

Министерство образования и науки РФ
Иркутский национальный исследовательский технический университет

Тестирование программ

Методические указания
по выполнению лабораторных работ

Издательство
Иркутского национального исследовательского технического университета
2018

УДК *****

Рецензент

Доктор техн. наук, зав.каф. кафедры Информационные системы и защита информации ФГБОУ ВО «ИрГУПС» Л.В.Аршинский.

Тестирование программ: Метод. указания по выполнению лабораторных работ / сост.: З.А.Бахвалова – Иркутск: Изд-во ИРНИТУ, 2018 - 36с.

Соответствуют требованиям ФГОС ВО 09.03.01- Информатика и вычислительная техника и 09.03.02 - Информационные системы и технологии.

Данные методические материалы содержат теоретические материалы и задания для выполнения лабораторной работы, посвященной знакомству со стратегиями и методами тестирования. Приведенные материалы используются при выполнении работ по курсам «Технологии разработки программных комплексов» и «Технология разработки программных комплексов».

©ФГБОУ ВО «ИРНИТУ», 2018

Учебное издание

Тестирование программ

Методические указания
по выполнению лабораторных работ

Составители:
Бахвалова Зинаида Андреевна

Оглавление

Введение.....	4
Задание:	5
Требования к выполнению работы:	5
Краткие теоретические сведения.....	7
1 Ручное тестирование программных продуктов.	7
1.1 Инспекции исходного текста (структурный контроль).	8
1.2 Сквозные просмотры.	8
1.3 Проверка за столом.	9
1.4 Оценка посредством просмотра.	9
2 Тестирование по принципу «белого ящика».....	9
2.1 Покрытие операторов.....	10
2.2 Покрытие решений (переходов).....	11
2.3 Покрытие условий	12
2.4 Покрытие решений/условий	13
2.5 Комбинаторное покрытие условий.	13
3 Тестирование по принципу «черного ящика» (функциональное тестирование).....	14
3.1 Эквивалентное разбиение	15
3.2 Анализ граничных значений.....	17
3.3 Анализ причинно-следственных связей.	18
3.4 Предположение об ошибке.	19
Пример тестирования программы по методу «черного ящика».	19
4 Оценочное тестирование.....	21
5 Тестирование пользовательского интерфейса.....	22
5.1 Функциональное тестирование пользовательских интерфейсов.....	23
5.2 Полнота покрытия пользовательского интерфейса	26
Общая стратегия тестирования.....	27
Отчет по тестированию программы	27
Порядок выполнения работы.	29
Варианты заданий	31
Список используемых источников.....	33
Приложение А. Перечень вопросов для структурного контроля текста.....	34

Введение

Одним из наиболее трудоемких этапов (от 30 до 60% общей трудоемкости) создания программного продукта является 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?

Цель 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.

Продолжительность работы - 4 часа.

Задание:

Написать программу расчета площади заштрихованной области геометрической фигуры. Варианты заданий указаны в табл. 1. Программа должна:

- строить геометрические фигуры и выводить их на экран,
- считать площадь заштрихованной области фигуры,
- изменять размеры построенной фигуры в пределах области окна,
- изменять размеры построенной фигуры пропорционально изменению размеров окна,
- фигуры, выводимые на экран, должны соответствовать введенным размерам,
- параметры фигур вводятся в сантиметрах(стороны) и градусах (углы),

•

Требования к выполнению работы:

- a) программа должна быть разработана студентом, выполняющим ее тестирование;
- b) для программы обязательна постановка решаемой задачи, которая включает задание и описание требований:
 - 1 функциональных:
 - бизнес требования - определяют назначение ПО,
 - пользовательские требования определяют набор пользовательских задач, которые должна решать программа, а также способы (сценарии) их решения в системе. Можно представить в виде диаграммы прецедентов (use case), наглядно демонстрирующей функции программы и действия пользователя;
 - 2 нефункциональных (к применяемому ПО, к применяемому оборудованию и т.д.)
- c) описание входных выходных параметров программы и ограничения на них;
- d) описание интерфейса;
- e) программа должна включать расчетную часть;
- f) описание алгоритма решения поставленной задачи (математическое описание);

- g) программа должна иметь современный пользовательский интерфейс;
- h) программа должна иметь возможность продемонстрировать методы тестирования для процедурного программирования или для объектно-ориентированного;
- i) качество программы может быть далеким от «хорошего».

Разработать тесты для выбранного программного продукта на основе различных стратегий тестирования (структурного тестирования и функционального тестирования).

