условие настоящего творчества, для него наконец-то появляется время. Иногда именно на организацию такого времени уходит немалая часть сил и талантов.

Писать следует ясно и понятно, стараясь основные положения формулировать четко и недвусмысленно (чтобы и самому понятно было), а также стремясь структурировать свой текст. Каждый раз надо представлять, что ваш текст будет кто-то читать и ему захочется сориентироваться в нем, быстро находить ответы на интересующие вопросы (заодно представьте себя на месте такого человека). Понятно, что работа, написанная «сплошным текстом» (без заголовков, без выделения крупным шрифтом наиболее важным мест и т, п.), у культурного читателя должна вызывать брезгливость и даже жалость к автору (исключения составляют некоторые древние тексты, когда и жанр был иной и к текстам относились иначе, да и самих текстов было гораздо меньше – не то, что в эпоху «информационного взрыва» и соответствующего «информационного мусора»).

Объем текста и различные оформительские требования во многом зависят от принятых в конкретном учебном заведении порядков.

Доклад - это самостоятельное исследование студентом определенной проблемы, комплекса взаимосвязанных вопросов.

Доклад не должна составляться из фрагментов статей, монографий, пособий. Кроме простого изложения фактов и цитат, в доклад е должно проявляться авторское видение проблемы и ее решения.

Рассмотрим основные этапы подготовки

а студентом.

Выполнение доклада начинается с выбора темы.

Затем студент приходит на первую консультацию к руководителю, которая предусматривает:

- обсуждение цели и задач работы, основных моментов избранной темы;
- консультирование по вопросам подбора литературы;
- составление предварительного плана.

Следующим этапом является работа с литературой. Необходимая литература подбирается студентом самостоятельно.

После подбора литературы целесообразно сделать рабочий вариант плана работы. В нем нужно выделить основные вопросы темы и параграфы, раскрывающие их содержание.

Составленный список литературы и предварительный вариант плана уточняются, согласуются на очередной консультации с руководителем.

Затем начинается следующий этап работы - изучение литературы. Только внимательно читая и конспектируя литературу, можно разобраться в основных вопросах темы и подготовиться к самостоятельному (авторскому) изложению содержания доклада. Конспектируя первоисточники, необходимо отразить основную идею автора и его позицию по исследуемому вопросу, выявить проблемы и наметить задачи для дальнейшего изучения данных проблем.

Систематизация и анализ изученной литературы по проблеме исследования позволяют студенту написать работу.

Рабочий вариант текста доклада предоставляется руководителю на проверку. На основе рабочего варианта текста руководитель вместе со студентом обсуждает возможности доработки текста, его оформление. После доработки доклад сдается на кафедру для его оценивания руководителем.

Требования к написанию доклада

Написание 1 доклада является обязательным условием выполнения плана СРС по любой дисциплине профессионального цикла.

Тема доклада может быть выбрана студентом из предложенных в рабочей программе или фонде оценочных средств дисциплины, либо определена самостоятельно, исходя из интересов студента (в рамках изучаемой дисциплины). Выбранную тему необходимо согласоваться с преподавателем.

Доклад должен быть написан научным языком.

Объем доклада должен составлять 20-25 стр.

Структура доклада:

● Введение (не более 3-4 страниц). Во введении необходимо обосновать выбор темы, ее актуальность, очертить область исследования, объект исследования, основные цели и задачи исследования.

● Основная часть состоит из 2-3 разделов. В них раскрывается суть исследуемой проблемы, проводится обзор мировой литературы и источников Интернет по предмету исследования, в котором дается характеристика степени разработанности проблемы и авторская аналитическая оценка основных теоретических подходов к ее решению. Изложение материала не должно ограничиваться лишь описательным подходом к раскрытию выбранной темы. Оно также должно содержать собственное видение рассматриваемой проблемы и изложение собственной точки зрения на возможные пути ее решения.

