Лекция 4. Тестирование методами «белого» и «черного» ящика

Тестирование методами «белого ящика»

Методы тестирования по принципу «белого ящика»:

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

1. Метод покрытия операторов

Цель метода – выполнение каждого оператора программы хотя бы один раз.

Пример

```
void func (int a, int b, float x)
{
    if ((a > 1) && (b == 0)) x = x/a;
    if (a == 2 || x > 1) x++;
}
```

$$a = 2\ b = 0\ x = 3$$

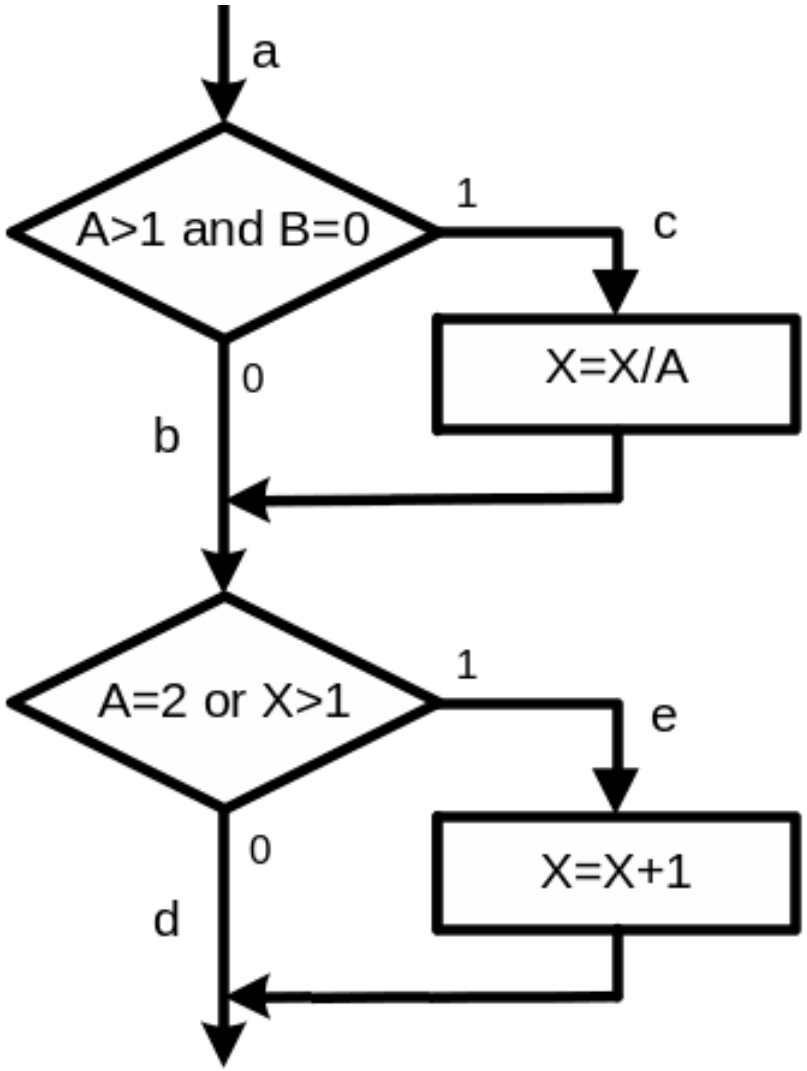

Рис. 1.а

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).