Процесс построения набора тестов при использовании стратегии «белого ящика» должен включать три этапа:

- конструирование УГП (управляющего графа программы);
- выбор тестовых путей и его обоснование;
- генерация тестов с использованием и указанием в отчете различных критериев тестирования.

В процессе построения набора тестов при функциональном (системном) тестировании необходимо по возможности использовать:

- Метод эквивалентного разбиения;
- Анализ граничных значений;
- Метод функциональных диаграмм причинно-следственных связей;
- Тестирование пользовательского интерфейса программы;
- Метод предположения об ошибке;
- Нагрузочное тестирование.

Выполнить тестирование программы по разработанному набору тестов с использованием ручного тестирования.

Представить отчет о выполнении лабораторной работы по схеме:

- титульный лист;
- постановка задачи;
- описания задачи и алгоритма ее решения;
- наборы тестов для каждой стратегии тестирования с пояснениями;
- исходный текст программы
- результаты тестирования, представленные в виде таблиц;
- протокол по результатам тестирования и выводы о качестве тестируемого программного продукта и наиболее удачных тестах.

Краткие теоретические сведения.

Тестированием называется процесс выполнения программы **с целью обнаружения ошибки(!)**. 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 не-которых методов тестирования - алгоритм объекта тестирования.

При тестировании рекомендуется соблюдать следующие *основные принципы*:

- a) Предполагаемые результаты должны быть известны до тестирования.
- b) Следует избегать тестирования программы автором.
- c) Необходимо досконально изучать результаты каждого теста.
- d) Необходимо проверять действия программы на неверных данных.
- e) Необходимо проверять программу на неожиданные побочные эффекты.
- f) Удачным считается тест, который обнаруживает хотя бы одну еще не обнаруженную ошибку.
- g) Вероятность наличия ошибки в части программы пропорциональна количеству ошибок, уже обнаруженных в этой части.

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.

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:

- a. участникам группы заранее выдается листинг программы и спецификация на нее;
- b. программист рассказывает о логике работы программы и отвечает на вопросы инспекторов;
- c. 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.

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:

- a) участникам группы заранее выдается листинг программы и спецификация на нее;
- b) участникам заседания предлагается несколько тестов, написанных на бумаге, и тестовые данные подвергаются обработке в соответствии с логикой программы (каждый тест мысленно выполняется);
- c) программисту задаются вопросы о логике проектирования и принятых допущениях;
- d) состояние программы (значения переменных) отслеживается на бумаге или доске.

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

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, проверку 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.

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.

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 сопоставить исчерпывающее тестирование маршрутов (*критерий покрытия маршрутов*). Подразумевается, что программа проверена полностью, если с помощью тестов удастся осуществить выполнение программы по всем возможным маршрутам передач 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.

