МОДУЛЬ 8: «ПОКРЫТИЕ ПРОГРАММНОГО КОДА»

ПОНЯТИЕ ПОКРЫТИЯ ПРОГРАММНОГО КОДА

Одна из оценок качества системы тестов – это ее полнота, т.е. величина той части функциональности системы, которая проверяется тестовыми примерами. Обычно за меру полноты берут отношение объема проверенной части системы к ее объему в целом. Полная система тестов позволяет утверждать, что система реализует: а) всю функциональность, указанную в требованиях; б) не реализует никакой другой функциональности.

Один из используемых методов определения полноты системы тестов является определение отношения количества тест-требований, для которых существуют тестовые примеры, к общему количеству тест-требований (покрытие тест-требований тестовыми примерами).

Для детальной оценки полноты системы тестов при тестировании стеклянного ящика анализируется покрытие программного кода, называемое также структурным покрытием. К анализу покрытия программы можно приступать только после полного покрытия требований. Полное покрытие программного кода не гарантирует того, что тесты проверяют все требования к системе.

УРОВНИ ПОКРЫТИЯ

ПО СТРОКАМ ПРОГРАММНОГО КОДА (STATEMENT COVERAGE). Для обеспечения полного покрытия программного кода на этом уровне, необходимо, чтобы в результате выполнения тестов каждый оператор был выполнен хотя бы один раз.

Особенность этого уровня покрытия состоит в том, что на нем затруднен анализ покрытия некоторых управляющих структур. Например, для полного покрытия всех строк следующего участка программного кода на языке С достаточно одного тестового примера [Синицын 2006]:

```
Вход: condition = true; Ожидаемый выход: *p = 123.
int* p = NULL;
if (condition)
   p = &variable;
*p = 123;
```

Даже если в состав тестов не будет входить тестовый пример, проверяющий работу фрагмента при значении condition=false, код будет покрыт. Однако в случае condition=false выполнение фрагмента вызовет ошибку.

Аналогичные проблемы возникают при проверке циклов do … while, при данном уровне покрытия достаточно выполнение цикла только один раз, при этом метод совершенно нечувствителен к логическим операторам || и &&.

Другой особенностью метода является зависимость уровня покрытия от структуры программы. На практике часто не требуется 100% покрытия программы, вместо этого устанавливается допустимый уровень покрытия, например 75%. Проблемы могут возникнуть при покрытии следующего фрагмента программы:

```
if (condition)
```

```
functionA();
else
functionB();
```

Если functionA() содержит 99 операторов, а functionB() один оператор, то одного тестового примера, устанавливающего condition в true, будет достаточно для достижения необходимого уровня покрытия. При этом один тестовый пример, устанавливающий значение condition в false даст очень низкий уровень покрытия.

ПО ВЕТКАМ УСЛОВНЫХ ОПЕРАТОРОВ (DECISION COVERAGE). Для обеспечения полного покрытия по этому методу каждая точка входа и выхода в программе и во всех ее функциях должна быть выполнена, по крайней мере, один раз и все логические выражения в программе должны принять каждое из возможных значений хотя бы один раз. Таким образом, для покрытия по веткам требуется как минимум два тестовых примера.

В отличие от предыдущего уровня покрытия этот метод учитывает покрытие условных операторов с пустыми ветками. Так, для покрытия по веткам участка программного кода [Синицын 2006].

```
a = 0;
if (condition) {
  a = 1;
}
```

необходимы два тестовых примера:

```
1. Вход: condition = true; Ожидаемый выход: a = 1;
2. Вход: condition = false; Ожидаемый выход: a = 0;
```

Особенность данного уровня покрытия заключается в том, что на нем не учитываются логические выражения, значения компонент которых получаются вызовом функций. Например, на фрагменте программного кода [Синицын 2006].

```
if ( condition1 && ( condition2 || function1() ) )
   statement1;
else
   statement2;
```

полное покрытие по веткам может быть достигнуто при помощи двух тестов:

```
1. Вход: condition1 = true, condition2 = true
2. Вход: condition1 = false, condition2 = true/false (любое значение)
```

В обоих случаях не происходит вызова функции function1(), хотя покрытие данного участка кода будет полным. Для проверки вызова функции function1() необходимо добавить еще один тестовый пример (который, однако, не улучшает степени покрытия по веткам):

> *3. Вход: condition1 = true, condition2 = false.*

Для более полного анализа компонент условий в логических операторах существует несколько методов, учитывающих структуру компонент условий и значения, которые они принимают при выполнении тестовых примеров.

