Министерство образования Республики Беларусь

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

**Факультет прикладной математики и информатики**

Буяк Роман Дмитриевич

Тестирование, отладка и верификация программ. Тестирование путем покрытия логики программы (“белый ящик”, “серый ящик”).

Реферат студента 1 курса 2 группы

**Преподаватель:**

Баранов Геннадий Аркадьевич

Минск 2017

**РЕФЕРАТ**

Реферат, 10с., 1 табл., 2 схемы, 3 источника.

**Ключевые слова:**

**Объект исследования** –

**Цель работы** –

**Методы исследования** –

**Область применения** –

**ВВЕДЕНИЕ**

**СОДЕРЖАНИЕ**

* Об особенностях тестирования методом “белого ящика”………………….4
* Покрытие операторов…………………………………………………………6
* Покрытие решений (переходов)……………………………………………...7
* Покрытие условий………………………………………………………….....8
* Покрытие решений/условий………………………………………………….9
* Комбинаторное покрытие условий…………………………………………..9
* Критерии С1 и С2……………………………………………………………..10

**Об особенностях тестирования методом “белого ящика”**

Технология тестирования и отладки программ, рассматриваемая в данном реферате, называется тестированием "белого ящика" — иногда ее еще называют тестированием "стеклянного ящика" — в противоположность классическому понятию "черного ящика". При тестировании "черного ящика" программа рассматривается как объект, внутренняя структура которого неизвестна. Тестировщик вводит данные и анализирует результат, но, как именно работает программа, он не знает. Подбирая тесты, специалист ищет интересные с его точки зрения входные данные и условия, которые могут привести к нестандартным ре­зультатам. Интересны для него прежде всего те представители каждого класса входных данных, на которых с наибольшей вероятностью могут проявиться ошибки тестируемой программы.

При тестировании "белого ящика" ситуация совершенно иная. Тестировщик (как правило, это программист) разрабатывает тесты, основы­ваясь на знании исходного кода, к которому он имеет полный доступ. В результате он получает следующие преимущества:

* **Направленность тестирования**. Программист может тестировать программу по частям, разработать специальные тестовые подпрограммы, которые вызывают тестируемый модуль и передают ему ин­тересующие программиста данные. Отдельный модуль гораздо легче протестировать именно как "белый ящик".
* **Полный охват кода**. Программист всегда может определить, какие именно фрагменты кода работают в каждом тесте. Он видит, какие еще ветви кода остались непротестированными и может подобрать условия, в которых они будут выполнены.
* **Управление потоком**. Программист всегда знает, какая функция должна выполняться в программе следующей и каким должно быть ее текущее состояние. Чтобы выяснить, работает ли программа так, как он думает, программист может включить в нее отладочные ко­манды, отображающие информацию о ходе ее выполнения, или воспользоваться для этого специальным программным средством, называемым отладчиком. Отладчик может отслеживать и менять последовательность выполне­ния команд программы, показывать содержимое ее переменных и их адреса в памяти и многое другое.
* **Отслеживание целостности данных.** Программисту известно, какая часть программы должна изменять каждый элемент данных. Отсле­живая состояние данных он может выявить такие ошибки, как изменение данных не теми модулями, их неверная интерпретация или неудачная организация.
* **Внутренние граничные точки**. В исходном коде видны те граничные точки программы, которые скрыты от взгляда "извне". Типичным примером может быть проблема переполнения буфера, используемого для временного хранения входных данных. Программист сразу может сказать, при каком количестве данных буфер переполнится, и ему не нужно при этом проводить тысячи тестов.

Тестирование "белого ящика" рассматривается как часть про­цесса программирования. Программисты выполняют эту работу постоянно, тестируя каждый модуль после его написания, а затем еще раз пос­ле интеграции его в систему. Рассмотрим ниже различные методы тестирования “белого ящика”.