2.1 Покрытие операторов

Критерий покрытия операторов подразумевает выполнение каждого оператора программы, по крайней мере, один раз. Это необходимое, но недостаточное 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;
```

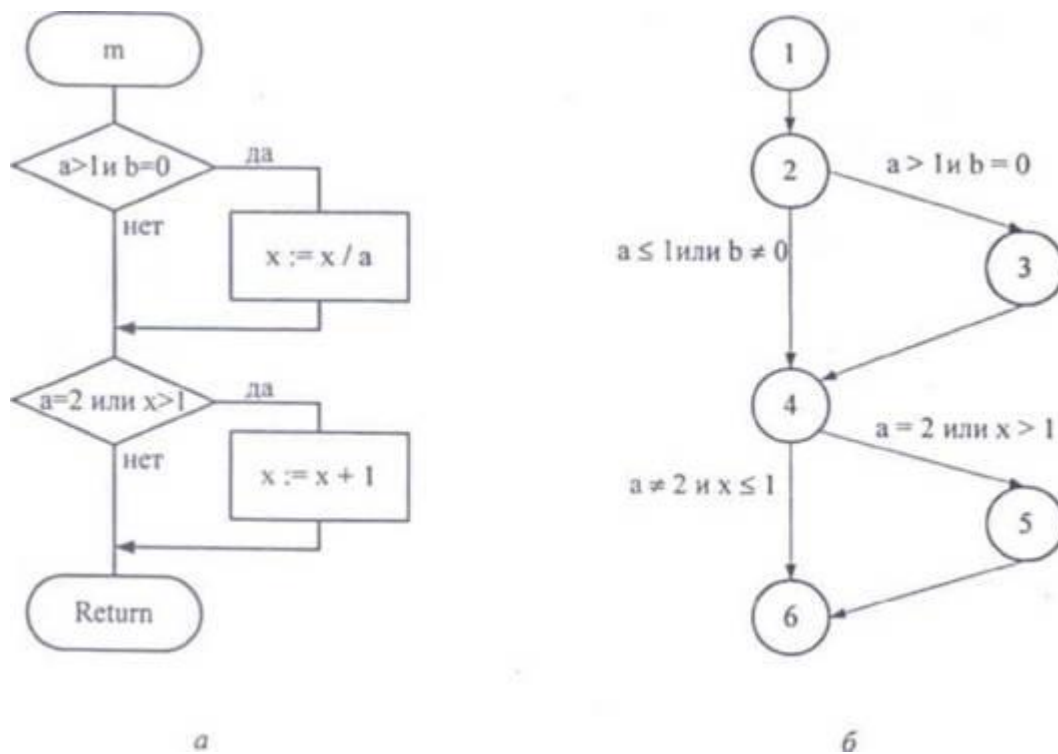


Рисунок 1- Схема алгоритма процедуры примера (а) и ее граф передач управления (б)

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 1246 (см. Рис.1), в котором x вообще не меняется и, если здесь есть ошибка, она не будет обнаружена.

Таким образом, хотя при тестировании действительно необходимо задавать исходные данные так, чтобы все операторы программы были выполнены хотя бы один раз, для проверки программы этого явно недостаточно.

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 покрытие операторов, но является более сильным.

Программа, представленная на рис.1, может быть протестирована по методу покрытия решений двумя тестами, покрывающими либо пути *1246,123456*, либо пути *12346, 12456*.

Входные условия для первого теста: $a=3, b=0, x=3$. Путь *123456*

Входные условия для второго теста: $a=2, b=1, x=1$. Путь *1246*

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.

2.3 Покрытие условий

Критерий покрытия условий более «сильный» по сравнению с предыдущим. В этом случае формируют некоторое количество 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 все возможные ситуации:

$a>1, a\leq 1, \quad b=0, b\neq 0, \quad a=2, a\neq 2, \quad x>1, x\leq 1.$

Тесты, удовлетворяющие этому условию:

1 $a=2, b=0, x=4$ (путь *123456*);

2 $a=1, b=1, x=1$ (путь *1246*);

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.

Критерий покрытия условий часто 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

$a=1, b=0, x=3$

$a=2, b=1, x=1$

покрывают результаты всех условий, но только два из четырех результатов решений (не выполняется результат «истина» первого решения и результат «ложь» второго).

2.4 Покрывание решений/условий

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

Анализ, проведенный выше, показывает, что этому критерию удовлетворяют тесты:

$a=2, b=0, x=4$ – путь 1234556

$a=1, b=1, x=1$ – путь 1246

Недостатки метода:

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

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 $a>1, b=0$; | 5 $a=2, x>1$; |
| 2 $a>1, b\neq 0$; | 6 $a=2, x\leq 1$ |
| 3 $a\leq 1, b=0$; | 7 $a\neq 2, x>1$; |
| 4 $a\leq 1, b\neq 0$; | 8 $a\neq 2, x\leq 1$. |

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 соответствует четыре пути, является совпадением. Представленные тесты не покрывают всех путей, например, 12346. 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.

3 Тестирование по принципу «черного ящика» (функциональное тестирование).

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 выяснения 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.

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 $1 \leq X \leq 999$ 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 обусловлено тем, что определенные проверки с ошибочными входами скрывают или заменяют другие проверки с ошибочными входами.

Недостатком метода эквивалентных разбиения в том, что он не исследует комбинации входных условий.

3.2 Анализ граничных значений.

Граничные условия - это ситуации, возникающие на, 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. Вторым недостатком связан с тем, что метод анализа граничных условий не позволяет проверять различные сочетания исходных данных.

3.3 Анализ причинно-следственных связей.

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

Для использования метода необходимо понимание булевой 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. Каждой причине и следствию присписывается 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).

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

Недостаток метода - неадекватно исследует граничные условия.

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.

Пример тестирования программы по методу «черного ящика».

Пусть необходимо выполнить тестирование программы, определяющей точку пересечения двух прямых на плоскости. Попутно, она должна определять параллельность прямой одной из осей координат.

В основе программы лежит решение системы линейных уравнений:

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

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.

2 По методу граничных условий:

- можно считать, что для исходных данных граничные условия отсутствуют (коэффициенты - "любые" вещественные числа);
