Лабораторная работа №1 Тема: Тестирование программных продуктов

Цель работы: изучить методы тестирования логики программы, формализованные описания результатов тестирования и стандарты по составлению схем программ.

Порядок выполнения работы:

1. Изучить теоретические сведения и разобрать примеры составления блок-схем задач.
2. Разработать схему алгоритма. Обозначить буквами или цифрами ветви алгоритма.
3. Спроектировать тесты по принципу «белого ящика» методами покрытия операторов, решений, условий, решений/условий и комбинаторного покрытия условий. Если разные методы покрываются одним и тем же набором тестов использовать один набор.
4. Выписать пути алгоритма, которые должны быть проверены тестами для каждого метода.
5. Внести в алгоритм несколько ошибок.
6. Протестировать разработанную Вами программу. Результаты оформить в виде таблиц (см. таблицы 1 - 4).
7. Проверить все виды тестов и сделать выводы об их эффективности.
8. Оформить отчет по лабораторной работе.
9. Сдать и защитить работу.

Защита отчета по лабораторной работе:

Отчет по лабораторной работе должен включать в себя:

1. Постановку задачи.
2. Блок-схемы программ.
3. Программу на языке C#, C++, Python (на выбор)
4. Тесты.
5. Таблицы тестирования программы.

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

Контрольные вопросы

1. Этап реализации и тестирования программного продукта.
2. Виды тестирования.
3. Критерии выбора тестов.
4. Свойства тестов.
5. Критерии надежности программ.
6. Оценка надежности программ.

   Задача.

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

Теоретические сведения:

Тестирование программного обеспечения

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

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

1. Виды тестов

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

Первый состоит в том, что тесты проектируются на основе внешних спецификаций программ и модулей, либо спецификаций сопряжения модуля

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

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

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

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

Например, если для определения равенства 3 чисел программируется выражение вида:

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

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

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

2. Стратегия «белого ящика»

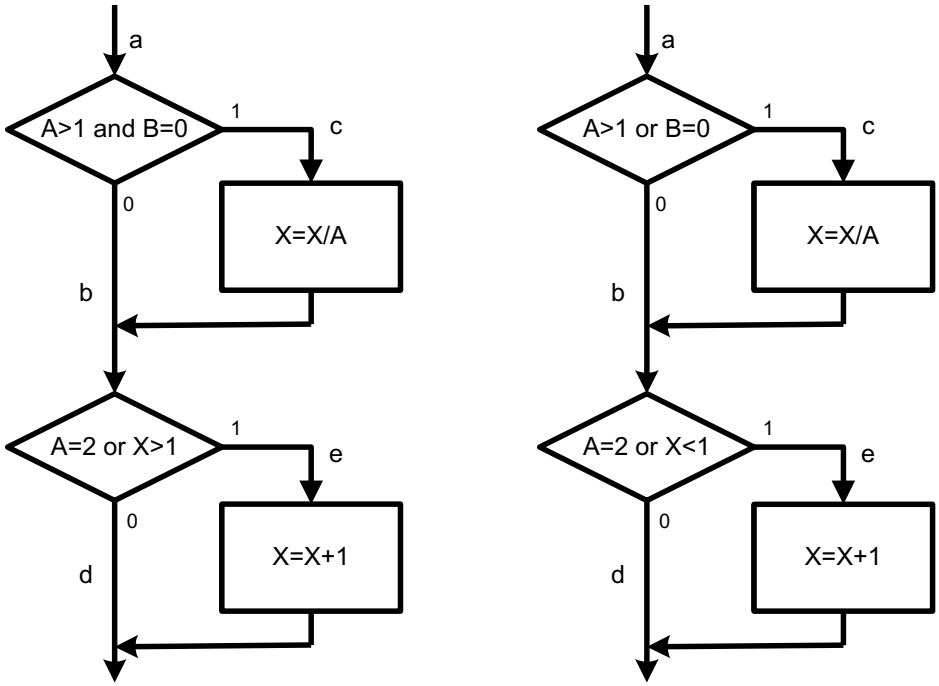
Существуют следующие методы тестирования по принципу «белого ящика»:

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

2.1. Метод покрытия операторов

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

Пример:



а б

Рис. 1. Пример алгоритма программы

а – правильный; б – с ошибкой

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

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

Таблица 1

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A=2, B=0, X=3	X=2,5	X=2,5	неуспешно

2.2. Метод покрытия решений (покрытия переходов)

Согласно методу покрытия решений каждое направление перехода должно быть реализовано, по крайней мере, один раз. Этот метод включает в себя критерий покрытия операторов, т.к. при выполнении всех направлений переходов выполнятся все операторы, находящиеся на этих направлениях.