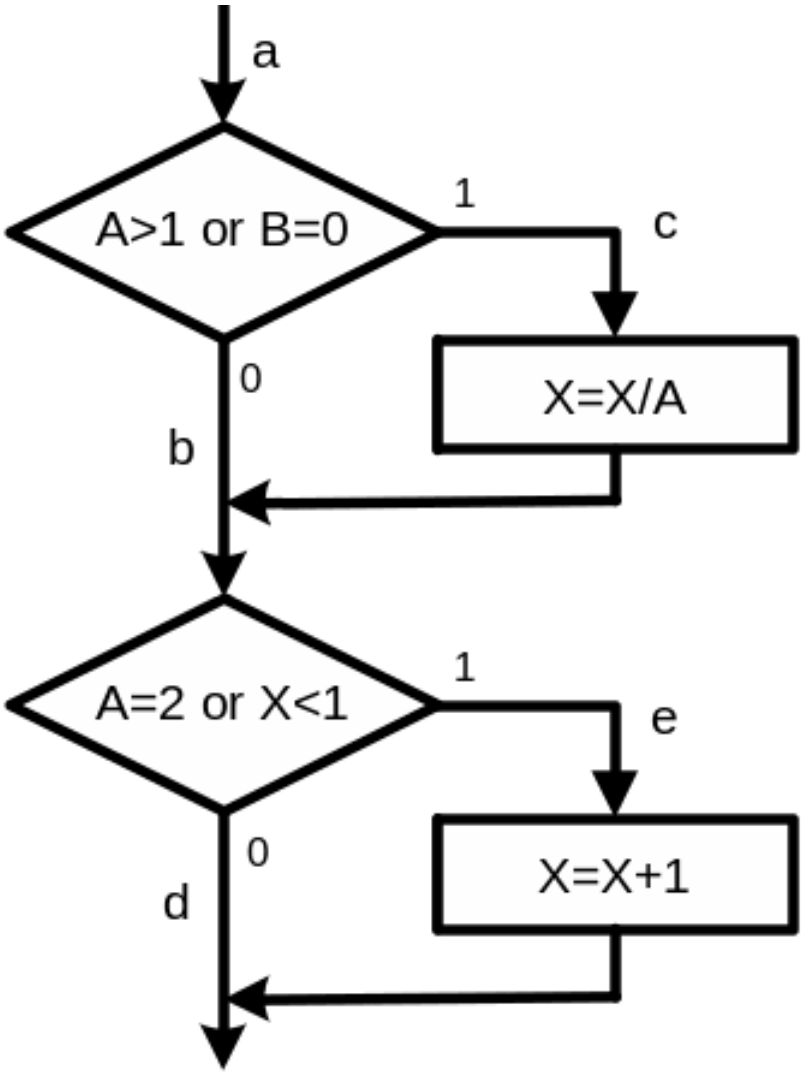

Рис. 1.б

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.

Кроме того
путь *abd* вообще не будет охвачен тестом A=2, B=0, X=3 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.

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

2. Метод покрытия решений (покрытия переходов)

Каждое направление перехода должно быть реализовано, по крайней мере, один раз.

Необходимо составить такое число тестов, при которых каждое условие в программе примет как истинное, так и ложное значение.

```
void func(int a, int b, float x) {
        if ((a (1) > 1) && (b (2) == 0)) x = x/a;
        if (a (3) == 2 || x (4) > 1) x++;
}
```

a=2, b=0, x=3
a=0, b=1, x=1.

Для программы (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.

Если во втором условии вместо условия X > 1 записано X < 1, то ошибка не будет обнаружена двумя тестами.

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

3. Метод покрытия условий

В соответствии с методом покрытия условий записывается число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении выполнялись, по крайней [illegible]раз.

```
void func(int a, int b, float x) {
        (1)              (2)
    if ((a > 1) && (b == 0)) x = x/a;
        (3)          (4)
    if (a == 2 || x > 1) x++;

}
```

(1)	(2)	(3)	(4)
1	1	1	1
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
0	0	0	0

В рассматриваемом примере 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*

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

4. Метод 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.