- для результатов - получаем, что возможны варианты: единственное решение, прямые сливаются (множество решений), прямые параллельны (отсутствие решений).

Следовательно, можно предложить тесты, с результатами внутри области:

- 1 результат - единственное решение ($\delta \neq 0$);
- 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 $\delta = 0, \delta y = -0,01, \delta x = 0$.

3 По методу анализа причинно-следственных связей:

Определяем множество условий.

а) для определения типа прямой:

$\left. \begin{array}{l} a=0 \\ b=0 \\ c=0 \end{array} \right\}$ - для определения типа и существования первой прямой

$\left. \begin{array}{l} d=0 \\ e=0 \\ f=0 \end{array} \right\}$ для определения типа и существования второй прямой

б) для определения точки пересечения:

$$\delta = 0$$

$$\delta x = 0$$

$$\delta y = 0$$

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 положения
0	1	0	прямая, параллельная оси OX
0	1	1	ось OX
1	0	0	прямая, параллельная оси OY
1	0	1	ось OY
1	1	X	множество точек плоскости

Такая же таблица строится для второй прямой.

$\delta = 0$	$\delta x = 0$	$\delta y = 0$	Ед. реш.	Мн. реш.	Реш. 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. При возможности (с учетом независимости групп) берутся данные, соответствующие строкам сразу двух или всех трех таблиц.

В результате к уже имеющимся тестам добавляются

а) 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;

б) 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;

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

Все коэффициенты - нули.

Всего получили 20 тестов по всем четырем методикам. Если еще попробовать вложить независимые проверки, то возможно число тестов можно еще сократить. (**Не забудьте для каждого теста заранее указывать результат!**).

4 Оценочное тестирование

После завершения функционального и структурного тестирования приступают к оценочному тестированию, целью которого является тестирование программы на соответствие основным требованиям. Эта стадия тестирования особенно важна для программных продуктов, предназначенных для продажи на рынке.

Оценочное тестирование, которое также называют «тестированием системы в целом», включает следующие виды:

– тестирование удобства использования - последовательная проверка соответствия программного продукта и документации на него основным положениям технического задания;

– тестирование на предельных объемах - проверка работоспособности программы на максимально больших объемах данных, например, объемах текстов, таблиц, большом количестве файлов и т. п.;

– тестирование на предельных нагрузках - проверка выполнения программы на возможность обработки большого объема данных, поступивших в течение короткого времени;

- тестирование удобства эксплуатации - анализ психологических факторов, возникающих при работе с программным обеспечением; это тестирование позволяет определить, удобен ли интерфейс, не раздражает ли цветовое или звуковое сопровождение и т. п.;
- тестирование защиты - проверка защиты, например, от несанкционированного доступа к информации;
- тестирование производительности - определение пропускной способности при заданной конфигурации и нагрузке;
- тестирование требований к памяти - определение реальных потребностей в оперативной и внешней памяти;
- тестирование конфигурации оборудования - проверка работоспособности программного обеспечения на разном оборудовании;
- тестирование совместимости - проверка преемственности версий: в тех случаях, если очередная версия системы меняет форматы данных, она должна предусматривать специальные конвекторы, обеспечивающие возможность работы с файлами, созданными предыдущей версией системы;
- тестирование удобства установки - проверка удобства установки;
- тестирование надежности - проверка надежности с использованием соответствующих математических моделей [66];
- тестирование восстановления - проверка восстановления программного обеспечения, например системы, включающей базу данных, после сбоя оборудования и программы;
- тестирование удобства обслуживания - проверка средств обслуживания, включенных в программное обеспечение;
- тестирование документации - тщательная проверка документации, например, если документация содержит примеры, то их все необходимо попробовать;
- тестирование процедуры - проверка ручных процессов, предполагаемых в системе.

Естественно, целью всех этих проверок является поиск несоответствий техническому заданию. Считают, что только после выполнения всех видов тестирования программный продукт может быть представлен пользователю или к реализации. Однако на практике обычно выполняют не все виды оценочного тестирования, так как это очень дорого и трудоемко. Как правило, для каждого типа программного обеспечения выполняют те виды тестирования, которые являются для него наиболее важными. Так базы данных обязательно тестируют на предельных объемах, а системы реального времени - на предельных нагрузках.

