МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ «МЕТОДЫ ТЕСТИРОВАНИЯ»

Направление подготовки	**09.04.02 Информационные системы и технологии**
Направленность (профиль)	**«Технологии работы с данными и знаниями, анализ информации»**
Квалификация выпускника	Магистр

Пятигорск, 2024

СОДЕРЖАНИЕ

1. Цель и задачи освоения дисциплины (модуля)

Целью освоение дисциплины «Методы тестирования» является получение устойчивых навыков самостоятельного программирования с применением современных программных средств для создания, редактирования, отладки и тестирования программ.

В соответствии с указанной целью при изучении дисциплины ставятся следующие задачи:

- привить навыки работы в среде визуального программирования,
- дать сведения об основных приемах отладки и тестирования программ;
- изучить основные методы тестирования программ.

2. Место дисциплины в структуре образовательной программы

Дисциплина «Методы тестирования» входит в вариативную часть дисциплин по выбору блока Б1.

3. Компетенции обучающегося, формируемые в результате изучения дисциплины

5.1. Наименование компетенции

Индекс	Формулировка:
ПК-3	способность выполнять администрирование систем управления базами данных, системного программного обеспечения инфокоммуникационной системы организации, управление развитием инфокоммуникационной системы организации
ПК-6	способность проводить организационное сопровождение разработки, отладки, модификации и поддержки информационных технологий и систем

В результате освоения дисциплины обучающийся должен:

ЗНАТЬ	- принципы тестирования; - основные критерии тестирования; - стратегию тестирования и критерии завершения тестирования; - основные этапы тестирования.
УМЕТЬ	- разрабатывать набор тестов для программы как основываясь на ее спецификации так и ее тексте; - оценивать качество набора тестов; - при необходимости создавать систему автоматизации этапов тестирования.
ВЛАДЕТЬ	- методами тестирования и отладки программных средств; - средствами автоматизации тестирования.

4. Структура и содержание дисциплины (модуля)

Лабораторная работа 1

Выбор и анализ критериев тестирования программы

Цель и содержание: научиться выполнять анализ функциональных возможностей разработанной программы и определять критерии тестирования для проверки функциональных возможностей программы.

Организационная форма занятий: практикум.

Вопросы для обсуждения на лабораторном занятии:

Анализ функциональных возможностей разработанной программы.

Определение критериев тестирования для проверки функциональных возможностей программы.

Аппаратура и материалы. Для выполнения задания необходим персональный компьютер, а также соответствующее лабораторной работе программное обеспечение.

Указания по технике безопасности. Самостоятельно не производить: установку и удаление программного обеспечения, ремонт персонального компьютера. Соблюдать правила технической безопасности при работе с электрооборудованием.

Теоретическое обоснование

Требования к идеальному критерию тестирования

- Критерий должен быть достаточным, т.е. показывать, когда некоторое конечное множество тестов достаточно для тестирования данной программы.

- Критерий должен быть полным, т.е. в случае ошибки должен существовать тест из множества тестов, удовлетворяющих критерию, который раскрывает ошибку.

- Критерий должен быть надежным, т.е. любые два множества тестов, удовлетворяющих ему, одновременно должны раскрывать или не раскрывать ошибки программы

- Критерий должен быть легко проверяемым, например вычисляемым на тестах

Для нетривиальных классов программ в общем случае не существует полного и надежного критерия, зависящего от программ или спецификаций.

Поэтому мы стремимся к идеальному общему критерию через реальные частные.

Классы критериев

Структурные критерии используют информацию о структуре программы (критерии так называемого "белого ящика")

Функциональные критерии формулируются в описании требований к программному изделию (критерии так называемого "черного ящика")

Критерии стохастического тестирования формулируются в терминах проверки наличия заданных свойств у тестируемого приложения, средствами проверки некоторой статистической гипотезы.

Мутационные критерии ориентированы на проверку свойств программного изделия на основе подхода Монте-Карло.

Структурные критерии (класс I).

Структурные критерии используют модель программы в виде "белого ящика", что предполагает знание исходного текста программы или спецификации программы в виде потокового графа управления. Структурная информация понятна и доступна разработчикам подсистем и модулей приложения, поэтому данный класс критериев часто используется на этапах модульного и интеграционного тестирования (Unit testing, Integration testing).

Структурные критерии базируются на основных элементах УГП, операторах, ветвях и путях.

Условие критерия тестирования команд (критерий C0) - набор тестов в совокупности должен обеспечить прохождение каждой команды не менее одного раза. Это слабый критерий, он, как правило, используется в больших программных системах, где другие критерии применить невозможно.

Условие критерия тестирования ветвей (критерий C1) - набор тестов в совокупности должен обеспечить прохождение каждой ветви не менее одного раза. Это

достаточно сильный и при этом экономичный критерий, поскольку множество ветвей в тестируемом приложении конечно и не так уж велико. Данный критерий часто используется в системах автоматизации тестирования.

Условие критерия тестирования путей (критерий С2) - набор тестов в совокупности должен обеспечить прохождение каждого пути не менее 1 раза. Если программа содержит цикл (в особенности с неявно заданным числом итераций), то число итераций ограничивается константой (часто - 2, или числом классов выходных путей).

Приведен пример простой программы. Рассмотрите условия ее тестирования в соответствии со структурными критериями.

```
1   public void Method (ref int x)
    {
2       if (x>17)
3           x = 17-x;
4       if (x==-13)
5           x = 0;
6   }
```

Пример простой программы, для тестирования по структурным критериям

```
1   void Method (int *x)
    {
2       if (*x>17)
3           *x = 17-*x;
4       if (*x==-13)
5           *x = 0;
6   }
```

Пример простой программы, для тестирования по структурным критериям

Тестовый набор из одного теста, удовлетворяет критерию команд (С0):

(X,Y)={(xвх=30, xвых=0)} покрывает все операторы трассы 1-2-3-4-5-6

Тестовый набор из двух тестов, удовлетворяет критерию ветвей (С1):

(X,Y)={(30,0), (17,17)} добавляет 1 тест к множеству тестов для С0 и трассу 1-2-4-6. Трасса 1-2-3-4-5-6 проходит через все ветви достижимые в операторах if при условии true, а трасса 1-2-4-6 через все ветви, достижимые в операторах if при условии false.

Тестовый набор из четырех тестов, удовлетворяет критерию путей (С2):

(X,Y)={(30,0), (17,17), (-13,0), (21,-4)}

Набор условий для двух операторов if с метками 2 и 4 приведен в таблица 1

Таблица 1. Условия операторов if

	(30,0)	(17,17)	(-13,0)	(21,-4)
2 if (x>17)	>	$\leq$	$\leq$	>
4 if (x==-13)	$\neq$	$\neq$	=	$\neq$

Критерий путей С2 проверяет программу более тщательно, чем критерии - С1, однако даже если он удовлетворен, нет оснований утверждать, что программа реализована в соответствии со спецификацией.

Например, если спецификация задает условие, что |x|<=100, невыполнимость которого можно подтвердить на тесте (-177,-177). Действительно, операторы 3 и 4 на тесте (-177,-177) не изменят величину x=-177 и результат не будет соответствовать спецификации.

Структурные критерии не проверяют соответствие спецификации, если оно не отражено в структуре программы. Поэтому при успешном тестировании программы по критерию С2 мы можем не заметить ошибку, связанную с невыполнением некоторых условий спецификации требований.

Содержание отчета и его форма
- Подготовьте отчет, в котором полностью опишите выполнение заданий.
- Отчет по лабораторной работе должен содержать:
- Название работы;
- Цель лабораторной работы;
- Формулировку задания и технологию его выполнения;
- Ответы на контрольные вопросы.

Контрольные вопросы:

1. Этапы разработки программного обеспечения и этапы жизненного цикла программы.
2. Временные диаграммы данных этапов.
3. Обоснование необходимости рассмотрения методов тестирования программ.
4. Определения теста, тестирования, удачного теста.
5. Экономика тестирования.

Защита лабораторной работы

По результатам отчета, представленного в письменной форме, проводится собеседование, которое имеет контролирующую и учебную функции.

Лабораторная работа 2

Модульное тестирование разработанного приложения

Цель и содержание: создавать приложения, состоящие из нескольких модулей и уметь тестировать каждый модуль.

Организационная форма занятий: практикум.

Вопросы для обсуждения на лабораторном занятии:

Разработка приложения, состоящего из нескольких модулей.

Тестирование каждого модуля.

Аппаратура и материалы. Для выполнения задания необходим персональный компьютер, а также соответствующее лабораторной работе программное обеспечение.

Указания по технике безопасности. Самостоятельно не производить: установку и удаление программного обеспечения, ремонт персонального компьютера. Соблюдать правила технической безопасности при работе с электрооборудованием.

Теоретическое обоснование

Каждая сложная программная система состоит из отдельных частей – модулей, выполняющих ту или иную функцию в составе системы. Для того, чтобы удостовериться в корректной работе системы в целом, необходимо вначале протестировать каждый модуль системы в отдельности. В случае возникновения проблем это позволит проще выявить модули, вызвавшие проблему, и устранить соответствующие дефекты в них. Такое тестирование модулей по отдельности получило называние модульного тестирования (unit testing).

Для каждого модуля, подвергаемого тестированию, разрабатывается тестовое окружение, включающее в себя драйвер и заглушки, готовятся тест-требования и тест-планы, описывающие конкретные тестовые примеры.

Основная цель модульного тестирования – удостовериться в соответствии требованиям каждого отдельного модуля системы перед тем, как будет произведена его интеграция в состав системы.

При этом в ходе модульного тестирования решаются четыре основные задачи.

Поиск и документирование несоответствий требованиям – это классическая задача тестирования, включающая в себя не только разработку тестового окружения и тестовых примеров, но и выполнение тестов, протоколирование результатов выполнения, составление отчетов о проблемах.

Поддержка разработки и рефакторинга низкоуровневой архитектуры системы и межмодульного взаимодействия – эта задача больше свойственна "легким" методологиям типа XP, где применяется принцип тестирования перед разработкой (Test-driven development), при котором основным источником требований для программного модуля является тест, написанный до самого модуля. Однако, даже при классической схеме тестирования модульные тесты могут выявить проблемы в дизайне системы и нелогичные или запутанные механизмы работы с модулем.

Поддержка рефакторинга модулей – эта задача связана с поддержкой процесса изменения системы. Достаточно часто в ходе разработки требуется проводить рефакторинг модулей или их групп – оптимизацию или полную переделку программного кода с целью повышения его сопровождаемости, скорости работы или надежности. Модульные тесты при этом являются мощным инструментом для проверки того, что новый вариант программного кода работает в точности так же, как и старый.

Поддержка устранения дефектов и отладки — эта задача сопряжена с обратной связью, которую получают разработчики от тестировщиков в виде отчетов о проблемах. Подробные отчеты о проблемах, составленные на этапе модульного тестирования, позволяют локализовать и устранить многие дефекты в программной системе на ранних стадиях ее разработки или разработки ее новой функциональности.