● Заключение (1-2 страницы). В заключении кратко излагаются достигнутые при изучении проблемы цели, перспективы развития исследуемого вопроса

● Список использованной литературы (не меньше 10 источников), в алфавитном порядке, оформленный в соответствии с принятыми правилами. В список использованной литературы рекомендуется включать работы отечественных и зарубежных авторов, в том числе статьи, опубликованные в научных журналах в течение последних 3-х лет и ссылки на ресурсы сети Интернет.

● Приложение (при необходимости).

Требования к оформлению:

● текст с одной стороны листа;
● шрифт Times New Roman;
● кегль шрифта 14;
● межстрочное расстояние 1,5;
● поля: сверху 2,5 см, снизу – 2,5 см, слева - 3 см, справа 1,5 см;
● доклад должен быть представлен в сброшюрованном виде.

Порядок защиты доклада:

Защита доклада проводится на лабораторных занятиях, после окончания работы студента над ним и исправления всех недочетов, выявленных преподавателем в ходе консультаций. На защиту доклада отводится 5-7 минут времени, в ходе которого студент должен показать свободное владение материалом по заявленной теме. При защите доклада приветствуется использование мультимедиа-презентации.

Оценка доклада

Доклад оценивается по следующим критериям:

• соблюдение требований к его оформлению;

• необходимость и достаточность для раскрытия темы приведенной в тексте доклада информации;

• умение студента свободно излагать основные идеи, отраженные в докладе;

• способность студента понять суть задаваемых преподавателем и сокурсниками вопросов и сформулировать точные ответы на них.

Критерии оценки:

Оценка «отлично» выставляется студенту, если в докладе студент исчерпывающе, последовательно, четко и логически стройно излагает материал; свободно справляется с задачами, вопросами и другими видами применения знаний; использует для написания доклада современные научные материалы; анализирует полученную информацию; проявляет самостоятельность при написании доклада.

Оценка «хорошо» выставляется студенту, если качество выполнения доклада достаточно высокое. Студент твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопросы по теме доклада.

Оценка «удовлетворительно» выставляется студенту, если материал доклада излагается частично, но пробелы не носят существенного характера, студент допускает неточности и ошибки при защите доклада, дает недостаточно правильные формулировки, наблюдаются нарушения логической последовательности в изложении материала.

Оценка «неудовлетворительно» выставляется студенту, если он не подготовил доклад или допустил существенные ошибки. Студент неуверенно излагает материал доклада, не отвечает на вопросы преподавателя.

Описание шкалы оценивания

Максимально возможный балл за весь текущий контроль устанавливается равным 55. Текущее контрольное мероприятие считается сданным, если студент получил за него не менее 60% от установленного для этого контроля максимального балла. Рейтинговый балл, выставляемый студенту за текущее контрольное мероприятие, сданное студентом в установленные графиком контрольных мероприятий сроки, определяется следующим образом:

Уровень выполнения контрольного задания	Рейтинговый балл (в % от максимального балла за контрольное задание)
Отличный	100
Хороший	80
Удовлетворительный	60
Неудовлетворительный	0

4.5. Методические рекомендации по выполнению исследовательских проектов

Исследовательская проектная работа – это групповая работа, для выполнения которой необходим выбор и приложение научной методики к поставленной задаче, получение собственного теоретического или экспериментального материала, на основании которого необходимо провести анализ и сделать выводы об исследуемом явлении. Выполнение проекта – это всегда коллективная, творческая Лабораторная работа, предназначенная для получения определенного продукта или научно-технического результата. Такая работа подразумевает четкое, однозначное формирование поставленной задачи, определение сроков выполнения намеченного, определение требований к разрабатываемому объекту.

Выполнение 1 группового проекта является обязательным условием выполнения самостоятельной работы по любой дисциплине профессионального цикла. Тема проектного задания может быть выбрана студентом из предложенных в рабочей программе или фонде оценочных средств дисциплины, либо определена самостоятельно, исходя из интересов студента (в рамках изучаемой дисциплины). Выбранную тему необходимо согласоваться с преподавателем.

Требования по выполнению и оформлению проекта

При выполнении проекта приветствуется работа в группе (2-3 человека). Проект – это исследовательская работа, в ходе которой студенты должны продемонстрировать владение навыками научного исследования, умения проводить анализ, обобщать информацию, делать выводы, предлагать свои решения проблемы, рассматриваемой в проекте.