5 Тестирование пользовательского интерфейса

Часть программной системы, обеспечивающая работу интерфейса с пользователем - один из наиболее нетривиальных объектов для верификации. Не-

тривиальность заключается в двояком восприятии термина "пользовательский интерфейс".

С одной стороны, пользовательский интерфейс - часть программной системы. Соответственно, на пользовательский интерфейс пишутся функциональные и низкоуровневые требования

С другой стороны пользовательский интерфейс - лицо системы, от организации и продуманности которого зависит эффективность работы пользователя с приложением.

Поэтому очень важно проводить тестирование пользовательского интерфейса, чтобы определить, насколько он является удобным для пользователя и соответствует ли его требованиям.

Проверка интерфейса на эффективность человеко-машинного взаимодействия получила название проверки удобства использования

В данной работе следует рассмотреть общие вопросы, как функционального тестирования пользовательских интерфейсов, так и тестирования удобства использования.

5.1 Функциональное тестирование пользовательских интерфейсов

Функциональное тестирование пользовательского интерфейса состоит из пяти фаз:

- анализ требований к пользовательскому интерфейсу;
- разработка тест-требований и тест-планов для проверки пользовательского интерфейса (тест-планы для проверки пользовательского интерфейса, как правило, представляют собой сценарии, описывающие действия пользователя при работе с системой. Сценарии могут быть записаны либо на естественном языке, либо на формальном языке какой-либо системы автоматизации пользовательского интерфейса. Выполнение тестов при этом производится либо оператором в ручном режиме, либо системой, которая эмулирует поведение оператора);
- выполнение тестовых примеров и сбор информации о выполнении тестов (обычно применяются технологии анализа выводимых на экран форм и их элементов (в случае графического интерфейса) или выводимого на экран текста (в случае текстового), а не проверка значений тех или иных переменных, устанавливаемых программной системой.);
- определение полноты покрытия пользовательского интерфейса требованиями (понимается то, что в результате выполнения всех тестовых примеров каждый интерфейсный элемент был использован хотя бы один раз во всех доступных режимах);

- составление отчетов о проблемах в случае несовпадения поведения системы и требований либо в случае отсутствия требований на отдельные интерфейсные элементы (отчеты могут включать в себя как описания несоответствий требованиям и реального поведения системы, так и описания проблем в требованиях к пользовательскому интерфейсу. Основным источником проблем в этих требованиях - их тестонепригодность, вызванная расплывчатостью формулировок и неконкретностью).

Типы требований к пользовательскому интерфейсу

Требования к пользовательскому интерфейсу могут быть разбиты на две группы:

- требования к внешнему виду пользовательского интерфейса и формам взаимодействия с пользователем;
- требования по доступу к внутренней функциональности системы при помощи пользовательского интерфейса.

Другими словами, первая группа требований описывает взаимодействие подсистемы интерфейса с пользователем, а вторая - с внутренней логикой системы.

К первой группе можно отнести следующие типы требований:

- Требования к размещению элементов управления на экранных формах

Данные требования могут определять общие принципы размещения элементов пользовательского интерфейса или требования к размещению конкретных элементов. Например, общие требования по размещению элементов на графической экранной форме могут выглядеть следующим образом:

Каждое окно приложения должно быть разбито на три части: строка меню, рабочая область и статусная строка. Строка меню должна быть горизонтальной и прижатой к верхней части окна, статусная строка должна быть горизонтальной и прижатой к нижней части окна, рабочая область должна находиться между строкой меню и статусной строкой и занимать всю оставшуюся площадь окна.

При тестировании данного требования достаточно определить, что в каждом окне системы действительно присутствуют три части, которые расположены и прижаты согласно требованиям даже при изменении размеров окна, его сворачивании/разворачивании, перемещении по экрану, при перекрытии его другими окнами.