В силу того, что модули, подвергаемые тестированию, обычно невелики по размеру, модульное тестирование считается наиболее простым (хотя и достаточно трудоемким) этапом тестирования системы. Однако, несмотря на внешнюю простоту, с модульным тестированием связано две проблемы.

Не существует единых принципов определения того, что в точности является отдельным модулем.

Различия в трактовке самого понятия модульного тестирования – понимается ли под ним обособленное тестирование модуля, работа которого поддерживается только тестовым окружением, или речь идет о проверке корректности работы модуля в составе уже разработанной системы. В последнее время термин "модульное тестирование" чаще используется во втором смысле, хотя в этом случае речь скорее идет об интеграционном тестировании.

Традиционное определение модуля с точки зрения его тестирования: "модуль – это компонент минимального размера, который может быть независимо протестирован в ходе верификации программной системы". В реальности часто возникают проблемы с тем, что считать модулем. Существует несколько подходов к данному вопросу:

модуль – это часть программного кода, выполняющая одну функцию с точки зрения функциональных требований;

модуль – это программный модуль, т.е. минимальный компилируемый элемент программной системы;

модуль – это задача в списке задач проекта (с точки зрения его менеджера);

модуль – это участок кода, который может уместиться на одном экране или одном листе бумаги;

модуль – это один класс или их множество с единым интерфейсом.

Обычно за тестируемый модуль принимается либо программный модуль (единица компиляции) в случае, если система разрабатывается на процедурном языке программирования, либо класс, если система разрабатывается на объектно-ориентированном языке.

В случае систем, написанных на процедурных языках, процесс тестирования модуля происходит так - для каждого модуля разрабатывается тестовый драйвер,

вызывающий функции модуля и собирающий результаты их работы, и набор заглушек, которые имитируют поведение функций, содержащихся в других модулях и не попадающих под тестирование данного модуля. При тестировании объектно-ориентированных систем существует ряд особенностей, прежде всего вызванных инкапсуляцией данных и методов в классах.

В случае объектно-ориентированных систем более мелкое деление классов и использование отдельных методов в качестве тестируемых модулей нецелесообразно в связи с тем, что для тестирования каждого метода потребуется разработка тестового окружения, сравнимого по сложности с уже написанным программным кодом класса. Кроме того, декомпозиция класса нарушает принцип инкапсуляции, согласно которому объекты каждого класса должны вести себя как единое целое с точки зрения других объектов.

Процесс тестирования классов как модулей иногда называют компонентным тестированием. В ходе такого тестирование проверяется взаимодействие методов внутри класса и правильность доступа методов к внутренним данным класса. При таком тестировании возможно обнаружение не только стандартных дефектов, связанных с выходами за границы диапазона или неверно реализованными требованиями, а также обнаружение специфических дефектов объектно-ориентированного программного обеспечения:

дефектов инкапсуляции, в результате которых, например, сокрытые данные класса оказывается недоступными при помощи соответствующих публичных методов;

дефектов наследования, при наличии которых схема наследования блокирует важные данные или методы от классов-потомков;

дефектов полиморфизма, при которых полиморфное поведение класса оказывается распространенным не на все возможные классы;

дефектов инстанцирования, при которых во вновь создаваемых объектах класса не устанавливаются корректные значения по умолчанию параметров и внутренних данных класса.

Однако, выбор класса в качестве тестируемого модуля имеет и ряд сопряженных проблем.

Определение степени полноты тестирования класса. В том случае, если в качестве тестируемого модуля выбран класс, не совсем ясно, как определять степень полноты его тестирования. С одной стороны, можно использовать классический критерий полноты покрытия программного кода тестами: если полностью выполнены все структурные элементы всех методов, как публичных, так и скрытых, — то тесты можно считать полными.

Однако существует альтернативный подход к тестированию класса, согласно которому все публичные методы должны предоставлять пользователю данного класса согласованную схему работы и достаточно проверить типичные корректные и некорректные сценарии работы с данным классом. Т.е., например, в классе, объекты которого представляют записи в телефонной книжке, одним из типичных сценариев работы будет "Создать запись $\longrightarrow$ искать запись и найти ее $\longrightarrow$ удалить запись $\longrightarrow$ искать запись вторично и получить сообщение об ошибке".

Различия в этих двух методах напоминают различия между тестированием "черного" и "белого" ящиков, но на самом деле второй подход отличается от "черного ящика" тем, что функциональные требования к системе могут быть составлены на уровне более высоком, чем отдельные классы, и установление адекватности тестовых сценариев требованиям остается на откуп тестировщику.

Протоколирование состояний объектов и их изменений. Некоторые методы класса предназначены не для выдачи информации пользователю, а для изменения внутренних данных объекта класса. Значение внутренних данных объекта определяет его состояние в каждый отдельный момент времени, а вызов методов, изменяющих данные, изменяет и

состояние объекта. При тестировании классов необходимо проверять, что класс адекватно реагирует на внешние вызовы в любом из состояний. Однако, зачастую из-за инкапсуляции данных невозможно определить внутреннее состояние класса программными способами внутри драйвера.

В этом случае может помочь составление схемы поведения объекта как конечного автомата с определенным набором состояний. Такая схема может входить в низкоуровневую проектную документацию (например, в составе описания архитектуры системы), а может составляться тестировщиком или разработчиком на основе функциональных требований к системе. В последнем случае для определения всех возможных состояний может потребоваться ручной анализ программного кода и определение его соответствия требованиям. Автоматизированное тестирование в этом случае может лишь определить, по всем ли выявленным состояниям осуществлялись переходы и все ли возможные реакции проверялись.

Тестирование изменений. Как уже упоминалось выше, модульные тесты – мощный инструмент проверки корректности изменений, внесенных в исходный код при рефакторинге. Однако, в результате рефакторинга только одного класса, как правило, не меняется его внешний интерфейс с другими классами (интерфейсы меняются при рефакторинге сразу нескольких классов). В результате обычных эволюционных изменений системы у класса может меняться внешний интерфейс, причем как по формальным (изменяются имена и состав методов, их параметры), так и по функциональным признакам (при сохранении внешнего интерфейса меняется логика работы методов). Для проведения модульного тестирования класса после таких изменений потребуется изменение драйвера и, возможно, заглушек. Но только модульного тестирования в данном случае недостаточно, необходимо также проводить и интеграционное тестирование данного класса вместе со всеми классами, которые связаны с ним по данным или по управлению.

Вне зависимости от того, на какие модули, подвергаемые тестированию, разбивается система, рекомендуется изложить принципы выделения тестируемых модулей в плане и стратегии тестирования, а также составить на базе структурной схемы архитектуры системы новую структурную схему, на которой отметить все тестируемые модули. Это позволит спрогнозировать состав и сложность драйверов и заглушек, требуемых для модульного тестирования системы. Такая схема также может использоваться позже на этапе модульного тестирования для выделения укрупненных групп модулей, подвергаемых интеграции.

Вне зависимости от того, какая минимальная единица исходных кодов системы выбирается за минимальный тестируемый модуль, существует еще одно различие в подходах к модульному тестированию.

Первый подход к модульному тестированию основывается на предположении, что функциональность каждого вновь разработанного модуля должна проверяться в автономном режиме без его интеграции с системой. Здесь для каждого вновь разрабатываемого модуля создается тестовый драйвер и заглушки, при помощи которых выполняется набор тестов. Только после устранения всех дефектов в автономном режиме производится интеграция модуля в систему и проводится тестирование на следующем уровне. Достоинством данного подхода является более простая локализация ошибок в модуле, поскольку при автономном тестировании исключается влияние остальных частей системы, которое может вызывать маскировку дефектов (эффект четного числа ошибок). Основной недостаток данного метода – повышенная трудоемкость написания драйверов и заглушек, поскольку заглушки должны адекватно моделировать поведение системы в различных ситуациях, а драйвер должен не только создавать тестовое окружение, но и имитировать внутреннее состояние системы, в составе которой должен функционировать модуль.

Второй подход построен на предположении, что модуль все равно работает в составе системы и если модули интегрировать в систему по одному, то можно протестировать поведение модуля в составе всей системы. Этот подход свойственен большинству современных "облегченных" методологий разработки, в том числе и XP.

В результате применения такого подхода резко сокращаются трудозатраты на разработку заглушек и драйверов – в роли заглушек выступает уже оттестированная часть системы, а драйвер выполняет только функции передачи и приема данных, не моделируя внутреннее состояние системы.

Тем не менее, при использовании данного метода возрастает сложность написания тестовых примеров – для приведения в нужное состояние системы заглушек, как правило, требуется только установить значения тестовых переменных, а для приведения в нужное состояние части реальной системы необходимо выполнить целый сценарий. Каждый тестовый пример в этом случае должен содержать такой сценарий.

Кроме того, при этом подходе не всегда удается локализовать ошибки, скрытые внутри модуля, которые могут проявиться при интеграции следующих модулей.

Создать тестовое окружение и запустить метод. Проверяем операцию сложения на примере 2+2, т.е. в стеке до начала выполнения самой операции (т.е. после компиляции) находятся следующие элементы: " 2 ", " 2 ", " + ".

```
private void buttonStart_Click(object sender, EventArgs e)
      {
         // создаем провайдер для генерирования и компиляции кода на C#
         System.CodeDom.Compiler.CodeDomProvider prov =
System.CodeDom.Compiler.CodeDomProvider.CreateProvider("CSharp");
         // создаем параметры компилирования
         System.CodeDom.Compiler.CompilerParameters cmpparam = new
System.CodeDom.Compiler.CompilerParameters();
         // результат компиляции - библиотека
         cmpparam.GenerateExecutable = false;
         // не включаем информацию отладчика
         cmpparam.IncludeDebugInformation = false;
         // подключаем 2-е стандартные библиотеки и библиотеку
CalcClass.dll
         cmpparam.ReferencedAssemblies.Add(Application.StartupPath +
"\\CalcClass.dll");
         cmpparam.ReferencedAssemblies.Add("System.dll");
         cmpparam.ReferencedAssemblies.Add("System.Windows.Forms.dll");
         // имя выходной сборки - My.dll
         cmpparam.OutputAssembly = "My.dll";
         // компилируем класс AnalaizerClass с заданными параметрами
         System.CodeDom.Compiler.CompilerResults res =
prov.CompileAssemblyFromFile(cmpparam, Application.StartupPath + "\\
AnalaizerClass.cs");
         // Выводим результат компилирования на экран
         if (res.Errors.Count != 0)
         {
            richTextBox1.Text += res.Errors[0].ToString();
         }
         else
         {
            // загружаем только что скомпилированную сборку(здесь тонкий
момент - если мы прото загрузим сборку из файла, то он будет заблокирован,
```

```
                // acces denied, поэтому вначале читаем его в поток и лишь
        потом подключаем)
                System.IO.BinaryReader reader = new
        System.IO.BinaryReader(new System.IO.FileStream(Application.StartupPath + "\\
My.dll", System.IO.FileMode.Open, System.IO.FileAccess.Read));
                Byte[] asmBytes = new Byte[reader.BaseStream.Length];
                reader.Read(asmBytes, 0, (Int32) reader.BaseStream.Length);
                reader.Close();
                reader = null;
                System.Reflection.Assembly assm =
        System.Reflection.Assembly.Load(asmBytes);
                Type[] types = assm.GetTypes();
                Type analaizer = types[0];
                // находим метод CheckCurrency - к счастью, он единственный
                System.Reflection.MethodInfo addinfo =
        analaizer.GetMethod("RunEstimate");
                System.Reflection.FieldInfo fieldopz =
        analaizer.GetField("opz");
                System.Collections.ArrayList ar = new
        System.Collections.ArrayList();
                ar.Add("2");
                ar.Add("2");
                ar.Add("+");
                fieldopz.SetValue(null, ar);
                richTextBox1.Text += addinfo.Invoke(null, null).ToString();
                asmBytes = null;
            }
            prov.Dispose();
        }
```