**Покрытие операторов**

Заметим, что выполнение каждого компонента программы хотя бы один раз является критерием покрытия программы. Однако, это слабый критерий, так как выполнение каждого оператора программы по крайней мере один раз есть необходимое, но не достаточное условие для приемлемого тестирования по принципу “белого ящика”. Рассмотрим следующую схему, описывающую логику некоторой части программы:



Код эквивалентной части программы на языке C++ выглядит следующим образом:

if (A>1 && B==0)

{

X=X/A;

}

if (A==2 || X>1)

{

X=X+1;

}

Можем выполнить каждый оператор программы, выполнив один тест, реализовав путь ace, например при значениях A=2 B=0 X=3. К сожалению, данный критерий хуже, чем кажется на первый взгляд. Например, если первый условный оператор будет связан логическим *или*, а не *и,* то при тестировании по данному критерию ошибка не будет обнаружена. В случае, когда во втором условном операторе будет записано X>0, такая ошибка также не будет найдена. Таким образом, критерий покрытия операторов является настолько слабым, что обычно не используется.

**Покрытие решений (условий)**

Более сильный критерий покрытия логики программы известен как *покрытие решений,* или *покрытие переходов.*Согласно данному критерию должно быть записано дос­таточное число тестов, такое, что каждое решение на этих тестах примет значение *истина* и *ложь,* по крайней мере, один раз. Иными словами, каждое направление перехода должно быть реализовано, по крайней мере, один раз. Примером оператора перехода может служить условный оператор **if.**

Можно показать, что покрытие решений обычно удовлетворяет критерию покрытия операторов. Посколь­ку каждый оператор лежит на некотором пути, исходящем либо из оператора перехода, либо из точки входа программы, при выполнении каждого направления перехода каждый оператор должен быть выполнен. Однако существует, по крайней мере три исключения. Первое— патологическая ситуация, когда программа не имеет ре­шений. Второе встречается в программах или подпрограммах с несколькими точками входа; данный оператор мо­жет быть выполнен только в том случае, если выполнение программы начинается с соответствующей точки вхо­да. Третье исключение—операторы внутри условных блоков: выполнение каждого направления перехода не обязатель­но будет вызывать выполнение всех операторов внутри блока. Так как покрытие операторов считается необходимым условием, покрытие решений, которое представляется более сильным критерием, должно включать покрытие операторов. Следовательно, покрытие решений требует, чтобы каждое решение имело результатом значения *истина* и *ложь,*и при этом каждый оператор выполнялся бы, по крайней мере, один раз. Альтернативный и более легкий способ выражения этого требования состоит в том, чтобы каждое решение имело результатом значения *истина* и *ложь*и что каждой точке входа должно быть передано управление при вызове программы, по крайней мере, один раз.

Покрытие решений — более сильный критерий, чем по­крытие операторов, но н он имеет свои недостатки. Например, путь, где *Х* не изменяется (если выбрано первое альтернативное покрытие), будет проверен с вероятностью 50%. Если во втором решении существует ошибка (например, *Х < 1* вместо *Х > 1),* то ошибка не будет обнаружена двумя тестами предыдущего примера.

**Покрытие условий**

Лучшим критерием по сравнению с предыдущим, является *покрытие условий.* В этом случае записывают число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении выполнялись по крайней мере один раз. Поскольку, как и при покрытии решений, это покрытие не всегда приводит к выполнению каждого оператора, к критерию требуется дополнение, которое заключается в том, что каждой точке входа в программу или подпрограмму должно быть передано управление при вызове по крайней мере один раз.

Заметим, что покрытие условий обычно лучше покрытия решений, поскольку оно *может* (но не всегда) вызвать выполнение решений в условиях, не реа­лизуемых при покрытии решений.

Хотя применение критерия покрытия условий на первый взгляд удовлетворяет критерию покрытия решений, это не всегда так. Если тестируется решение IF (А&В), то при критерии покрытия условий требовались бы два теста — А есть *истина,* В есть *ложь* и А есть *ложь,* В есть *истина.* Но в этом случае не выполнялось бы предложение оператора IF. Тесты критерия покрытия ус­ловий для ранее рассмотренного примера покрывают результаты всех решений, но это только случайное совпадение. Например, для следующей схемы два теста 

Рис. 2

*1. A**=1, B**=0**,**X=3*

*2. A=2, B=1, X=1*

Покрывают результаты всех условий, но только два из четырех результатов решений

**Покрытие решений/условий**

Очевидным следствием из дилеммы прошлого пункта является критерий, названный *покрытие**м решений/условий.* Он требует такого достаточного набора тестов, чтобы все возможные резуль­таты каждого усло­вия в решении выполнялись по крайней мере один раз, все результаты каж­дого решения выполнялись по крайней мере один раз и каждой точке входа пе­редавалось управление по крайней мере один раз.