ПОКРЫТИЕ ПО УСЛОВИЯМ (CONDITION COVERAGE). Для полного покрытия по этому методу каждая компонента логического условия в результате выполнения тестовых примеров должна принимать все возможные значения, но при этом не требуется, чтобы само логическое условие принимало все возможные значения. Так, например, при тестировании следующего фрагмента [Синицын 2006]:

```
if (condition1 | condition2)
  functionA();
else
  functionB();
```

Для покрытия по условиям потребуется два тестовых примера:

> *Вход: condition1 = true, condition2 = false*
>
> *Вход: condition1 = false, condition1 = true.*

При этом значение логического условия будет принимать значение только true, таким образом, при полном покрытии по условиям не будет достигаться покрытие по веткам.

ПОКРЫТИЕ ПО ВЕТКАМ/УСЛОВИЯМ (CONDITION/DECISION COVERAGE). Этот метод сочетает требования предыдущих двух методов – для обеспечения полного покрытия необходимо, чтобы как логическое условие, так и каждая его компонента приняла все возможные значения. Для покрытия рассмотренного выше фрагмента с условием condition1 | condition2 потребуется 2 тестовых примера:

> *Вход: condition1 = true, condition2 = true*
>
> *Вход: condition1 = false, condition2 = false.*

Однако эти два тестовых примера не позволят протестировать правильность логической функции – вместо OR в программном коде могла быть ошибочно записана операция AND.

ПОКРЫТИЕ ПО ВСЕМ УСЛОВИЯМ (MULTIPLE CONDITION COVERAGE). Для выявления неверно заданных логических функций был предложен метод покрытия по всем условиям. При данном методе покрытия должны быть проверены все возможные наборы значений компонент логических условий. Т.е. в случае n компонент потребуется 2n тестовых примеров, каждый из которых проверяет один набор значений, Тесты, необходимые для полного покрытия по данному методу, дают полную таблицу истинности для логического выражения.

Несмотря на полноту системы тестов, обеспечивающей этот уровень покрытия, данный метод редко применяется на практике из-за его избыточности и сложности. Еще одним недостатком метода является

зависимость количества тестов от структуры логического выражения. Так, для условий, содержащих одинаковое количество компонент и логических операций [Синицын 2006]:

> *a && b && (c || (d && e))*
>
> *((a || b) && (c || d)) && e*

потребуется разное количество тестовых примеров. Для первого случая для полного покрытия нужно 6 тестов, для второго – 11.

УМЕНЬШЕНИЕ КОЛИЧЕСТВА ТЕСТОВЫХ ПРИМЕРОВ

Для уменьшения количества тестовых примеров при тестировании логических условий фирмой Boeing был разработан модифицированный метод покрытия по веткам/условиям (Modified Condition/Decision Coverage или MC/DC). Данный метод широко используется при верификации бортового авиационного программного обеспечения согласно процессам стандарта DO-178B.

Для обеспечения полного покрытия по этому методу необходимо выполнение следующих условий:

- ✓ каждое логическое условие должно принимать все возможные значения;
- ✓ каждая компонента логического условия должна хотя бы один раз принимать все возможные значения;
- ✓ должно быть показано независимое влияние каждой из компонент на значение логического условия, т.е. влияние при фиксированных значениях остальных компонент.

Покрытие по этой метрике требует достаточно большого количества тестов для того, чтобы проверить каждое условие, которое может повлиять на результат выражения, однако это количество значительно меньше, чем требуемое для метода покрытия по всем условиям.

АНАЛИЗ ПОЛНОТЫ ПОКРЫТИЯ КОДА

Целью анализа полноты покрытия кода является выявление участков кода, которые не выполняются при выполнении тестовых примеров. Тесты, основанные на требованиях, могут не обеспечивать полного выполнения всей структуры кода. Для улучшения покрытия проводится анализ полноты покрытия кода тестами и дополнительные проверки, направленные на выяснение причины недостаточного покрытия и определение необходимых действий по его устранению. Обычно анализ покрытия выполняется с учетом следующих соглашений:

- ✓ анализ должен подтвердить, что полнота покрытия тестами структуры кода соответствует требуемому виду покрытия и заданному минимально допустимому проценту покрытия;
- ✓ анализ полноты покрытия тестами структуры кода может быть выполнен с использованием исходного текста;
- ✓ анализ должен подтвердить правильность передачи данных и управления между компонентами кода.