Замечание. На самом деле данный подход позволяет выявить множество недостатков программы, которые другими тестами не выявляются. Можно попробовать поэкспериментировать с "Калькулятором" и убедиться, что он работает корректно. Однако, если в тестируемый метод подать на вход не " 2 ", " 2 ", " + ", а " 2 ", " 2 ", " + ", " + ", то программа закончит работу с исключением. Это говорит о том, что метод RunEstimate написан не корректно. Можно, например, было бы скрыть, т. е. сделать доступ private, всем методам AnalaizerClass, кроме Estimate (это было бы более правильно, но для простоты тестирования они сделаны public. Стоит отметить, что Visual Studio 2005 имеет также механизмы для тестирования подобных методов.). Тем самым мы не позволим другим выполнять "потенциально опасные" методы и передавать им некорректные значения. Однако это не является достаточным механизмом защиты программы. Необходимо провести более качественную валидацию используемых методами параметров.

Замечание. К проблеме создания тестового окружения можно подойти с двух сторон – либо откомпилировать код, с заранее подключенными dll файлами к проекту, либо воспользоваться областью CodeDom и компилировать в процессе выполнения. Это особенно удобно, если нужно менять тестовое окружение в процессе работы.

Составить тест-план и провести модульное тестирование следующих методов:

1. /// <summary>
   /// Проверка корректности скобочной структуры входного выражения
   /// </summary>
   /// <returns>true - если все нормально,

false - если есть ошибка</returns>
/// метод бежит по входному выражению, символ за
символом анализируя его и считая количество скобок.
В случае возникновения
/// ошибки возвращает false, а в erposition записывает позицию,
на которой возникла ошибка.
public static bool CheckCurrency()

2. /// <summary>
/// Форматирует входное выражение, выставляя между
операторами пробелы и удаляя лишние, а также отлавливает
неопознанные операторы, следит за концом строки
/// а также отлавливает ошибки на конце строки
/// </summary>
/// <returns>конечную строку или сообщение об ошибке,
начинающиеся со спец. символа &</returns>
public static string Format()

3. /// <summary>
/// Создает массив, в котором располагаются операторы и
символы, представленные в обратной польской записи (безскобочной)
/// На этом же этапе отлавливаются почти все остальные
ошибки (см код). По сути - это компиляция.
/// </summary>
/// <returns>массив обратной польской записи</returns>
public static System.Collections.ArrayList CreateStack()

4. /// <summary>
/// Вычисление обратной польской записи
/// </summary>
/// <returns>результат вычислений или сообщение об ошибке</returns>
public static string RunEstimate()

Содержание отчета и его форма

- Подготовьте отчет, в котором полностью опишите выполнение заданий.
- Отчет по лабораторной работе должен содержать:
- Название работы;
- Цель лабораторной работы;
- Формулировку задания и технологию его выполнения;
- Ответы на контрольные вопросы.

Контрольные вопросы:

1. Методология "черного" и "белого" ящика.
2. Невозможность построения полного теста в каждой из стратегий.
3. Принципы тестирования.
4. Критерии "черного" ящика.
5. Эквивалентное разбиение, граничные значения, функциональные диаграммы и предположение об ошибке.

Защита лабораторной работы

По результатам отчета, представленного в письменной форме, проводится собеседование, которое имеет контролирующую и учебную функции.

Лабораторная работа 3

Интеграционное тестирование разработанного приложения

Цель и содержание: выполнять сборку модулей в единое приложение, применять восходящее и нисходящее тестирование разработанного приложения.

Организационная форма занятий: практикум.

Вопросы для обсуждения на лабораторном занятии:

Сборка модулей в единое приложение.

Восходящее и нисходящее тестирование разработанного приложения.

Аппаратура и материалы. Для выполнения задания необходим персональный компьютер, а также соответствующее лабораторной работе программное обеспечение.

Указания по технике безопасности. Самостоятельно не производить: установку и удаление программного обеспечения, ремонт персонального компьютера. Соблюдать правила технической безопасности при работе с электрооборудованием.

Теоретическое обоснование

Как правило, интеграционное тестирование проводится уже по завершении модульного тестирования для всех интегрируемых модулей. Однако это далеко не всегда так. Существует несколько методов проведения интеграционного тестирования:

восходящее тестирование;

монолитное тестирование;

нисходящее тестирование.

Все эти методики основываются на знаниях об архитектуре системы, которая часто изображается в виде структурных диаграмм или диаграмм вызовов функций. Каждый узел на такой диаграмме представляет собой программный модуль, а стрелки между ними представляют собой зависимость по вызовам между модулями. Основное различие методик интеграционного тестирования заключается в направлении движения по этим диаграммам и в широте охвата за одну итерацию.

Восходящее тестирование. При использовании этого метода подразумевается, что сначала тестируются все программные модули, входящие в состав системы и только затем они объединяются для интеграционного тестирования. При таком подходе значительно упрощается локализация ошибок: если модули протестированы по отдельности, то ошибка при их совместной работе есть проблема их интерфейса. При таком подходе область поиска проблем у тестировщика достаточно узка, и поэтому гораздо выше вероятность правильно идентифицировать дефект.

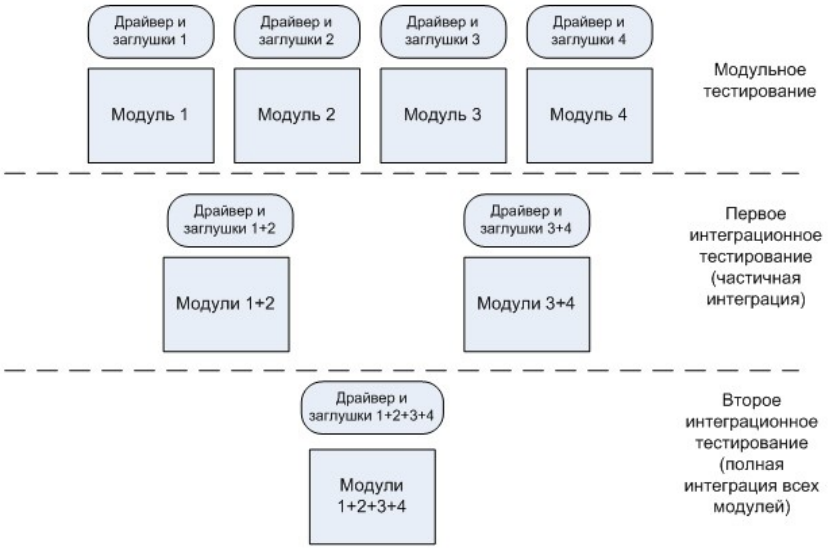

Рис. 1. Разработка драйверов и заглушек при восходящем интеграционном тестировании

Однако, у восходящего метода тестирования есть существенный недостаток - необходимость в разработке драйвера и заглушек для модульного тестирования перед проведением интеграционного тестирования и необходимость в разработке драйвера и заглушек при интеграционном тестировании части модулей системы (Рис.1)

С одной стороны драйверы и заглушки - мощный инструмент тестирования, с другой - их разработка требует значительных ресурсов, особенно при изменении состава интегрируемых модулей, иначе говоря, может потребоваться один набор драйверов для модульного тестирования каждого модуля, отдельный драйвер и заглушки для тестирования интеграции двух модулей из набора, отдельный - для тестирования интеграции трех модулей и т.п. В первую очередь это связано с тем, что при интеграции модулей отпадает необходимость в некоторых заглушках, а также требуется изменение драйвера, которое поддерживает новые тесты, затрагивающие несколько модулей.

Монолитное тестирование предполагает, что отдельные компоненты системы серьезного тестирования не проходили. Основное преимущество данного метода - отсутствие необходимости в разработке тестового окружения, драйверов и заглушек. После разработки всех модулей выполняется их интеграция, затем система проверяется вся в целом. Этот подход не следует путать с системным тестированием, которому посвящена следующая лекция. Несмотря на то, что при монолитном тестировании проверятся работа всей системы в целом, основная задача этого тестирования - определить проблемы взаимодействия отдельных модулей системы. Задачей же системного тестирования является оценка качественных и количественных характеристик системы с точки зрения их приемлемости для конечного пользователя.

Монолитное тестирование имеет ряд серьезных недостатков.

Очень трудно выявить источник ошибки (идентифицировать ошибочный фрагмент кода). В большинстве модулей следует предполагать наличие ошибки. Проблема сводится к определению того, какая из ошибок во всех вовлечённых модулях привела к полученному результату. При этом возможно наложение эффектов ошибок. Кроме того, ошибка в одном модуле может блокировать тестирование другого.

Трудно организовать исправление ошибок. В результате тестирования тестировщиком фиксируется найденная проблема. Дефект в системе, вызвавший эту проблему, будет устранять разработчик. Поскольку, как правило, тестируемые модули написаны разными людьми, возникает проблема - кто из них является ответственным за поиск устранение дефекта? При такой "коллективной безответственности" скорость устранения дефектов может резко упасть.

Процесс тестирования плохо автоматизируется. Преимущество (нет дополнительного программного обеспечения, сопровождающего процесс тестирования) оборачивается недостатком. Каждое внесённое изменение требует повторения всех тестов.

Нисходящее тестирование предполагает, что процесс интеграционного тестирования движется следом за разработкой. Сначала тестируют только самый верхний управляющий уровень системы, без модулей более низкого уровня. Затем постепенно с более высокоуровневыми модулями интегрируются более низкоуровневые. В результате применения такого метода отпадает необходимость в драйверах (роль драйвера выполняет более высокоуровневый модуль системы), однако сохраняется нужда в заглушках (Рис.2).

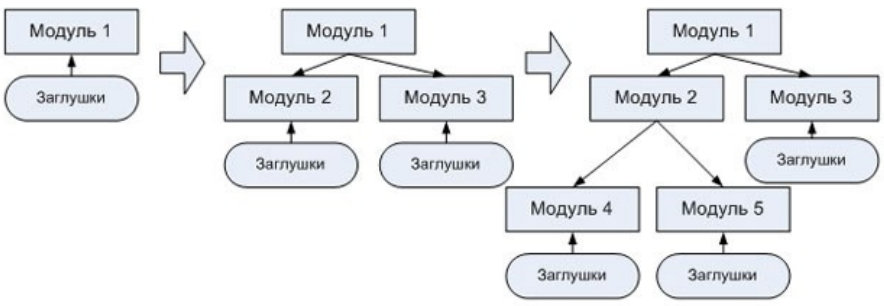

Рис. 2. Постепенная интеграция модулей при нисходящем методе тестирования

У разных специалистов в области тестирования разные мнения по поводу того, какой из методов более удобен при реальном тестировании программных систем. Йордан доказывает, что нисходящее тестирование наиболее приемлемо в реальных ситуациях, а Майерс полагает, что каждый из подходов имеет свои достоинства и недостатки, но в целом восходящий метод лучше.