Недостатком критерия покрытия решений/условий является невозможность его применения для выполнения всех результатов всех условий; часто подобное выполнение имеет место вследствие того, что определенные условия скрыты другими условиями. Недостаток обуславливается тем, что, как показано на рис. 2, результаты условий в выражениях *и* и *или* мо­гут скрывать и блокировать действие других условий. На­пример, если условие *и* есть *ложь,* то никакое из после­дующих условий в выражении не будет выполнено. Ана­логично если условие *или* есть *истина,* то никакое из по­следующих условий не будет выполнено. Следовательно, критерии покрытия условий и покрытия решений/условий недостаточно чувствительны к ошибкам в логических вы­ражениях.

**Комбинаторное покрытие условий**

Критерием, который решает вышеизложенные проблемы а также некоторые другие, является *комбинаторное покр**ытие условий.*Он требует создания такого числа тестов, чтобы все воз­можные комбинации результатов условия в каждом ре­шении и все точки входа выполнялись по крайней мере один раз. Набор тестов, удовлетворяющий критерию комбинаторного покрытия условий, будет удовлетворять также и критериям покрытия решений, покрытия условий и покрытия решений/условий.

Для программ, содержащих только одно условие на каждое решение, минимальным являет­ся критерий, набор тестов которого

1) вызывает выпол­нение всех результатов каждого решения по крайней ме­ре один раз

2) передает управление каждой точке вхо­да по крайней мере один раз (чтобы обеспечить выполнение каждого опера­тора программы 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 программы по крайней мере один раз. (Слово «возможных» упот­реблено здесь потому, что некоторые комбинации условий могут быть нереализуемыми; например, в выражении (А>2)&&(A<10) могут быть реализованы только три комбинации условий.)

Ни один из критериев покрытия логики программы, как методом “черного ящика”, так и методом “белого ящика”, не является универсальным и имеет свои недостатки. Поэтому оптимальной стратегией считается комбинирование различных методов. Симбиозом методов черного и белого ящиков является метод “серого ящика”, сочетающего элементы двух предыдущих подходов:

• с одной стороны, тестирование, ориентированное на пользователя, а значит, мы используем шаблоны поведения пользователя, т.е. применяем методику "черного ящика".

• с другой — информированное тестирование, т.е. мы знаем, как устроена хотя бы часть тестируемой программы, и активно используем это знание.

**Критерии С1 и С2**

Несмотря на старания разработчиков, ошибки при создании программ все же случаются. Для определения серьезности ошибки, они распределяются по специальным категориям, от C1 до С4

Серьезность ошибки — это степень воздействия ошибки (magnitude of impact) на ПО, исходя из принадлежности ошибки к определенной технической категории.

Вот пример категоризации:

|  |  |
| --- | --- |
| Серьезность ошибки | Определение |
| С1-Критический | • критический системный сбой (crash);  • потеря данных (data loss);  • проблема с безопасностью (security issue) |
| С2-Значительный | • программа "зависает" (hang);  • ошибка блокирует кодирование, тестирование или использование программы (blocker) |
| С3-Умеренный | • функциональные проблемы (functional bugs) |
| С4-Косметический | • косметическая проблема (cosmetic problem) |

# **ЛИТЕРАТУРА**

1. С. Канер, Д. Фолк, Е. Нгуен. Тестирование программного обеспечения. — К.: Диасофт, 2000. — 544 с.
2. 2. Г. Майерс. Искусство тестирования программ. - М.: Финансы и статистика, 1982.
3. Савин Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах. — М.: Дело, 2007. — 312 с.