Примером требований по размещению конкретного элемента может служить следующее:

Кнопка "Начать передачу" должна находиться непосредственно под строкой меню в левой части рабочей области окна.

При тестировании такого требования также необходимо определить, сохраняется ли расположение элемента при изменении размера окна, а также при использовании элемента (в данном случае - при нажатии).

- Требования к содержанию и оформлению выводимых сообщений

Требования к содержанию и оформлению выводимых сообщений определяют текст сообщений, выводимых системой, его шрифтовое и цветовое оформление. Также часто в таких требованиях определяется, в каких случаях выводится то или иное сообщение.

Так, например, для тестирования требования:

Сообщение "Невозможно открыть файл" должно выводиться в статусную строку прижатым к левому краю, красным цветом, полужирным шрифтом в случае недоступности открываемого файла по чтению.

- необходимо проверить, что при возникновении указанной ситуации сообщение действительно выводится согласно требованиям

- Требования к форматам ввода

Данная группа требований определяет, в каком виде информация поступает от пользователя в систему. При этом кроме собственно требований, определяющих корректный формат, к этой группе относятся требования, определяющие реакцию системы на некорректный ввод. Для проверки таких требований необходимо проверять как корректный ввод, так и некорректный. Желательно при этом разбивать различные варианты ввода на классы эквивалентности (как минимум на два - корректные и некорректные).

Ко второй группе относятся следующие типы требований:

- Требования к реакции системы на ввод пользователя

Данный тип требований определяет связь внутренней логики системы и интерфейсных элементов. Например,

При нажатии кнопки "Сброс" значение таймера синхронизации передачи должно сбрасываться в 0.

Для проверки такого требования в тестовом примере должно быть симулировано нажатие на кнопку "Сброс", после чего должна проводиться проверка значения таймера. Однако некоторые требования определяют в качестве реакции системы не то, как меняется ее внутреннее состояние, а реакцию пользовательского интерфейса. Например, в требовании:

При нажатии кнопки "Отложенный сброс" должно выводиться окно "Ввод значения времени для отложенного сброса" -

в качестве реакции на использование одного интерфейсного элемента определяется появление другого интерфейсного элемента. Такие требования проверяются при помощи имитации ввода пользователя и анализа появляющихся интерфейсных элементов.

- Требования к времени отклика на команды пользователя

В качестве отдельного типа требований можно выделить требования к времени отклика системы на различные пользовательские операции. Это связано с тем, что подсознательно пользователь воспринимает операции продолжительностью более 1 секунды как длительные. Если в этот момент система не сообщает пользователю о том, что она выполняет какую-либо операцию, пользователь начнет считать, что система зависла или работает в неверном режиме. В связи с этим либо каждое предельное время отклика должно быть указано в требованиях и пользовательской документации, либо во время длительных операций должны выводиться информационные сообщения (например, индикатор прогресса). Значения предельного времени и равномерность увеличения значений индикатора прогресса должны проверяться соответствующими тестами.

5.2 Полнота покрытия пользовательского интерфейса

При определении понятия покрытия пользовательского интерфейса можно ввести следующие его уровни:

- **функциональное покрытие** - покрытие требований к пользовательскому интерфейсу;
- **структурное покрытие** - для обеспечения полного структурного покрытия каждый интерфейсный элемент должен быть использован в тестовых примерах хотя бы один раз;
- **структурное покрытие с учетом состояния элементов интерфейса** - для обеспечения этого уровня покрытия необходимо не только использовать каждый элемент интерфейса, но и привести его во все возможные состояния (например, для чек-боксов - отмечен/не отмечен, для полей ввода - пустое/заполненное не целиком/заполненное полностью и т.п.)
- **структурное покрытие с учетом состояния элементов интерфейса и внутреннего состояния системы** - поведение некоторых интерфейсных элементов может изменяться в зависимости от внутреннего состояния системы. Каждое такое различимое поведение интерфейсного элемента должно быть проверено. Например, система может иметь два режима работы - нормальный и для начинающего пользователя, в котором нажатие каждого элемента сопровождается появлением всплывающей подсказки.