В литературе часто упоминается метод интеграционного тестирования объектно-ориентированных программных систем, который основан на выделении кластеров классов, имеющих вместе некоторую замкнутую и законченную функциональность. По своей сути такой подход не является новым типом интеграционного тестирования, просто меняется минимальный элемент, получаемый в результате интеграции. При интеграции модулей на процедурных языках программирования можно интегрировать любое количество модулей при условии разработки заглушек. При интеграции классов в кластеры существует достаточно нестрогое ограничение на законченность функциональности кластера. Однако, даже в случае объектно-ориентированных систем возможно интегрировать любое количество классов при помощи классов-заглушек.

Для интеграционного тестирования наиболее существенным является рассмотрение модели программы, построенной с использованием диаграмм потоков управления. Контролируются также связи через данные, подготавливаемые и используемые другими группами программ при взаимодействии с тестируемой группой. Каждая переменная межмодульного интерфейса проверяется на тождественность описаний во взаимодействующих модулях, а также на соответствие исходным программным спецификациям. Состав и структура информационных связей реализованной группы модулей проверяются на соответствие спецификации требований этой группы. Все реализованные связи должны быть установлены, упорядочены и обобщены.

При сборке модулей в единый программный комплекс появляется два варианта построения графовой модели проекта:

Плоская или иерархическая модель проекта.

Граф вызовов.

Если программа P состоит из p модулей, то при интеграции модулей в комплекс фактически получается громоздкая плоская или более простая - иерархическая - модель программного проекта. В качестве критерия тестирования на интеграционном уровне обычно используется критерий покрытия ветвей C1. Введем также следующие обозначения:

n - число узлов в графе;

e - число дуг в графе;

q - число бинарных выборов из условий ветвления в графе;

kin - число входов в граф;

kout - число выходов из графов;

kext - число точек входа, которые могут быть вызваны извне.

Тогда сложность интеграционного тестирования всей программы P по критерию C1 может быть выражена формулой [17] :

$$V(P,C1) = \Sigma V(Mod_i, C1) - k_{in} + k_{ext} =$$
$$e - n - k_{ext} + k_{out} =$$
$$q + k_{ext}, (\forall Mod_i \in P)$$

Однако при подобном подходе к построению ГМП разработчик тестового набора неизбежно сталкивается с неприемлемо высокой сложностью тестирования V(P,C) для проектов среднего и большого объема (размером в 105 - 107 строк), что следует из роста

топологической сложности управляющего графа по МакКейбу. Таким образом, используя плоскую или иерархическую модель, трудно дать оценку тестированности TV(P,C,T) для всего проекта и оценку зависимости тестированности проекта от тестированности отдельного модуля TV(Modi,C), включенного в этот проект.

Рассмотрим вторую модель сборки модулей в процедурном программировании - граф вызовов. В этой модели в случае интеграционного тестирования учитываются только вызовы модулей в программе. Поэтому из множества M(Modi,C) тестируемых элементов можно исключить те элементы, которые не подвержены влиянию интеграции, т. е. узлы и дуги, не соединенные с вызовами модулей: $M(Mod_i, C') = E' \cup N_{in}$, где $E' = \{(n_i, n_j) \in E | n_i$ или nj содержит вызовы модулей$\}$, т.е. E' - подмножество ребер графа модуля, а Nin - "входные" узлы графа. Эта модификация ГМП приводит к получению нового графа - графа вызовов, каждый узел в этом графе представляет модуль (процедуру), а каждая дуга - вызов модуля (процедуры). Для процедурного программирования подобный шаг упрощает графовую модель программного проекта до приемлемого уровня сложности. Таким образом, может быть определена цикломатическая сложность упрощенного графа модуля Modi как V'(Modi,C'), а громоздкая формула, выражающая сложность интеграционного тестирования программного проекта, принимает следующий вид [19] :

$$V'(P, C1') = \Sigma V'(Mod_i, C1') - k_{in} + k_{ext}$$

Так, для программы, для получения графа вызовов из иерархической модели проекта должны быть исключены все дуги, кроме:

Дуги 1-2, содержащей входной узел 1 графа G.

Дуг 2-8, 8-7, 7-10, содержащих вызов модуля G1.

Дуг 2-9, 9-7, 7-10, содержащих вызов модуля G2.

В результате граф вызовов примет вид, показанный на Рис. 3, а сложность данного графа по критерию C1' равна:

V'(G,C1') = q + Kext =1+1=2.

V'(Modi,C') также называется в литературе сложностью модульного дизайна (complexity of module design).

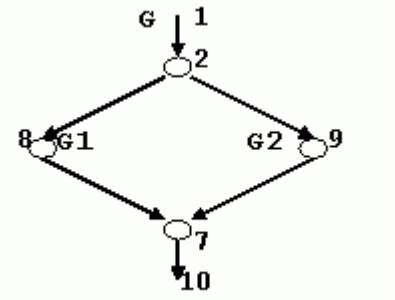

Рис. 3. Граф вызовов модулей

Сумма сложностей модульного дизайна для всех модулей по критерию C1 или сумма их аналогов для других критериев тестирования, исключая значения модулей самого нижнего уровня, дает сложность интеграционного тестирования для процедурного программирования

Содержание отчета и его форма

- Подготовьте отчет, в котором полностью опишите выполнение заданий.
- Отчет по лабораторной работе должен содержать:
- Название работы;
- Цель лабораторной работы;

- Формулировку задания и технологию его выполнения;
- Ответы на контрольные вопросы.

Контрольные вопросы:

1. Критерии "белого" ящика.
2. Критерии потока управления: покрытие операторов, покрытие решений или C1, покрытие условий, покрытие решений и условий, комбинаторное покрытие условий.
3. Критерии потока данных: покрытие всех определений (all-defs), all-p-uses, all-c-uses/some-p-uses, all-p-uses/some-c-uses, покрытие всех def-use цепочек (all-uses), all-du-paths, покрытие контекстов используемых переменных (U-context), all_uses2 (включает C1).
4. Определение включения одного критерия в другой.
5. Схема строгого включения для потоковых критериев, ее обоснование.

Защита лабораторной работы

По результатам отчета, представленного в письменной форме, проводится собеседование, которое имеет контролирующую и учебную функции.

Лабораторная работа 4

Составление тестового набора

Цель и содержание: выполнять разработку функции и составление тестового набора для автоматизации тестирования.

Организационная форма занятий: практикум.

Вопросы для обсуждения на лабораторном занятии:

Разработка функции и составление тестового набора для автоматизации тестирования

Аппаратура и материалы. Для выполнения задания необходим персональный компьютер, а также соответствующее лабораторной работе программное обеспечение.

Указания по технике безопасности. Самостоятельно не производить: установку и удаление программного обеспечения, ремонт персонального компьютера. Соблюдать правила технической безопасности при работе с электрооборудованием.

Теоретическое обоснование

Функциональный критерий - важнейший для программной индустрии критерий тестирования. Он обеспечивает, прежде всего, контроль степени выполнения требований заказчика в программном продукте. Поскольку требования формулируются к продукту в целом, они отражают взаимодействие тестируемого приложения с окружением. При функциональном тестировании преимущественно используется модель "черного ящика". Проблема функционального тестирования - это, прежде всего, трудоемкость; дело в том, что документы, фиксирующие требования к программному изделию (Software requirement specification, Functional specification и т.п.), как правило, достаточно объемны, тем не менее, соответствующая проверка должна быть всеобъемлющей.

Ниже приведены частные виды функциональных критериев.

Тестирование пунктов спецификации - набор тестов в совокупности должен обеспечить проверку каждого тестируемого пункта не менее одного раза.

Спецификация требований может содержать сотни и тысячи пунктов требований к программному продукту и каждое из этих требований при тестировании должно быть проверено в соответствии с критерием не менее чем одним тестом

Тестирование классов входных данных - набор тестов в совокупности должен обеспечить проверку представителя каждого класса входных данных не менее одного раза.

При создании тестов классы входных данных сопоставляются с режимами использования тестируемого компонента или подсистемы приложения, что заметно сокращает варианты перебора, учитываемые при разработке тестовых наборов. Следует заметить, что перебирая в соответствии с критерием величины входных переменных (например, различные файлы - источники входных данных), мы вынуждены применять мощные тестовые наборы. Действительно, наряду с ограничениями на величины входных данных, существуют ограничения на величины входных данных во всевозможных комбинациях, в том числе проверка реакций системы на появление ошибок в значениях или структурах входных данных. Учет этого многообразия - процесс трудоемкий, что создает сложности для применения критерия

Тестирование правил - набор тестов в совокупности должен обеспечить проверку каждого правила, если входные и выходные значения описываются набором правил некоторой грамматики.

Следует заметить, что грамматика должна быть достаточно простой, чтобы трудоемкость разработки соответствующего набора тестов была реальной (вписывалась в сроки и штат специалистов, выделенных для реализации фазы тестирования)

Тестирование классов выходных данных - набор тестов в совокупности должен обеспечить проверку представителя каждого выходного класса, при условии, что выходные результаты заранее расклассифицированы, причем отдельные классы результатов учитывают, в том числе, ограничения на ресурсы или на время (time out).

При создании тестов классы выходных данных сопоставляются с режимами использования тестируемого компонента или подсистемы, что заметно сокращает варианты перебора, учитываемые при разработке тестовых наборов.

Тестирование функций - набор тестов в совокупности должен обеспечить проверку каждого действия, реализуемого тестируемым модулем, не менее одного раза.

Очень популярный на практике критерий, который, однако, не обеспечивает покрытия части функциональности тестируемого компонента, связанной со структурными и поведенческими свойствами, описание которых не сосредоточено в отдельных функциях (т.е. описание рассредоточено по компоненту).

Критерий тестирования функций объединяет отчасти особенности структурных и функциональных критериев. Он базируется на модели "полупрозрачного ящика", где явно указаны не только входы и выходы тестируемого компонента, но также состав и структура используемых методов (функций, процедур) и классов.

Комбинированные критерии для программ и спецификаций - набор тестов в совокупности должен обеспечить проверку всех комбинаций непротиворечивых условий программ и спецификаций не менее одного раза.

При этом все комбинации непротиворечивых условий надо подтвердить, а условия противоречий следует обнаружить и ликвидировать.

Стиль кодирования:

1. Точно представлять логическую структуру кода. Смысл форматирования — показать логическую структуру кода. Для демонстрации логической структуры программисты обычно применяют отступы и другие неотображаемые символы.