Для программы приведенной на рис. 1 покрытие решений может быть выполнено двумя тестами, покрывающими пути {ace, abd}, либо {acd, abe}. Для этого выберем следующие исходные данные: {A=3, B=0, X=3} – в первом случае и {A=2, B=1, X=1} – во втором. Однако путь, где X не меняется, будет проверен с вероятностью 50%: если во втором условии вместо условия X > 1 записано X < 1, то ошибка не будет обнаружена двумя тестами.

Результаты тестирования приведены в табл. 2.

Таблица 2

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

2.3. Метод покрытия условий

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

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

а) A=2, B=0, X=4 *ace*

б) A=1, B=1, X=0 *abd*

Результаты тестирования приведены в табл. 3.

Таблица 3

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

2.4. Метод покрытия решений/условий

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

Недостатки метода:

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

Так в рассматриваемом примере два теста метода покрытия условий

а) A=2, B=0, X=4 *ace*

б) A=1, B=1, X=0 *abd*

отвечают и критерию покрытия решений/условий. Это является следствием того, что одни условия приведенных решений скрывают другие условия в этих решениях. Так, если условие A>1 будет ложным, транслятор может не проверять условия B=0, поскольку при любом результате условия B=0, результат решения ((A>1)&(B=0)) примет значение *ложь.* Т.е. в варианте на рис. 1 не все результаты всех условий выполнятся в процессе тестирования.

Рассмотрим реализацию того же примера на рис. 2. Наиболее полное покрытие тестами в этом случае выполняется так, чтобы выполнялись все возможные результаты каждого простого решения. Для этого нужно покрыть

пути *aceg* (тест A=2,B=0,X=4), *acdfh* (тест A=3, B=1, X=0), *abfh* (тест A=0, B=0, X=0), *abfi* (тест A=0, B=0, X=2)..

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

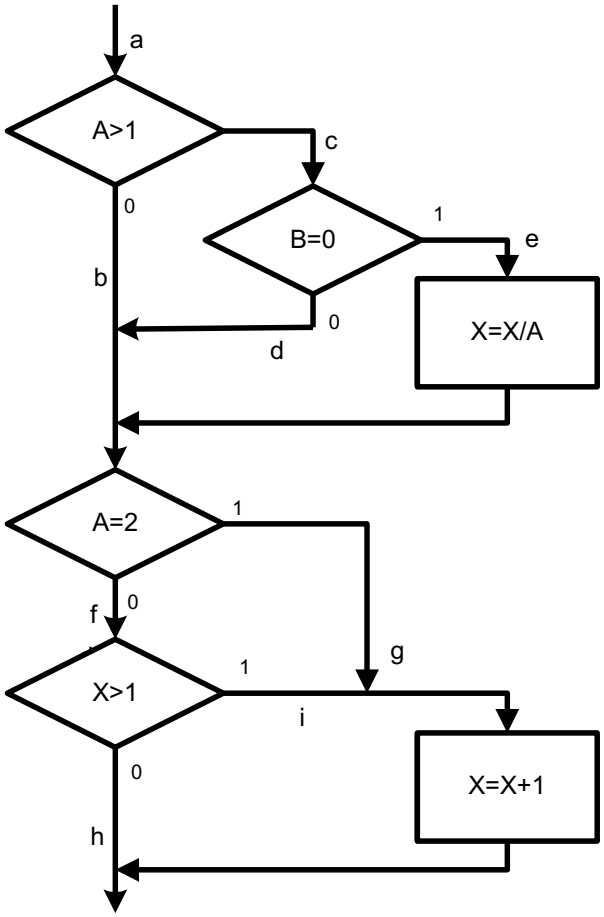


Рис. 2. Пример алгоритма программы

2.5. Метод комбинаторного покрытия условий

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

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

1. A>1, B=0.
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.

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

- A=2, B=0, X=4 {покрывает 1, 5};
- A=2, B=1, X=1 {покрывает 2, 6};
- A=0,5, B=0, X=2 {покрывает 3, 7};
- A=1, B=0, X=1 {покрывает 4, 8}.

Таблица 4

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A=2, B=0, X=4	X=3	X=3	неуспешно
A=2, B=1, X=1	X=2	X=1,5	успешно
A=0,5 B=0, X=2	X=3	X=4	успешно
A=1, B=0, X=1	X=1	X=1	неуспешно