Московский Государственный Технический Университет имени Н. Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Компьютерные системы и сети»

Иванова Г.С., Ничушкина Т.Н.

УТВЕРЖДАЮ
Зав. кафедрой ИУ6

д.т.н., проф. _____ Сюзев В.В.
"____"_____2013 г.

# Тестирование программного обеспечения

Методические указания по выполнению лабораторной работы

по дисциплине "Технология разработки программных систем"

Москва 2013

**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.

**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 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.

## 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:

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.

### *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.

### *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 удается 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)and(x>1) then x:=x+1;
  end;
```

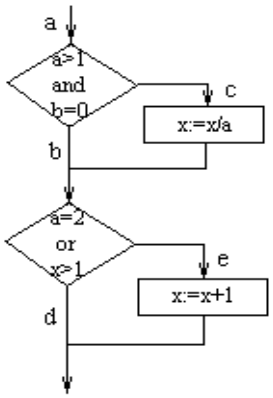


Рис.1 Процедура и соответствующий ей алгоритм.

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.

### *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 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.

### *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, f=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).

### *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.

### *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 соответствует 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.

## 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 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 $1 \le X \le 999$ 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 непо-

крытых 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. Второй недостаток 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 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;
- для классов эквивалентности входных условий дополнительно необходимо

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.**

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

В основе программы лежит решение системы линейных уравнений:

Ax+By=C и Dx+Ey=F.

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:
a=0 ⎫
b=0 ⎬ - для определения типа и существования первой прямой
c=0 ⎭
d=0 ⎫
e=0 ⎬ - для определения типа и существования второй прямой
f=0 ⎭

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 OУ |
| 1 | 0 | 1 | ось OУ |
| 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

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;
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!**).

**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

**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<br>b=67<br>c - вводится | Все переменные инициализированы |

| | | | |
|---|---|---|---|
| 1.2 | - | | |
| и т.д. | | | |

Примечание. Вопросы, которые не актуальны для данной программы можно в таблице не фиксировать.

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 |
|---|---|---|---|---|---|
| | | | | | |

Получите у преподавателя выполняемый модуль программы. 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.

**Формулировки задач к пункту 4.**

*Задача 1.*
Реализовать калькулятор, который выполняет два действия «+» и «-» с действительными числами.

*Задача 2.*
Реализовать калькулятор, который выполняет два действия «+» и «-» с целыми числами.

*Задача 3.*

Разработать программу решения уравнения $ax^2 + bx + c = 0$ , где a, b, c - любые вещественные числа.

*Задача 4.*

Программа должна определять корни уравнения $y = x^2 - 2$ на заданном интервале и с заданной точностью на основе «Метода хорд».

*Задача 5.*
Создать программу , которая позволяет в заданном диапазоне и количестве с помощью генератора случайных чисел создавать массив, а затем сортировать его.

*Задача 6.*
Создать программу , которая решает систему из 3-х линейных алгебраических уравнений с заданной точностью с помощью итерационного метода. Если количество итераций превысит максимально допустимое, то программа должна выдавать сообщение о том, что сходимости нет.

*Задача 7.*

Программа должна вычислять с заданной точностью значение интеграла функции $F(x) = x^2$ на введенном с клавиатуры интервале от a до b.

*Задача 8.*

Программа должна вычислять значение интеграла функции $F(x) = \frac{1}{x}$ . Исходными данными являются : интервал и количество шагов.

*Задача 9.*

Написать программу, которая выводит на экран гистограмму. Параметры гистограммы должны вводится с помощью таблицы.

*Задача10.*

Написать программу, которая реализует секундомер. Пользователю должны выводиться сотые доли секунды, секунды и минуты.

*Задача11.*

Программа должна строить график функции по заданным в таблице значениям. Обеспечить возможность выбора вида графика : точки отдельно или точки соединены.

*Задача12.*

Программа должна определять во сколько рублей обойдется поездка на дачу (туда и обратно). Исходными данными являются: расстояние, цена бензина, потребление бензина.

*Задача13.*

Написать программу, которая определяет доход по вкладу и сумму в конце срока хранения. Исходными данными являются: сумма, срок и процентная ставка.

*Задача14.*

Написать программу, которая вычисляет по закону Ома ток, напряжение и сопротивление. Исходными данными являются: сопротивление и напряжение, сопротивление и ток , ток и напряжение.

*Отчет должен включать*:

- название работы и ее цель;
- описания задач или схемы алгоритмов, для которых разрабатываются тесты;
- наборы тестов для каждой из заданных стратегий с пояснениями;
- выводы о том, в каких случаях должен использоваться тот или иной метод и стратегия в целом.

**Литература**

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.

**Приложение**

**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?