2. Единообразно показывать логическую структуру кода. Некоторые стили форматирования состоят из правил с таким количеством исключений, что последовательно их соблюдать практически невозможно. Действительно хороший стиль подходит в большинстве случаев.

3. Улучшать читабельность. Стратегия использования отступов, соответствующая логике, но усложняющая процесс чтения кода, бесполезна. Схема форматирования,

использующая пробелы и разделители только там, где они требуются компилятору, логична, но читать такой код невозможно. Хорошая структура форматирования упрощает чтение кода.

4. Выдерживать процедуру исправления. Лучшие схемы форматирования хорошо переносят модификацию кода. Исправление одной строки не должно приводить к изменению нескольких других.

Порядок выполнения работы

1) По результатам лабораторных работ 1-4 составить тестовые наборы для функционального тестирования. При составлении тестовых наборов используйте правила аксиом тестирования. написать код программы для решения поставленной задачи на языке программирования, выбранном на этапе проектирования.

2) Отладить программу.

3) Оформить документацию к разработанному программному обеспечению.

4) Представить отчет по лабораторной работе для защиты.

Содержание отчета и его форма

- Подготовьте отчет, в котором полностью опишите выполнение заданий.
- Отчет по лабораторной работе должен содержать:
- Название работы;
- Цель лабораторной работы;
- Формулировку задания и технологию его выполнения;
- Ответы на контрольные вопросы.

Контрольные вопросы:

1. Монолитное, пошаговое тестирование.
2. Нисходящее и восходящее тестирование.
3. Методы тестирования за столом - инспекции, сквозные просмотры и обзоры программ.
4. Стратегия тестирования. Критерии завершения тестирования.
5. Основные направления автоматизации тестирования.

Защита лабораторной работы

По результатам отчета, представленного в письменной форме, проводится собеседование, которое имеет контролирующую и учебную функции.

Лабораторная работа 5

Сравнительный анализ различных методов тестирования

Цель и содержание: научиться проводить анализ методик, использованных студентами для тестирования программ.

Организационная форма занятий: практикум.

Вопросы для обсуждения на лабораторном занятии:

Проведение анализа методик, использованных студентами для тестирования программ

Аппаратура и материалы. Для выполнения задания необходим персональный компьютер, а также соответствующее лабораторной работе программное обеспечение.

Указания по технике безопасности. Самостоятельно не производить: установку и удаление программного обеспечения, ремонт персонального компьютера. Соблюдать правила технической безопасности при работе с электрооборудованием.

Теоретическое обоснование

Методы проверки (тестирования) программ можно разделить на статические и динамические. К первым относятся неформальное, контрольное и техническое рецензирование, инспекция, пошаговый разбор, аудит, а также статический анализ потока данных и управления.

Динамические техники следующие:

Тестирование методом белого ящика. Это подробное исследование внутренней логики и структуры программы. При этом необходимо знание исходного кода.

Тестирование методом черного ящика. Данная техника не требует каких-либо знаний о внутренней работе приложения. Рассматриваются только основные аспекты системы, не связанные или мало связанные с ее внутренней логической структурой.

Метод серого ящика. Сочетает в себе предыдущие два подхода. Отладка с ограниченным знанием о внутреннем функционировании приложения сочетается со знанием основных аспектов системы.

Необходимо провести анализ методик, использованных для тестирования программ предыдущих лабораторных работ. Результаты анализа показать в виде произвольной таблицы.

Содержание отчета и его форма

- Подготовьте отчет, в котором полностью опишите выполнение заданий.
- Отчет по лабораторной работе должен содержать:
- Название работы;
- Цель лабораторной работы;
- Формулировку задания и технологию его выполнения;
- Ответы на контрольные вопросы.

Контрольные вопросы:

1. Автоматизация построения тестов, символьное исполнение программ.
2. Построение минимального дугового покрытия УГ и на его основе минимального набора тестов для критерия С1.
3. Контроль качества набора тестов.
4. Системы контроля полноты набора тестов для определенных критериев. Системы Тестор-Фортран, Ритм, TGS, ОСТ (инструментация исходного кода программ, язык описания тестовых условий, генератор отчетов, комплексный критерий).
5. Инструментация объектного кода.

Защита лабораторной работы

По результатам отчета, представленного в письменной форме, проводится собеседование, которое имеет контролирующую и учебную функции.

5. Учебно-методическое и информационное обеспечение дисциплины (модуля)

7.1. Рекомендуемая литература

7.1.1. Основная литература:

1. Луиза Тамре. Введение в тестирование программного обеспечения. - М.: Издательский дом «Вильямс», 2013. - С. 368.

2. Фаронов, В.В. Delphi. Программирование на языке высокого уровня: В. В. Фаронов- СПб.: Лидер, 2010.

7.1.2. Дополнительная литература:

1. Гагарина, Л.Г. Современные проблемы информатики и вычислительной техники: учеб. пособие/ Л. Г. Гагарина, А. А. Петров- М.: ИД "ФОРУМ", 2011.

7.1.4. Интернет-ресурсы:

1. http://www.intuit.ru – сайт дистанционного образования в области информационных технологий
2. http://www.iqlib.ru - интернет библиотека образовательных изданий, в которой собраны электронные учебники, справочные и учебные пособия;
3. http://www.biblioclub.ru - электронная библиотечная система «Университетская библиотека – online»: специализируется на учебных материалах для ВУЗов по научно-гуманитарной тематике, а так же содержит материалы по точным и естественным наукам.
4. http://window.edu.ru – образовательные ресурсы ведущих вузов

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ СТУДЕНТОВ ПО ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ «МЕТОДЫ ТЕСТИРОВАНИЯ»

Направление подготовки	**09.04.02 Информационные системы и технологии**
Направленность (профиль)	**«Технологии работы с данными и знаниями, анализ информации»**
Квалификация выпускника	Магистр

Пятигорск, 2024

СОДЕРЖАНИЕ

6. ВВЕДЕНИЕ

Целью освоения дисциплины «Методы тестирования» является получение устойчивых навыков самостоятельного программирования с применением современных программных средств для создания, редактирования, отладки и тестирования программ.

В соответствии с указанной целью при изучении дисциплины ставятся следующие задачи:

- привить навыки работы в среде визуального программирования,
- дать сведения об основных приемах отладки и тестирования программ;
- изучить основные методы тестирования программ.

Место дисциплины в структуре образовательной программы

Дисциплина «Методы тестирования» входит в вариативную часть дисциплин по выбору блока Б1. Ее освоение происходит во 2 семестре.

Компетенции обучающегося, формируемые в результате изучения дисциплины

Наименование компетенции

Индекс	Формулировка:
ПК-3	способность выполнять администрирование систем управления базами данных, системного программного обеспечения инфокоммуникационной системы организации, управление развитием инфокоммуникационной системы организации
ПК-6	способность проводить организационное сопровождение разработки, отладки, модификации и поддержки информационных технологий и систем

В результате освоения дисциплины обучающийся должен:

ЗНАТЬ	- принципы тестирования; - основные критерии тестирования; - стратегию тестирования и критерии завершения тестирования; - основные этапы тестирования.
УМЕТЬ	- разрабатывать набор тестов для программы как основываясь на ее

	спецификации так и ее тексте; - оценивать качество набора тестов; - при необходимости создавать систему автоматизации этапов тестирования.
ВЛАДЕТЬ	- методами тестирования и отладки программных средств; - средствами автоматизации тестирования.

7. 1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТА

Основными видами учебной работы по достижению результатов освоения дисциплины являются лекции, лабораторные занятия, самостоятельная работа студентов (СРС).

На лекциях раскрываются основные положения и понятия курса, формируются знания в области методов тестирования программ. На лабораторных работах формируются умения и навыки, необходимые для решения задач профессиональной деятельности.

Приступая к изучению учебной дисциплины, необходимо ознакомиться с учебной программой, получить в библиотеке рекомендованные учебные пособия, а также получить у ведущего преподавателя в электронном виде конспекты лекций, методические рекомендации к лабораторным занятиям, завести отдельные тетради для конспектирования лекций, выполнения лабораторных работ.

Для изучения дисциплины предлагается список основной и дополнительной литературы. Основная литература предназначена для обязательного изучения, дополнительная – поможет более глубоко освоить отдельные вопросы, подготовить исследовательские задания.

В ходе лекционных занятий студент обязан осуществлять конспектирование учебного материала, особое внимание, обращая на категории, формулировки, раскрывающие содержание тех или иных понятий, процессов, эффектов. В рабочих конспектах желательно оставлять поля, на которых следует делать пометки из рекомендованной литературы, дополнять материал прослушанной лекции, формулировать научные выводы и практические рекомендации. Студент имеет право задавать преподавателю уточняющие вопросы с целью уяснения теоретических положений, разрешения спорных ситуаций.

Самостоятельная работа студентов над материалом учебной дисциплины является неотъемлемой частью учебного процесса и должна предполагать углубление знания учебного материала, излагаемого на аудиторных занятиях, и приобретение дополнительных знаний по отдельным вопросам самостоятельно.

Основными видами самостоятельной работы студентов по учебной дисциплине являются:

− самостоятельное изучение учебного материала по заданным темам;

− выполнение и защита контрольной работы;

− подготовка и оформление отчета по выполненному лабораторному занятию и подготовка к собеседованию по результатам работы.

Основными методами самостоятельной работы студентов являются:

1) для овладения знаниями:

− чтение текста (конспекта лекций, учебника, дополнительной литературы) и конспектирование текста;

− поиск информации в Интернет;

2) для закрепления и систематизации знаний:

− работа с конспектом лекции (обработка текста);

− повторная работа над учебным материалом (учебника, дополнительной литературы, слайдами);

− ответы на контрольные вопросы;

3) для формирования умений:

− подготовка к лабораторным занятиям;

− выполнение индивидуальных заданий лабораторных работ.

− подготовка отчетов по лабораторным занятиям;

В случае пропуска учебного занятия студент может воспользоваться содержанием различных блоков учебно-методического комплекса (лекции, лабораторные занятия, контрольные вопросы) для самоподготовки и освоения темы. Для самоконтроля необходимо использовать вопросы и задания, предлагаемые к лабораторным занятиям.