Анализ полноты покрытия тестами может выявить часть исходного кода, которая не исполнялась в ходе тестирования. Для разрешения этого обстоятельства могут потребоваться дополнительные действия в процессе проверки программного обеспечения. Эта неисполняемая часть кода может быть результатом:

- ✓ недостатков в формировании тестовых примеров или тестовых процедур, основанных на требованиях;
- ✓ неадекватности в требованиях на программное обеспечение;

- ✓ «мертвый» код. Этот код должен быть удален и проведен анализ для оценки эффекта удаления и необходимости перепроверки;
- ✓ дезактивируемый код;
- ✓ избыточность условия. Логика работы такого условия должна быть пересмотрена.
- ✓ защитный код. Часть кода, используемая для предотвращения исключительных ситуаций, которые могут возникнуть в процессе работы программы.

ОТЧЕТЫ О ПОКРЫТИИ ПРОГРАММНОГО КОДА

Степень покрытия программного кода тестами – важный количественный показатель, позволяющий оценить качество системы тестов, а в некоторых случаях – и качество тестируемой программной системы. Данные о степени покрытия помещаются в отчеты о покрытии, генерируемые при выполнении тестов инструментальными средствами, поддерживающими процесс тестирования, т.е. по сути, генерируются средой тестирования (Рис.20 [Синицын 2006]).

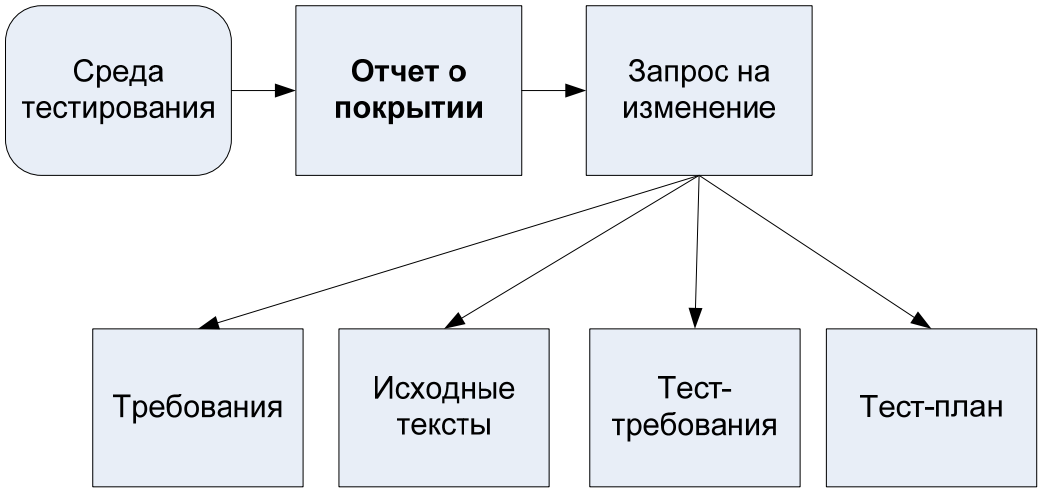

Рис.20 Генерация отчета о покрытии и изменения по результатам его анализа

Формат отчетов о покрытии един внутри проекта или нескольких проектов и зависит от особенностей инструментальных средств тестирования. В отчете о покрытии в стандартизированной форме указываются участки программного кода тестируемой системы (или ее части), которые не были выполнены во время выполнения тестов. По результатам анализа причин недостаточного покрытия составляются отчеты о проблемах и запросы на изменение – документы, в которых описывается объекты, которые необходимо изменить и причины этих изменений.

1. Недостаточное покрытие может свидетельствовать о неполноте системы тестов или тест-требований, в этом случае в запросе об изменении указывается на необходимость расширения системы тестов или тест-требований.
2. Другой причиной недостаточного покрытия могут быть участки защитного кода, которые никогда не выполнятся даже в случае нештатной работы системы. В этом случае в запросе на изменение указывается на необходимость модификации исходных текстов, либо отмечается, что для этого участка программной системы не требуется покрытие.
3. Третьей причиной недостаточного покрытия может быть рассогласование требований и программного кода системы, в результате которого в коде могут остаться неиспользуемые более участки, либо наоборот, появиться участки, рассчитанные на будущее. В этом случае в запросе на изменение указывается на необходимость модификации требований и/или кода системы для приведения их в согласованное состояние.