При подготовке материалов проекта студенты должны продемонстрировать владение современными методами компьютерной обработки данных.

Критерии оценки работы участника проекта.

Для каждого из участников проекта оцениваются:

• профессиональные теоретические знания в соответствующей области;

• умение работать со справочной и научной литературой, осуществлять поиск необходимой информации в Интернет;

• умение работать с техническими средствами;

• умение пользоваться соответствующими выполняемому проекту информационными технологиями;

• умение готовить материалы проекта для презентации: составлять и редактировать тексты, формировать презентацию проекта;

• умение работать в команде;

• умение публично представлять результаты собственной деятельности;

• коммуникабельность, инициативность, творческие способности.

Критерии выставления оценки участникам проекта

Оценка	Профессиональные компетенции	Компетенции, связанные с использованием соответствующих выполняемому проекту технических средств и информационных технологий	Иные универсальные компетенции (коммуникабельность, инициативность, умение работать в «команде», управленческие навыки и т.д.)	Отчетность
«Отлично»	Работа выполнена на высоком профессиональном уровне. Представленный материал в основном фактически верен, допускаются негрубые фактические неточности. Студент свободно отвечает на вопросы, связанные с проектом.	Технические средства и информационные технологии освоены и использованы для реализации проекта полностью	Студент проявил инициативу, творческий подход, способность к выполнению сложных заданий, навыки работы в коллективе, организационные способности.	Проект представлен полностью и в срок.
«Хорошо»	Работа выполнена на достаточно высоком профессиональном уровне. Допущено до 4–5 фактических ошибок. Студент отвечает на вопросы, связанные с проектом, но недостаточно полно.	Обнаруживаются некоторые ошибки в использовании соответствующих технических средств и информационных технологий	Студент достаточно полно, но без инициативы и творческих находок выполнил возложенные на него задачи.	Проект представлен достаточно полно и в срок, но с некоторыми недоработками.
«Удовлетворительно»	Уровень недостаточно высок. Допущено до 8 фактических ошибок. Студент может ответить лишь на некоторые из заданных вопросов,	Обнаруживает недостаточное владение навыками работы с техническими средствами и соответствующими и	Студент выполнил большую часть возложенной на него работы.	Проект сдан со значительным опозданием (более недели) и не полностью

Оценка	Профессиональные компетенции	Компетенции, связанные с использованием соответствующих выполняемому проекту технических средств и информационных технологий	Иные универсальные компетенции (коммуникабельность, инициативность, умение работать в «команде», управленческие навыки и т.д.)	Отчетность
	связанных с проектом.	информационным и технологиями		
«Неудовлетворительно»	Работа не выполнена или выполнена на низком уровне. Допущено более 8 фактических ошибок. Ответы на связанные с проектом вопросы обнаруживают непонимание предмета и отсутствие ориентации в материале проекта.	Навыков работы с техническими средствами нет, информационные технологии не освоены	Студент практически не работал, не выполнил свои задачи или выполнил лишь отдельные не существенные поручения в групповом проекте.	Проект не сдан.

Студенты должны: защитить проект в режиме презентации, предъявить файлы выполненного проекта, уметь рассказать о технологиях, использованных ими при выполнении проекта, дать оценку работы каждого члена группы (*если проект групповой*).

Максимально возможный балл за весь текущий контроль устанавливается равным **55.** Текущее контрольное мероприятие считается сданным, если студент получил за него не менее 60% от установленного для этого контроля максимального балла. Рейтинговый балл, выставляемый студенту за текущее контрольное мероприятие, сданное студентом в установленные графиком контрольных мероприятий сроки, определяется следующим образом:

Уровень выполнения контрольного задания	Рейтинговый балл (в % от максимального балла за контрольное задание)
Отличный	100
Хороший	80
Удовлетворительный	60
Неудовлетворительный	0

4.6. Методические рекомендации по подготовке к экзаменам и зачетам

Изучение многих общепрофессиональных и специальных дисциплин завершается экзаменом. Подготовка к экзамену способствует закреплению, углублению и обобщению знаний, получаемых, в процессе обучения, а также применению их к решению лабораторных задач. Готовясь к экзамену, студент ликвидирует имеющиеся пробелы в

знаниях, углубляет, систематизирует и упорядочивает свои знания. На экзамене студент демонстрирует то, что он приобрел в процессе обучения по конкретной учебной дисциплине.