8. 2. ТЕХНОЛОГИЧЕСКАЯ КАРТА САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТА

Для студентов очной формы обучения:

Коды реализуемых компетенций, индикатора(ов)	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе		
			СРС	Контактная работа с преподавателем	Всего
2 семестр					
ПК-3 (ИД 1 $_{ПК-3}$, ИД 2 $_{ПК-3}$, ИД 3 $_{ПК-3}$) ПК-6 (ИД 1 $_{ПК-6}$, ИД 2 $_{ПК-6}$, ИД 3 $_{ПК-6}$)	Подготовка к лекциям	Коллоквиум	1,62	0,18	1,8
ПК-3 (ИД 1 $_{ПК-3}$, ИД 2 $_{ПК-3}$, ИД 3 $_{ПК-3}$) ПК-6 (ИД 1 $_{ПК-6}$, ИД 2 $_{ПК-6}$, ИД 3 $_{ПК-6}$)	Самостоятельное изучение литературы	Коллоквиум	81,72	9,08	90,8
ПК-3 (ИД 1	Подготовка и	Отчет	4,86	0,54	5,4

ПК-3 (ИД 1 ПК-3, ИД 2 ПК-3, ИД 3 ПК-3) ПК-6 (ИД 1 ПК-6, ИД 2 ПК-6, ИД 3 ПК-6)	выполнение лабораторных работ	письменный			
ПК-3 (ИД 1 ПК-3, ИД 2 ПК-3, ИД 3 ПК-3) ПК-6 (ИД 1 ПК-6, ИД 2 ПК-6, ИД 3 ПК-6)	Выполнение контрольной работы	Контрольная работа	9	1	10
Итого за 2 семестр			97,2	10,8	108

Для студентов заочной формы обучения:

Коды реализуемых компетенций, индикатора(ов)	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе		
			СРС	Контактная работа с преподавателем	Всего
2 семестр					
ПК-3 (ИД 1 ПК-3, ИД 2 ПК-3, ИД 3 ПК-3) ПК-6 (ИД 1 ПК-6, ИД 2 ПК-6, ИД 3 ПК-6)	Подготовка к лекциям	Коллоквиум	0,54	0,06	0,6
ПК-3 (ИД 1 ПК-3, ИД 2 ПК-3, ИД 3 ПК-3) ПК-6 (ИД 1 ПК-6, ИД 2 ПК-6, ИД 3 ПК-6)	Самостоятельное изучение литературы	Коллоквиум	107,64	11,96	119,6
ПК-3 (ИД 1 ПК-3, ИД 2 ПК-3, ИД 3 ПК-3) ПК-6 (ИД 1 ПК-6, ИД 2 ПК-6, ИД 3 ПК-6)	Подготовка и выполнение лабораторных работ	Отчет письменный	1,62	0,18	1,8
ПК-3 (ИД 1 ПК-3, ИД 2 ПК-3, ИД 3 ПК-3) ПК-6 (ИД 1 ПК-6, ИД 2 ПК-	Выполнение контрольной работы	Контрольная работа	9	1	10

$_{6}$, ИД 3 $_{ПК-6}$)					
Итого за 2 семестр			118,8	13,2	132

9. 3. ПАСПОРТ ФОНДА ОЦЕНОЧНЫХ СРЕДСТВ ДЛЯ ПРОВЕРКИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Фонд оценочных средств, позволяющий оценить уровень сформированности компетенций, размещен в УМК дисциплины «Методы тестирования» на кафедре информационной безопасности, систем и технологий и представлен следующими компонентами:

Код оцениваемой компетенции	Этап формирования компетенции (№ темы)	Средства и технологии оценки	Тип контроля (текущий / промежуточ-ный)	Вид контроля (устный / письменный	Наименование оценочного средства
ПК-3, ПК-6	Темы 1 - 3	Коллоквиум	текущий	устный	Вопросы для коллоквиумов
ПК-3, ПК-6	Темы 1 - 3	Контрольная работа	текущий	письменный	Комплект заданий для контрольной работы
ПК-3, ПК-6	Темы 1 – 3	Отчет (письмен-ный)	текущий	письменный	Комплект заданий для лабораторных работ

10. 4. Методические указания для обучающихся по освоению дисциплины

На первом этапе необходимо ознакомиться с рабочей программой дисциплины, в которой рассмотрено содержание тем дисциплины лекционного курса, взаимосвязь тем лекций с лабораторными занятиями, темы и виды самостоятельной работы. По каждому виду самостоятельной работы предусмотрены определённые формы отчетности.

Для успешного освоения дисциплины, необходимо выполнить следующие виды самостоятельной работы, используя рекомендуемые источникам информации:

№ п/п	Виды самостоятельной работы	Рекомендуемые источники информации (№ источника)			
		Основная	Дополнительная	Методическая	Интернет-ресурсы
1.	Подготовка к лекциям	1-2	1	1-3	1-4
2.	Самостоятельное изучение литературы	1-2	1	1-3	1-4
3.	Подготовка к лабораторным работам	1-2	1	1-3	1-4
4.	Выполнение контрольной работы	1-2	1	1-3	1-4

11.

12. 5. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

5.1. Рекомендуемая литература

5.1.1. Основная литература

1. Луиза Тамре. Введение в тестирование программного обеспечения. - М.: Издательский дом «Вильямс», 2013. - С. 368.

2. Фаронов, В.В. Delphi. Программирование на языке высокого уровня: В. В. Фаронов- СПб.: Лидер, 2010.

5.1.2. Дополнительная литература

1. Гагарина, Л.Г. Современные проблемы информатики и вычислительной техники: учеб. пособие/ Л. Г. Гагарина, А. А. Петров- М.: ИД "ФОРУМ", 2011.

5.1.4. Интернет-ресурсы

1. http://www.intuit.ru – сайт дистанционного образования в области информационных технологий
2. http://www.iqlib.ru - интернет библиотека образовательных изданий, в которой собраны электронные учебники, справочные и учебные пособия;
3. http://www.biblioclub.ru - электронная библиотечная система «Университетская библиотека – online»: специализируется на учебных материалах для ВУЗов по научно-гуманитарной тематике, а так же содержит материалы по точным и
4. http://window.edu.ru – образовательные ресурсы ведущих вузов

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КОНТРОЛЬНОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ «МЕТОДЫ ТЕСТИРОВАНИЯ»

Направление подготовки	**09.04.02 Информационные системы и технологии**
Направленность (профиль)	**«Технологии работы с данными и знаниями, анализ информации»**
Квалификация выпускника	Магистр

Пятигорск, 2024

СОДЕРЖАНИЕ

13.

14. Введение

Одним из наиболее трудоемких этапов (от 30 до 60% общей трудоемкости) создания программного продукта является его тестирование. Причем доля стоимости тестирования в общей стоимости разработки имеет тенденцию возрастать при увеличении сложности комплексов программ и повышении требований к их качеству. В связи с этим большое внимание уделяется выбору стратегии и методов тестирования, что не является тривиальной задачей. Таким образом, при подготовке к тестированию необходимо ответить на следующие вопросы:

- Какую стратегию тестирования выбрать и почему? Как ее реализовать?
- Какой из методов выбранной стратегии тестирования выбрать и почему?
- Как грамотно подготовить тестовый набор данных и сколько тестов необходимо разработать?

Методические указания содержат перечень вариантов заданий для контрольных работ, требования к оформлению контрольных работ и пример выполнения задания.

15. 1. Цель, задачи и реализуемые компетенции

Цель контрольной работы - знакомство с существующими стратегиями тестирования, приобретение навыков выбора стратегии и разработки тестов для отдельных задач, сравнение и оценка различных методов тестирования и их возможностей.

В результате изучения дисциплины у обучающихся должны быть сформированы следующие компетенции:

Индекс	Формулировка:
ПК-3	способность выполнять администрирование систем управления базами данных, системного программного обеспечения инфокоммуникационной системы организации, управление развитием инфокоммуникационной системы организации
ПК-6	способность проводить организационное сопровождение разработки, отладки, модификации и поддержки информационных технологий и систем

16. 2. Формулировка задания и его объем

1. Ознакомьтесь с теоретическими сведениями по стратегиям тестирования.

2. Внимательно изучите формулировку своего варианта задачи, подготовьте тесты по методикам стратегии "черного ящика". Предлагаемые тесты сведите в таблицу.

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

3. На одном из языков программирования напишите программу для решения задачи в соответствии с номером варианта. Сохраните первоначальный вариант программы.

4. Проведите тестирование программы и заполните подготовленную ранее таблицу с тестами.

5. 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.

6. Сделайте вывод о роли тестирования с использованием стратегии "черного ящика" и возможностях его применения. Сформулируйте его достоинства и недостатки.

Контрольная работа сдается в распечатанном виде и должна включать следующие разделы:

1. Постановку задачи.
2. Описание процедуры формирования тестов с использованием различных методов с использованием стратегии «черного ящика».
3. Описание алгоритма решения задачи (словесное или в виде блок-схемы).
4. Первоначальный код программы (до тестирования).
5. Таблица результатов первого тестирования.
6. Код программы после исправления.
7. Таблица результатов повторного тестирования.
8. Выводы о роли тестирования с использованием стратегии "черного ящика" и возможностях его применения.

17.

18. 3. Варианты заданий КОНТРОЛЬНОЙ РАБОТЫ

Номер варианта	ФИО студента
1	
2	
3	
4	
5	

Вариант 1. Создать программу, которая решает систему из 3-х линейных алгебраических уравнений с заданной точностью с помощью метода Крамера.

Вариант 2. Программа должна определять корни уравнения $y = x^2 - 2$ на заданном интервале (A,B) и с заданной точностью Eps на основе метода хорд.

Вариант 3. Разработать программу, которая создает массив из N целых чисел с помощью генератора случайных чисел в заданном диапазоне [A, B], а затем сортирует массив по возрастанию.

Вариант 4. Идентифицировать треугольник по трем сторонам (остроугольный, прямоугольный, тупоугольный, равносторонний, равнобедренный).

Вариант 5. Идентифицировать четырехугольник по четырем сторонам (квадрат или ромб, прямоугольник, трапеция или обыкновенный четырехуголъник).

19. 4. План-график выполнения задания

Дата получения задания	Дата предоставления выполненного задания
Зимняя сессия. Ноябрь 2018г.	Летняя сессия. За две недели до начала сессии. Май 2019 г.

20.

21. 5. Порядок защиты работы

Защита контрольной работы проводится в виде собеседования, в ходе которого студент отвечает на вопросы преподавателя по содержанию контрольной работы.

22. 6. Критерии оценивания работы

Оценка «отлично» выставляется студенту, если он продемонстрировал глубокие, исчерпывающие знания и творческие способности в понимании, изложении и использовании учебно-программного материала; логически последовательные, содержательные, полные, правильные и конкретные ответы на все поставленные вопросы и дополнительные вопросы преподавателя; свободное владение основной и дополнительной литературой, рекомендованной учебной программой.

Оценка «хорошо» выставляется студенту, если он продемонстрировал твердые и достаточно полные знания всего программного материала, правильное понимание сущности и взаимосвязи рассматриваемых процессов и явлений; последовательные, правильные, конкретные ответы на поставленные вопросы при свободном устранении замечаний по отдельным вопросам; достаточное владение литературой, рекомендованной учебной программой.

Оценка «удовлетворительно» выставляется студенту, если он продемонстрировал твердые знания и понимание основного программного материала; правильные, без грубых ошибок ответы на поставленные вопросы при устранении неточностей и несущественных ошибок в освещении отдельных положений при наводящих вопросах преподавателя; недостаточное владение литературой, рекомендованной учебной программой.

Оценка «неудовлетворительно» выставляется студенту, если он продемонстрировал неправильные ответы на основные вопросы, допущены грубые ошибки в ответах, непонимание сущности излагаемых вопросов; неуверенные и неточные ответы на дополнительные вопросы.