В этом случае нужно проверить оба режима и при этом проверить, что подсказки появляются только в режиме для начинающих.

При определении степени покрытия необходимо учитывать, что реакция на некоторые интерфейсные элементы определяется не программной системой, а на уровне операционной системы или среды выполнения. Так, например, реакция на использование многих интерфейсных элементов стандартного диалогового окна открытия файла определяется операционной системой и может не тестироваться.

Если уровень покрытия интерфейсных элементов тестами недостаточен, это является сигналом либо к уточнению требований к пользовательскому интерфейсу, либо к снижению степени подробности тестирования.

Общая стратегия тестирования.

Все методологии проектирования тестов могут быть объединены в общую стратегию. Это оправдано тем, что каждый метод обеспечивает создание определенного набора тестов, но ни один из них сам по себе не может дать полный набор тестов. Приемлемая стратегия состоит в следующем:

- 1 Если спецификация состоит из комбинации входных условий, то начать рекомендуется с применения метода функциональных диаграмм.
- 2 В любом случае необходимо использовать анализ граничных значений.
- 3 Определить правильные и неправильные классы эквивалентности для входных и выходных данных и дополнить, если это необходимо, тесты, построенные на предыдущих шагах.
- 4 Для получения дополнительных тестов рекомендуется использовать метод предположения об ошибке.
- 5 Проверить логику программы на полученном наборе тестов. Для этого воспользоваться критериями покрытия решений, покрытия условий, покрытия решений / условий либо комбинаторного покрытия условий. Если необходимость выполнения критерия покрытия требует построения тестов, не встречающихся среди построенных на предыдущих шагах, то следует дополнить уже построенный набор

Отчет по тестированию программы

Каждое отклонение от спецификации обязательно документируют, заполняя специальный протокол (рис.2)

Название компании № _____	Конфиденциально	Отчет о проблеме
Программа _____	Выпуск _____	Версия _____
Тип проблемы (1-6) _____	Степень важности (1-3) _____	Приложения (Д/Н) _____
1 - Ошибка кодирования	1 - Фатальная	Если да, то какие
2 - Ошибка проектирования	2 - Серьезная	
3 - Предложение	3 - Незначительная	
4 - Расхождение с документацией		
5 - Взаимодействие с аппаратурой		
6 - Вопрос		
ПРОБЛЕМА		
Можете ли вы воспроизвести проблемную ситуацию? (Д/Н) _____		
Подробное описание проблемы и способа ее воспроизведения: _____		

Предлагаемое исправление (необязательно): _____		

Отчет представлен сотрудником _____ Дата __/__/__		
Заполняется разработчиком		
Функциональная область _____	Ответственный _____	
Комментарии _____		

Состояние (1-2) _____	Приоритет (1-5) _____	
1 - Открыто 2 - Закрыто		
Резолюция (1-9) _____	Иправленная версия _____	
1 - Рассматривается	6 - Не может быть исправлено	
2 - Исправлено	7 - Отозвано составителем	
3 - Не воспроизводится	8 - Нужна дополнительная информация	
4 - Отложено	9 - Не согласен с предложением	
5 - Соответствует проекту		
Рассмотрено _____	Дата __/__/__	
Проконтролировано _____	Дата __/__/__	
Считать отложенным (Д/Н) _____		

Рисунок 2 – Бланк отчета об обнаруженном несоответствии

Порядок выполнения работы.

1. Ознакомьтесь с теоретическими сведениями по стратегиям тестирования.

2. Для своего варианта задания выполните **структурный контроль**, используя перечень вопросов в Приложении. В процессе выполнения заполните таблицу вида:

Номер вопроса	Строки, подлежащие проверке	Результат проверки	Вывод
1.1	4, 8,9	a=0 b=67 с - вводится	Все переменные инициализированы
1.2			
и т.д.			

Сделайте общий вывод о роли структурного контроля в процессе создания программы. Сформулируйте его достоинства и недостатки.

3. Для заданного фрагмента схемы алгоритма подготовьте тесты, используя методы **стратегии "белого ящика"**. Предлагаемые тесты сведите в таблицу.

Номер теста	Назначение теста	Значения исходных данных	Ожидаемый результат
-------------	------------------	--------------------------	---------------------

Сравните тесты, предлагаемые различными методами. Сделайте вывод о роли тестирования с использованием стратегии "белого ящика" и возможностях его применения. Сформулируйте его достоинства и недостатки.

4. 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
-------------	------------------	--------------------------	---------------------	-------------------	-------

Напишите программу, согласно выданному заданию. Выполните тестирование. Занесите в таблицу результаты.



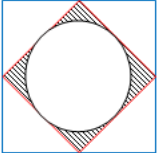
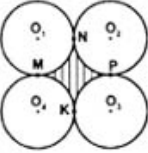