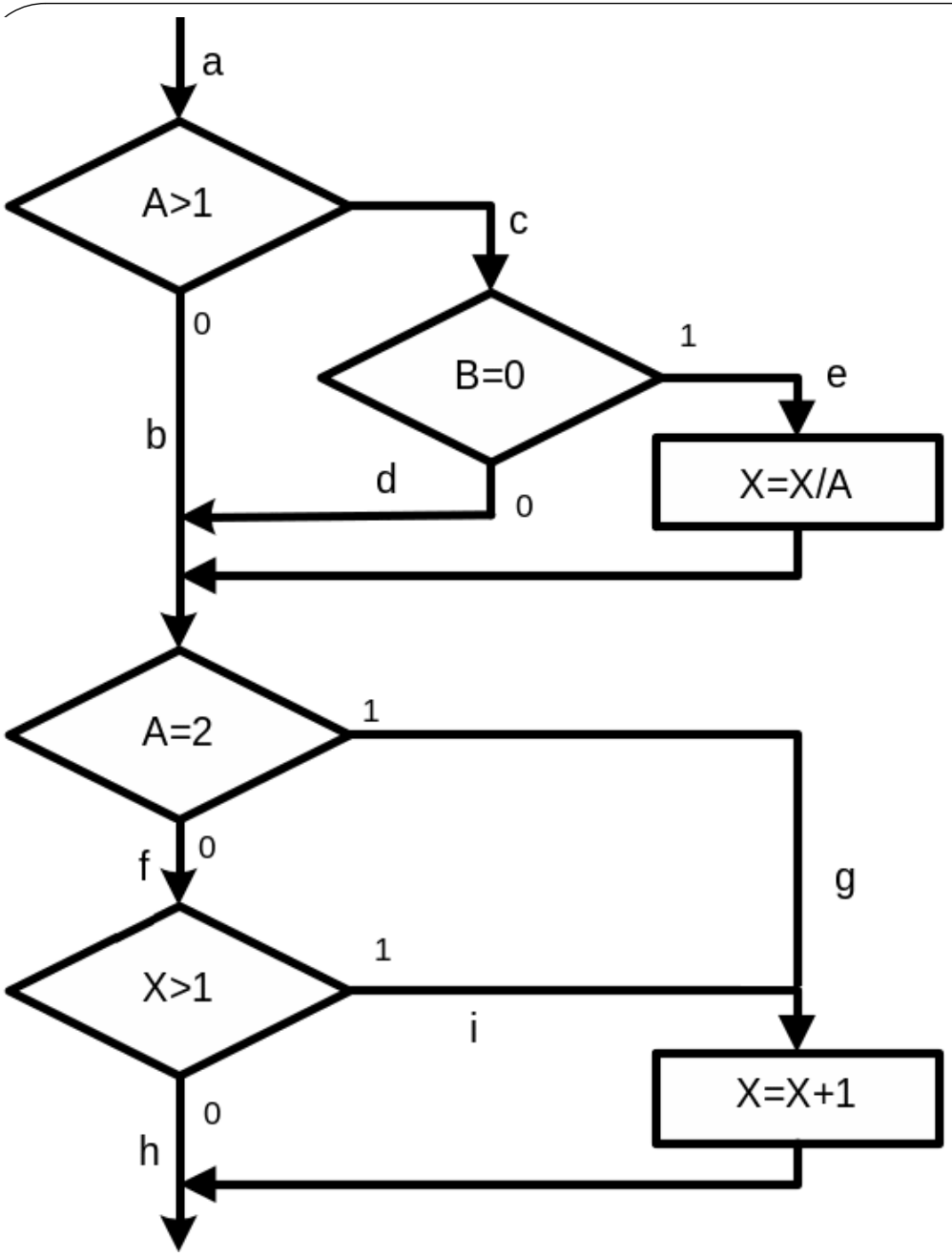

Рис. 2.

Другая реализация того же примера (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}.

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:

- $A>1, B=0$
- $A>1, B\neq 0$
- $A\leq 1, B=0$
- $A\leq 1, B\neq 0$
- $A=2, X>1$
- $A=2, X\leq 1$
- $A\neq 2, X>1$
- $A\neq 2, X\leq 1$

Чтобы протестировать эти комбинации, необязательно использовать все 8 тестов. Фактически они могут быть покрыты четырьмя:

- 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}.

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

Тестирование методами «черного ящика»

Методы формирования тестовых наборов:

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

1. Эквивалентное разбиение

- 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 (используется для разработки набора «интересных» условий, которые должны быть протестированы);
- каждый тест должен включать по возможности максимальное количество различных входных условий, что позволяет минимизировать общее число необходимых тестов (используется для разработки минимального набора тестов).