23. 7. Краткие теоретические сведения

Тестированием называется процесс выполнения программы ***с целью обнаружения ошибок***. Никакое тестирование не может доказать отсутствие ошибок в программе. Исходными данными для этапа тестирования являются техническое задание, спецификация и разработанные на предыдущих этапах структурная и функциональная схемы программного продукта, а для некоторых методов тестирования - алгоритм объекта тестирования.

При тестировании рекомендуется соблюдать следующие ***основные принципы***:

1. предполагаемые результаты должны быть известны до тестирования;
2. следует избегать тестирования программы автором;
3. необходимо досконально изучать результаты каждого теста;
4. необходимо проверять действия программы на неверных данных;
5. необходимо проверять программу на неожиданные побочные эффекты;
6. удачным считается тест, который обнаруживает хотя бы одну еще не обнаруженную ошибку;

7. вероятность наличия ошибки в части программы пропорциональна количеству ошибок, уже обнаруженных в этой части.

Различают четыре основные стратегии тестирования.

1. *Ручное тестирование*. Применяется в процессе разработки программы.

2. *Тестирование методом белого ящика.* Это подробное исследование внутренней логики и структуры программы. При этом необходимо знание исходного кода.

3. *Тестирование методом черного ящика.* Данная техника не требует знания программного кода и применяемого алгоритма решения задачи. Рассматриваются только основные аспекты системы, не связанные или мало связанные с ее внутренней логической структурой.

4. *Метод серого ящика.* Сочетает в себе предыдущие два подхода. Отладка с ограниченным знанием о внутреннем функционировании приложения сочетается со знанием основных аспектов системы.

7.1 Ручное тестирование программных продуктов

Эксперименты показали, что с точки зрения нахождения ошибок, достаточно эффективными являются методы ручного контроля. Поэтому один или несколько из них должны использоваться в каждом программном проекте. Методы ручного контроля предназначены для периода разработки, когда программа закодирована, но тестирование на машине еще не началось. Доказано, что эти методы способствуют существенному увеличению производительности и повышению надежности программ, и с их помощью можно находить от 30 до 70% ошибок логического проектирования и кодирования. Основными методами ручного тестирования являются:

- инспекции исходного текста;
- сквозные просмотры;
- просмотры за столом;
- обзоры программ.

7.1.1 Инспекции исходного текста (структурный контроль)

Инспекции исходного текста представляют собой набор процедур и приемов обнаружения ошибок при изучении текста группой специалистов, в которую входят автор программы, проектировщик, специалист по тестированию и координатор (компетентный программист, но не автор программы). Общая процедура инспекции состоит из следующих этапов:

1. участникам группы заранее выдается листинг программы и спецификация на нее;

2. программист рассказывает о логике работы программы и отвечает на вопросы инспекторов;

3. программа анализируется по списку вопросов для выявления исторически сложившихся общих ошибок программирования.

Кроме нахождения ошибок результаты инспекции позволяют программисту увидеть сделанные им ошибки, получить возможность оценить свой стиль программирования, выбор алгоритмов и методов тестирования. Инспекция является способом раннего выявления частей программы, с большей вероятностью содержащих ошибки, что позволяет при тестировании уделить внимание именно этим частям.

7.1.2 Сквозные просмотры

Сквозной просмотр, как и инспекция, представляет собой набор способов обнаружения ошибок, осуществляемых группой лиц, просматривающих текст программы. Такой просмотр имеет много общего с процессом инспектирования, но отличается процедурой и методами обнаружения ошибок. Группа по выполнению сквозного контроля состоит из 3-5 человек (председатель или координатор, секретарь, фиксирующий все ошибки, специалист по тестированию, программист и независимый эксперт).

Этапы процедуры сквозного контроля:

1) участникам группы заранее выдается листинг программы и спецификация на нее;

2) участникам заседания предлагается несколько тестов, написанных на бумаге, и тестовые данные подвергаются обработке в соответствии с логикой программы (каждый тест мысленно выполняется);

3) программисту задаются вопросы о логике проектирования и принятых допущениях;

4) состояние программы (значения переменных) отслеживается на бумаге или доске.

В большинстве сквозных просмотров при выполнении самих тестов находят меньше ошибок, чем при опросе программиста.

7.1.3 Проверка за столом

Третьим методом ручного обнаружения ошибок является применявшаяся ранее других методов «проверка за столом». Это проверка исходного текста или сквозные просмотры, выполняемые одним человеком, который читает текст программы, проверяет его по списку и пропускает через программу тестовые данные. Исходя из принципов тестирования, проверка за столом должна проводиться человеком, не являющимся автором программы.

Недостатки метода:

- проверка представляет собой полностью неупорядоченный процесс;
- отсутствие обмена мнениями и здоровой конкуренции;
- меньшая эффективность по сравнению с другими методами.

7.1.4 Оценка посредством просмотра

Этот метод непосредственно не связан с тестированием. Он является методом оценки анонимной программы в терминах ее общего качества, простоты эксплуатации и ясности. Цель этого метода - обеспечить сравнительно объективную оценку и самооценку программистов. Выбирается программист, который должен выполнять обязанности администратора процесса. Администратор набирает группу от 6 до 20 участников, которые должны быть одного профиля. Каждому участнику предлагается представить для рассмотрения две программы с его точки зрения наилучшую и наихудшую. Отобранные программы случайным образом распределяются между участниками. Им дается по 4 программы - две наилучшие и две наихудшие, но программист не знает, какая из них плохая, а какая хорошая. Программист просматривает их и заполняет анкету, в которой предлагается оценить их относительное качество по семибалльной шкале. Кроме того, проверяющий дает общий комментарий и рекомендации по улучшению программы.

7.2 Тестирование по принципу «белого ящика»

Стратегия тестирования по принципу «белого ящика», или стратегия тестирования, управляемая логикой программы (с учетом алгоритма), позволяет проверить внутреннюю структуру программы. В этом случае тестирующий получает тестовые данные путем анализа логики программы.

Исчерпывающему входному тестированию стратегии "черного ящика" здесь можно противопоставить исчерпывающее тестирование маршрутов (критерий покрытия маршрутов). Подразумевается, что программа проверена полностью, если с помощью тестов удается осуществить выполнение программы по всем возможным маршрутам передачи управления.

Однако нетрудно видеть, что даже в программе среднего уровня сложности число неповторяющихся маршрутов астрономическое и, следовательно, исчерпывающее тестирование маршрутов невозможно.

Кроме того, метод исчерпывающего тестирования маршрутов имеет ряд недостатков:

- метод не обнаруживает пропущенные маршруты;
- не обнаруживает ошибок, появление которых зависит от обрабатываемых данных (например, if (a-b)<eps - пропуск функции abs проявится только, если a<b);
- не дает гарантии, что программа соответствует описанию (например, если вместо сортировки по убыванию написана сортировка по возрастанию).

Стратегия «белого ящика» включает в себя следующие методы тестирования:

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

7.2.1 Покрытие операторов

Критерий покрытия операторов подразумевает выполнение каждого оператора программы, по крайней мере, один раз. Это необходимое, но не достаточное условие для приемлемого тестирования. Рассмотрим пример.

```
Procedure m(a,b:real; var x:real);

begin

        if(a>1) and (b=0) then x:=x/a;

        if (a=2) or (x>1) then x:=x+1;

end;
```

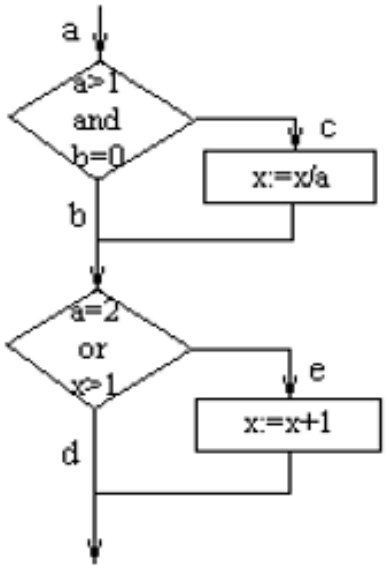

Рисунок 1 - Процедура и соответствующий ей алгоритм

Для приведенного фрагмента можно было бы выполнить каждый оператор один раз, задав в качестве входных данных a=2, b=0, x=3. Но при этом, из второго условия следует, что x может принимать любое значение, и оно не проверяется. Кроме того:

- если при написании программы в первом условии описать (a>1) or (b=0), то ошибка обнаружена не будет;
- если во втором условии вместо x>1 записано x>0, то эта ошибка тоже не будет обнаружена;
- существует путь abd (см. рис.1), в котором x вообще не меняется и, если здесь есть ошибка, она не будет обнаружена.

7.2.2 Покрытие решений (переходов)

Для реализации этого критерия необходимо достаточное число тестов, такое, что каждое решение на этих тестах принимает значение «истина» или «ложь», по крайней мере, один раз.

Нетрудно доказать, что критерий покрытия решений удовлетворяет критерию покрытия операторов, но является более сильным.

Программа, представленная на рис.1, может быть протестирована по методу покрытия решений двумя тестами, покрывающими либо пути ace abd, либо пути acd abe. Входные условия для первого теста: a=3, b=0, x=3. Входные условия для второго теста: a=2, b=1, x=1. Однако путь, где x не меняется, будет проверен с вероятностью 50%: если во втором условии вместо условия x>1 записано x<1, то ошибка не будет обнаружена двумя тестами.

7.2.3 Покрытие условий

Критерий покрытия условий более сильный по сравнению с предыдущим. В этом случае записывается число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении были выполнены, по крайней мере, один раз.

Однако, как и в случае покрытия решений, это покрытие не всегда приводит к выполнению каждого оператора, по крайней мере, один раз. К критерию требуется дополнение, заключающееся в том, что каждой точке входа управление должно быть передано, по крайней мере, один раз.

Программа на рис.1 имеет четыре условия: a>1, b=0, a=2, x>1. Необходимо реализовать ситуации, где a>1, a<=1, b=0, b≠0, a=2, a≠2, x>1, x<=1.

Тесты, удовлетворяющие этому условию:

1. a=2, b=0, x=4 (путь ace);
2. a=1, b=1, x=1 (путь abd).

Хотя количество тестов такое же, как в случае покрытия решений, этот критерий может (но не всегда) вызвать выполнение решений в условиях, не реализуемых при покрытии решений. То есть, этот критерий в основном удовлетворяет критерию покрытия решений, но не всегда. Тесты критерия покрытия условий для ранее рассмотренных примеров покрывают результаты всех решений, но это случайное совпадение. Например, тесты a=1,b=0,x=3 и a=2, b=1, x=1 покрывают результаты всех условий, но только два из четырех результатов решений (не выполняется результат «истина» первого решения и результат «ложь» второго).

7.2.4 Покрытие решений / условий

Этот метод требует составить тесты так, чтобы все возможные результаты каждого условия выполнились, по крайней мере, один раз, все результаты каждого решения выполнились, по крайней мере, один раз, и каждой точке входа управление передается, по крайней мере, один раз.

Недостатки метода:

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

7.2.5 Комбинаторное покрытие условий

Этот критерий требует создания такого числа тестов, чтобы все возможные комбинации результатов условий в каждом решении и все точки входа выполнялись, по крайней мере, один раз.