Экзаменационная сессия - это серия экзаменов, установленных учебным планом. Между экзаменами интервал 3-4 дня. Не следует думать, что 3-4 дня достаточно для успешной подготовки к экзаменам.

В эти 3-4 дня нужно систематизировать уже имеющиеся знания. На консультации перед экзаменом студентов познакомят с основными требованиями, ответят на возникшие у них вопросы. Поэтому посещение консультаций обязательно.

Требования к организации подготовки к экзаменам те же, что и при занятиях в течение семестра, но соблюдаться они должны более строго. Во-первых, очень важно соблюдение режима дня; сон не менее 8 часов в сутки, занятия заканчиваются не позднее, чем за 2-3 часа до сна. Оптимальное время занятий - утренние и дневные часы. В перерывах между занятиями рекомендуются прогулки на свежем воздухе, неутомительные занятия спортом. Во-вторых, наличие хороших собственных конспектов лекций. Даже в том случае, если была пропущена какая-либо лекция, необходимо во время ее восстановить (переписать ее на кафедре), обдумать, снять возникшие вопросы для того, чтобы запоминание материала было осознанным. В-третьих, при подготовке к экзаменам у студента должен быть хороший учебник или конспект литературы, прочитанной по указанию преподавателя в течение семестра. Здесь можно эффективно использовать листы опорных сигналов.

Вначале следует просмотреть весь материал по сдаваемой дисциплине, отметить для себя трудные вопросы. Обязательно в них разобраться. В заключение еще раз целесообразно повторить основные положения, используя при этом листы опорных сигналов.

Систематическая подготовка к занятиям в течение семестра позволит использовать время экзаменационной сессии для систематизации знаний.

Контроль самостоятельной работы студентов

Контроль самостоятельной работы проводится преподавателем в аудитории.

Предусмотрены следующие виды контроля: собеседование, оценка доклада, оценка презентации, оценка участия в круглом столе, оценка выполнения проекта.

Подробные критерии оценивания компетенций приведены в Фонде оценочных средств для проведения текущей и промежуточной аттестации.

Список литературы для выполнения СРС

Основная литература:

1. Сенченко, П.В. Надежность, эргономика и качество АСОИУ: учебное пособие / П.В. Сенченко; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования Томский государственный университет систем управления и радиоэлектроники. - Томск: ТУСУР, 2016. - 189 с.: [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=480960.

2. Надёжность информационных систем: лабораторный практикум / Ю.Ю. Громов, И.В. Дидрих, О.Г. Иванова и др.; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Тамбовский государственный технический университет». - Тамбов: Издательство ФГБОУ ВПО «ТГТУ», 2015. - 113 с. [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=444906

Дополнительная литература:

1. Антонов В.Ф. Методы и средства проектирования информационных систем: учебное пособие / В.Ф. Антонов, А.А. Москвитин; Министерство образования и науки Российской Федерации, Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Северо-Кавказский федеральный университет». - Ставрополь: СКФУ, 2016. - 342 с.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=458663

Методическая литература:

1. Методические рекомендации для самостоятельной работы студентов по дисциплине «Тестирования и отладка программного обеспечения»
2. Методические указания к лабораторным работам по дисциплине «Тестирования и отладка программного обеспечения»

Интернет-ресурсы:

http://www.intuit.ru – сайт дистанционного образования в области информационных технологий

http://www.iprbookshop.ru – ЭБС «IPRbooks».

http://www.biblioclub.ru – университетская библиотека онлайн

http://window.edu.ru – система федеральных образовательных порталов. Каталоги, библиотеки, форумы, законы, документы, стандарты

http://www.iqlib.ru - интернет библиотека образовательных изданий, в которой собраны электронные учебники, справочные и учебные пособия.