Этапы разработки тестов методом эквивалентного разбиения:

- 1.1. Выделение классов эквивалентности.
- 1.2. Построение тестов.

1.1. 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.

Правила выделение классов эквивалентности:

- если входные условия описывают *область* значений (например, «целое данное может принимать значения от 1 до 99», то выделяют один правильный класс 1<X<99 и два неправильных X<1 и X>99);
- 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 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 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 есть любое основание считать, что различные элементы класса эквивалентности трактуются программой неодинаково, то данный класс разбивается на меньшие классы эквивалентности.

1.2. 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.

2. Анализ граничных значений

Граничные условия – ситуации, возникающие на, выше или ниже границ входных классов эквивалентности.

Отличие анализа граничных значений от эквивалентного разбиения:

- выбор любого элемента в классе эквивалентности в качестве представительного при анализе граничных условий осуществляется так, чтобы проверить тестом каждую границу этого класса;
- при разработке тестов рассматриваются не только входные условия (*пространство входов*), но и *пространство результатов.*

Общие правила метода:

1) 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);
2) 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 25 записей, то проверить 0, 1, 25 и 26 записей;

3) использовать правило 1 для каждого выходного условия. Важно проверить границы пространства результатов, поскольку не всегда границы входных областей представляют такой же набор условий, как и границы выходных областей. Не всегда можно получить результат вне выходной области, но стоит рассмотреть эту возможность;
4) использовать правило 2 для каждого выходного условия;
5) если вход или выход программы есть упорядоченное множество (например, линейный список, таблица), то сосредоточить внимание на первом и последнем элементах этого множества;
6) попробовать свои силы в поиске других граничных условий.

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.

Этапы построения тестов:

1) спецификация разбивается на «рабочие» участки, т.к. таблицы причинно-следственных связей становятся громоздкими при применении метода к большим спецификациям.
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 содержания спецификации строится таблица истинности, в которой последовательно перебираются все возможные комбинации причин и определяются следствия каждой комбинации причин. Аналогично, при необходимости строится таблица истинности для класса эквивалентности.

Примечание:

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

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

Недостаток метода заключается в том, что он неадекватно исследует граничные условия.

4. Предположение об ошибке

Часто программист с большим опытом выискивает ошибки «без всяких методов», подсознательно используя метод «предположение об ошибке».

Процедура метода предположения об ошибке в значительной степени основана на интуиции.

Основная идея метода – 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

Выполнить тестирование программы, определяющей точку пересечения двух прямых на плоскости. Также она должна определять параллельность прямой одной их осей координат.

В основе программы лежит решение системы линейных уравнений:

$Ax + By = C$ и $Dx + Ey = F$.

1. Метод эквивалентных разбиений. 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. Метод 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:

- результат – единственное решение ($\Delta \neq 0$), Δ – главный определитель матрицы системы;
- результат – множество решений ($\Delta = 0$ и $\Delta_x = \Delta_y = 0$), Δ_x, Δ_y – определители вспомогательных матиц;
- результат – отсутствие решений ($\Delta = 0$, но $\Delta_x \neq 0$ или $\Delta_y \neq 0$).

и с результатами на границе:

- $\Delta = 0{,}01$;
- $\Delta = -0{,}01$;
- $\Delta = 0$, $\Delta_x = 0{,}01$, $\Delta_y = 0$;
- $\Delta = 0$, $\Delta_y = -0{,}01$, $\Delta_x = 0$.

3. Метод 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 и второй прямой, 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 с коэффициентами D, E, F.

Определение точки пересечения.

$\Delta = 0$	$\Delta_x = 0$	$\Delta_y = 0$	Единствен-ное решение	Множество решений	Решений нет
0	X	X	1	0	0
1	0	X	0	0	1
1	X	0	0	0	1
1	1	1	0	1	0

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

В результате к уже имеющимся тестам добавляются:

- 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;
- 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. Метод предположения об ошибке. Добавим еще один тест: все коэффициенты нули.

Всего получили 20 тестов по всем четырем методикам.