Для примера на рис.1 необходимо покрыть тестами восемь комбинаций:

1. a>1, b=0;
2. a>1, b≠0;
3. a<=1, b=0;
4. a<=1; b≠0;
5. a=2, x>1;
6. a=2, x<=1
7. a≠2, x>1;
8. a≠2, x<=1.

Эти комбинации можно проверить четырьмя тестами:

1. a=2, b=0, x=4 проверяет комбинации 1,5;
2. a=2, b=1, x=1 проверяет комбинации 2,6;
3. a=1, b=0, x=2 проверяет комбинации 3,7;
4. a=1, b=1, x=1 проверяет комбинации 4,8.

В данном случае то, что четырем тестам соответствуют четыре пути, является совпадением. Представленные тесты не покрывают всех путей, например, acd. Поэтому иногда необходима реализация восьми тестов. Таким образом, для программ, содержащих только одно условие на каждое решение, минимальным является критерий, набор тестов которого:

- вызывает выполнение всех результатов каждого решения, по крайней мере, один раз;
- передает управление каждой точке входа, по крайней мере, один раз (чтобы обеспечить выполнение каждого оператора, 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, по крайней мере, один раз. Термин «возможных» употреблен здесь потому, что некоторые комбинации условий могут быть нереализуемы. Например, для комбинации k<0 и k>40 задать k невозможно.

7.3 Тестирование по принципу «черного ящика»

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.

7.3.1 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 включать 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.

7.3.2 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.

7.3.3 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 причинно-следственных 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:

- по возможности выделять независимые группы причинно-следственных связей в отдельные таблицы;
- истина обозначается "1", ложь обозначается "0", для обозначения безразличных состояний условий применять обозначение "x", которое предполагает произвольное значение условия (0 или 1).

4) Каждая строка таблицы истинности преобразуется в тест. При этом по возможности следует совмещать тесты из независимых таблиц.

Недостаток метода - неадекватно исследует граничные условия.

7.3.4 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.

24. 8. Пример выполнения задания

Пусть необходимо выполнить тестирование программы, определяющей точку пересечения двух прямых на плоскости. Попутно программа должна определять параллельность прямой одной их осей координат. В основе программы лежит решение системы линейных уравнений:

$$\begin{cases} Ax + By = C \\ Dx + Ey = F \end{cases},$$

где A, B, C, D, E, F – заданные вещественные числа.

Напомним, что система линейных уравнений имеет единственное решение, если ее определитель $\Delta = A \cdot E - B \cdot D \neq 0$. В этом случае решение находят по формулам:

$$\begin{cases} x = \dfrac{\Delta x}{\Delta} \\ y = \dfrac{\Delta y}{\Delta} \end{cases},$$

где $\Delta x = \begin{vmatrix} C & B \\ F & E \end{vmatrix} = C \cdot E - F \cdot B, \quad \Delta y = \begin{vmatrix} A & C \\ D & F \end{vmatrix} = A \cdot F - D \cdot C$.

Если Δ=Δx=Δy=0, то решений бесконечно много (прямые совпадают).

Если Δ=0, Δx≠0 или Δy≠0, то система не имеет решений (прямые параллельны).

Рассмотрим различные методы тестирования с использованием стратегии «черного ящика».

1. Используя ***метод эквивалентных разбиений***, получаем для всех коэффициентов один правильный класс эквивалентности (коэффициент - вещественное число) и один неправильный (коэффициент - не вещественное число). Откуда можно предложить 7 тестов:

1) все коэффициенты - вещественные числа;

2) - 7) поочередно каждый из коэффициентов - не вещественное число (например, буква или другие символы).

Номер теста	Назначение теста	Значения исходных данных, A, B, C, D, E, F	Ожидаемый результат
1	Проверка входного условия «A, B, C, D, E, F – вещественные числа» (правильный класс)	1.2, 1, 1, 2, -2.5, -2.5	Прямые пересекаются в единственной точке (0, 1)
2	Проверка входного условия «A, B, C, D, E, F – вещественные числа» (неправильный класс)	a, 1, 0, 2, 5, 4	«Недопустимые данные»
3	Проверка входного условия «A, B, C, D, E, F – вещественные числа» (неправильный класс)	1, aa, 0, 2, 5, 4	«Недопустимые данные»
4	Проверка входного условия «A, B, C, D, E, F – вещественные числа» (неправильный класс)	0, 2, $, 1, 5, 4	«Недопустимые данные»
5	Проверка входного условия «A, B, C, D, E, F – вещественные числа» (неправильный класс)	10, 1, 0, D, 5, 4	«Недопустимые данные»
6	Проверка входного условия «A, B, C, D, E, F – вещественные числа» (неправильный класс)	1, 0, 2, 5, F, 4	«Недопустимые данные»
7	Проверка входного условия «A, B, C, D, E, F – вещественные числа» (неправильный класс)	1, 0, 2, 5, 4,)	«Недопустимые данные»

2. По ***методу граничных условий***: можно считать, что

для исходных данных граничные условия отсутствуют (коэффициенты - "любые" вещественные числа);

для результатов получаем, что возможны следующие варианты: единственное решение, прямые совпадают (множество решений), прямые параллельны (отсутствие решений). Следовательно, можно предложить тесты, с *результатами внутри области*:

1) результат - единственное решение (определитель системы $\Delta \neq 0$);
2) результат - множество решений ($\Delta = 0$ и $\Delta x = \Delta y = 0$);
3) результат - отсутствие решений ($\Delta = 0$, но $\Delta x \neq 0$ или $\Delta y \neq 0$);

и *с результатами на границе*:

1) $\Delta = 0{,}01$;
2) $\Delta = -0{,}01$;
3) $\Delta = 0$, $\Delta x = 0{,}01$, $\Delta y = 0$;
4) $\Delta = 0$, $\Delta y = -0{,}01$, $\Delta x = 0$.

Номер теста	Назначение теста	Значения исходных данных, A, B, C, D, E, F	Ожидаемый результат
8	Проверка случая единственного решения	1, 2, 8, 2, 3, 4	Прямые пересекаются в единственной точке (2, 3)
9	Проверка случая множества решений	1, 2, 3, 2, 4, 6	«Прямые совпадают»
10	Проверка случая отсутствия решений	2, -1, -3, 4, -2, 5	«Прямые параллельны»
11	Проверка граничного условия для результата (Δ=0,01)	1, 0, 1, 2, 0.01, -1	Прямые пересекаются в единственной точке (1, -300)
12	Проверка граничного условия для результата (Δ= -0,01)	-1, 0, 1, -2, 0.01, -1	Прямые пересекаются в единственной точке (-1, -300)
13	Проверка граничного условия для результата ($\Delta = 0$, $\Delta x = 0{,}01$, $\Delta y = 0$)	0, 1, 0.01, 0, 2, 0.01	«Прямые параллельны»
14	Проверка граничного условия для результата ($\Delta = 0$, $\Delta y = -0{,}01$, $\Delta x = 0$)	0, 2, 0.01, 0, 1, 0.01	«Прямые параллельны»

3. По ***методу анализа причинно-следственных связей***:

Определяем множество условий.

а) для определения типа прямой:

$\begin{cases} A = 0 \\ B = 0 \\ C = 0 \end{cases}$ - для определения типа и существования первой прямой

$\begin{cases} D = 0 \\ E = 0 \\ F = 0 \end{cases}$ - для определения типа и существования второй прямой

б) для определения точки пересечения:

$\begin{cases} \Delta = 0 \\ \Delta x = 0 \\ \Delta y = 0 \end{cases}$

Выделяем три группы причинно-следственных связей (определение типа и существования первой линии, определение типа и существования второй линии, определение точки пересечения) и строим таблицы истинности.

A=0	B=0	C=0	Результат
0	0	x	прямая общего вида
0	1	0	прямая, параллельная оси OY
0	1	1	ось OY
1	0	0	прямая, параллельная оси OX
1	0	1	ось OX
1	1	1	множество точек плоскости

D=0	E=0	F=0	Результат
0	0	x	прямая общего вида
0	1	0	прямая, параллельная оси OY
0	1	1	ось OY
1	0	0	прямая, параллельная оси OX
1	0	1	ось OX
1	1	1	множество точек плоскости

Δ=0	Δx=0	Δy=0	Единств. решение	Множество решений	Решений нет
0	x	x	1	0	0
1	0	x	0	0	1
1	x	0	0	0	1
1	1	1	0	1	0

Каждая строка этих таблиц преобразуется в тест. По возможности (с учетом независимости групп) берутся данные, соответствующие строкам сразу двух или всех трех таблиц.

В результате к уже имеющимся тестам добавляются:

1) проверки всех случаев расположения обеих прямых - 6 тестов по первой прямой вкладываются в 6 тестов по второй прямой так, чтобы варианты не совпадали, - 6 тестов;

2) выполняется отдельная проверка несовпадения условия $\Delta x = 0$ или $\Delta y = 0$ (в зависимости от того, какой тест был выбран по методу граничных условий) - тест также можно совместить с предыдущими 6 тестами.

Номер теста	Назначение теста	Значения исходных данных, A, B, C, D, E, F	Ожидаемый результат
15	Проверка случая прямых общего вида		
16	Проверка условия совпадения прямой с осью OX		
17	Проверка условия совпадения прямой с осью OY		
18	Проверка условия параллельности прямой оси OX		

19	Проверка условия параллельности прямой оси OY		
20	Проверка случая равенства нулю всех коэффициентов одной прямой		«Недопустимые данные»

4. ***По методу предположения об ошибке*** добавим тест:

Все коэффициенты - нули.

Номер теста	Назначение теста	Значения исходных данных, A, B, C, D, E, F	Ожидаемый результат
21	Проверка случая единственного решения	0, 0, 0, 0, 0, 0	«Недопустимые данные»

Всего получили 21 тест по всем четырем методикам. Заметим, что число тестов можно еще сократить (например, тесты 1, 8 и 15 можно объединить). (*Не забудьте для каждого теста заранее указывать результат!*).

25. 9. Учебно-методическое и информационное обеспечение дисциплины

9.1. Рекомендуемая литература

9.1.1 Основная литература

3. Луиза Тамре. Введение в тестирование программного обеспечения. - М.: Издательский дом «Вильямс», 2013. - С. 368.

4. Фаронов, В.В. Delphi. Программирование на языке высокого уровня: В. В. Фаронов- СПб.: Лидер, 2010.

9.1.2 Дополнительная литература

2. Гагарина, Л.Г. Современные проблемы информатики и вычислительной техники: учеб. пособие/ Л. Г. Гагарина, А. А. Петров- М.: ИД "ФОРУМ", 2011.

9.1.4 Интернет-ресурсы

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

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

7. http://www.biblioclub.ru - электронная библиотечная система «Университетская библиотека – online»: специализируется на учебных материалах для ВУЗов по научно-гуманитарной тематике, а так же содержит материалы по точным и

8. http://window.edu.ru – образовательные ресурсы ведущих вузов