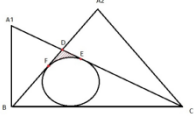
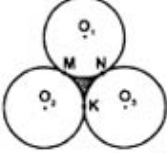

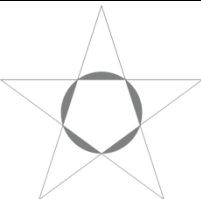
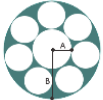
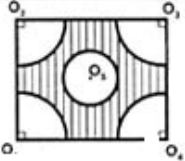

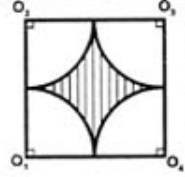

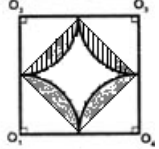
Сделайте вывод о роли тестирования с использованием стратегии "черного ящика" и возможностях его применения. Сформулируйте его достоинства и недостатки.


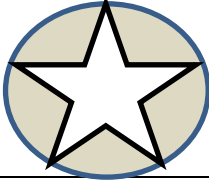
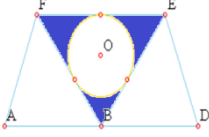
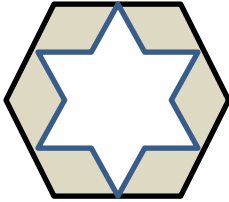
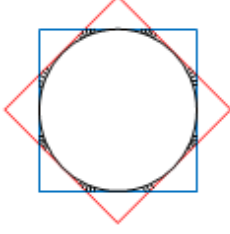
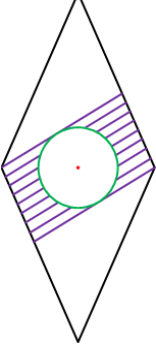

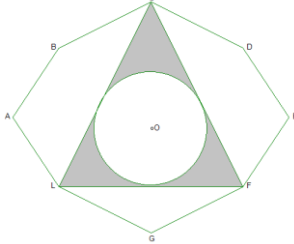
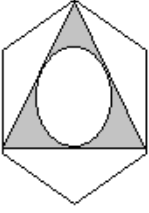
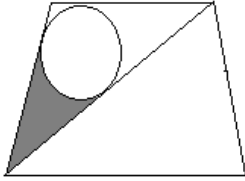
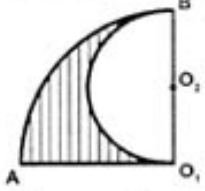
Пример сценария для ручного тестирования пользовательского интерфейса

№ п/п	Действие	Реакция системы	Результат
1	Щелкните на пиктограмме System и выберите пункт меню 'System Management Applet'.	Появится окно ввода логина и пароля	Верно
2	Введите в появившееся окно ввода имя пользователя 'guest1' и пароль 'guest'. Затем нажмите кнопку 'Login'.	Появится окно 'System Management Applet'. В верхнем правом углу должно быть выведено имя вошедшего пользователя guest1 Все опции в окне должны быть отключены (выведены серым цветом)	Неверно Окно имеет название 'System Management Application'
3	Завершите сеанс работы с апплетом щелчком по пиктограмме 'Logout'.	Окно 'System Management Applet' должно быть закрыто	

Варианты заданий

Таблица 1- Варианты заданий

Вариант	Задание	Вариант	Задание
1		14	
2		15	
3		16	
4		17	
5		18	
6		19	
7		120	

8		21	
9		22	
10		23	
11		24	
12		25	
13		26	

Список используемых источников.

- 1 . Майерс Г. Искусство тестирования программ / Пер. с англ. под ред. Б. А. Позина. - М.: Финансы и статистика, 1982.-176с.,ил.
2. Иванова Г.С. Технология программирования. -М.:КноРус – 2011- 333с.
3. Куликов, С.С. Тестирование программного обеспечения. Базовый курс: практ. пособие. – Минск: Четыре четверти, 2015. – 294 с.
4. Глоссарий международной организации по сертификации тестирования программного обеспечения ISTQB: [сайт]. [2018]. URL: <http://glossary.istqb.org/> (дата обращения: 03.01.2018).
5. Вигерс, Карл. Разработка требований к программному обеспечению / Пер, с англ. – М.: Издательско-торговый дом «Русская Редакция», 2004. – 576с.
6. Канер, Сэм. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений: Пер. с англ. / Сэм Канер, Джек Фолк, Енг Кек Нгуен. – К.: Издательство «ДиаСофт», 2001. – 544 с.
7. Формализация требований на практике. В.В. Кулямин и др. [Электронный ресурс] // Сайт Института системного программирования им. В.П. Иванникова РАН: [сайт]. URL: <http://panda.ispras.ru/~kuliain/docs/Req-2006-ru.pdf> (дата обращения: 05.01.2018).

Приложение А. Перечень вопросов для структурного контроля текста.

1. Обращения к данным.

- 1) Все ли переменные инициализированы?
- 2) Не превышены ли максимальные (или реальные) размеры массивов и строк?
- 3) Не перепутаны ли строки со столбцами при работе с матрицами?
- 4) Присутствуют ли переменные со сходными именами?
- 5) Используются ли файлы? Если да, то
 - При вводе из файла проверяется ли завершение файла?
 - Соответствуют ли типы записываемых и читаемых значений?
- 6) Использованы ли нетипизированные переменные, открытые массивы, динамическая память? Если да, то
 - Соответствуют ли типы переменных при "наложении" формата?
 - Не выходят ли индексы за границы массивов?

2. Вычисления.

- 1) Правильно ли записаны выражения (порядок следования операторов)?
- 2) Корректно ли производятся вычисления неарифметических переменных?
- 3) Корректно ли выполнены вычисления с переменными различных типов (в том числе с использованием целочисленной арифметики)?
- 4) Возможно ли переполнение разрядной сетки или ситуация машинного нуля?
- 5) Соответствуют ли вычисления заданным требованиям точности?
- 6) Присутствуют ли сравнения переменных различных типов?

3. Передачи управления.

- 1) Будут ли корректно завершены циклы?
- 2) Будет ли завершена программа?
- 3) Существуют ли циклы, которые не будут выполняться из-за нарушения условия входа? Корректно ли продолжатся вычисления?
- 4) Существуют ли поисковые циклы? Корректно ли отрабатываются ситуации "элемент найден" и "элемент не найден"?

4. Интерфейс.

- 1) Соответствуют ли списки параметров и аргументов по порядку, типу, единицам измерения?

2) Не изменяет ли подпрограмма аргументов, которые не должны изменяться?

3) Не происходит ли нарушения области действия глобальных и локальных переменных с одинаковыми именами?