ВОЗМОЖНЫЕ ФОРМЫ ОТЧЕТОВ О ПОКРЫТИИ

Типичный отчет о покрытии представляет собой список структурных элементов покрываемого программного кода (функций или методов), содержащий для каждого структурного элемента следующую информацию:

- ✓ Название функции или метода;
- ✓ Тип покрытия (по строкам, по ветвям, MC/DC или иной);
- ✓ Количество покрываемых элементов в функции или методе (строк, ветвей, логических условий);
- ✓ Степень покрытия функции или метода (в процентах или в абсолютном выражении);
- ✓ Список непокрытых элементов (в виде участков непокрытого программного кода с номерами строк).

Кроме того, отчет о покрытии содержит заголовочную информацию, позволяющую идентифицировать отчет и общий итог – общую степень покрытия всех функций, для которых собирается информация о покрытии. Пример такого отчета о покрытии приведен ниже [Синицын 2006]:

Coverage Report

Generated 10/07/2006 for file Testing_Facilities.cpp

---

1) function main_Menu()

Coverage: Instructions

Elements: 25 structured lines of code (SLOCs)

Covered: 22 lines (88%)

Not covered:

291 default:

292 return -1;

293 break;

Coverage: Branches

Elements: 5 branches

Covered: 4 branches (80%)

Not covered (starting and ending lines only):

default:

break;

---

2) function item_Help()

Coverage: Instructions

Elements: 180 structured lines of code (SLOCs)

Covered: 180 lines (100%)

Coverage: Branches

Elements: 2 branches

Covered: 2 branches (80%)

Total functions: 2

Total instructions coverage: 98.5%

Total branches coverage: 86%

Отчет о покрытии может создаваться либо для всех функций программного модуля или всего проекта, либо выборочно для определенных функций. Если размер функций, для которых генерируется выборочный отчет, невелик, может применяться форма отчета о покрытии, в котором покрытый и непокрытый программный код выделяется различными цветами. Такая форма неприменима для покрытия ветвей и логических условий, но может применяться для покрытия по строкам. Пример такого отчета приведен ниже [Синицын 2006]:

Coverage report for BaseCalculator.AnalaizerClass.Format method.

Generated on 25/07/2006

```
public static string Format(
{
  string formstr = "";
  string prev = "";
  if (expression.Length <= 65536) {
            for (int i = 0; i < expression.Length; i++) {
                  switch (expression[i]) {
                        case '0': {
                              if (prev == "число" || prev == "") {
                                    formstr += expression[i].ToString();
                              } else {
                                    formstr += " " + expression[i].ToString();
                              }
                              prev = "число";
                              break;
```

```
                    }
                }
            }
        } else {
            MessageBox.Show("Слишком длинное выражение.");
            Program.res = 7;
            return "&Error 07";
        }
    }
```

Зеленым цветом отмечены выполненные в результате тестирования участки метода, красным – не выполненные.

Для примера рассмотрим отчет о структурном покрытии, генерируемый полученного при помощи средства для анализа программного кода CodeTEST компании Metrowerks [Синицын 2006]. Сбор покрытия производится по определенному уровню покрытия, информация об этом входит в состав заголовка отчета о покрытии. В данном случае программный код покрывался по MC/DC.

```
CodeTEST Advanced Coverage Tools

Modified Condition Decision Coverage Report (MCDC), DO-178B Level A

CodeTEST Report Generator 2.2.04

Report generated on Sun Aug 21 14:28:16 2005

IDB File:

  <Path_to_IDB_file>

  42963330-1C0 Thu May 26 21:36:00 2005
```

Далее выводится информация о проценте покрытия по всем модулям в совокупности (в данном случае, это 33 модуля).

```
Overall Summary (MCDC): 33 Files

 1405 true-false decisions

    674  48.0%  covered

    494  35.2%  partially covered

    237  16.9%  not covered

 1928 conditions in the decisions

   1068  55.4%  covered
```

```
    571  29.6%  partially covered
    289  15.0%  not covered
278 case branches
    154  55.4%  covered
    124  44.6%  not covered
581 coverage events
    504  86.7%  covered
     77  13.3%  not covered
```

Рассмотрим более подробно каждую из характеристик:

TRUE-FALSE DECISIONS – количество логических выражений, в данном случае их 1405. Из этого числа на оба возможных значения (True и False) покрыто лишь 674, еще 494 покрыто на какое-то одно из значений, а 237 не покрыто вообще.

CONDITIONS IN THE DECISIONS – количество логических условий внутри логических выражений. Их количество не может быть меньше количества логических выражений. Непокрытие логических условий явным образом влечет за собой непокрытие логических выражений. Если все условия будут покрыты, то все выражения также будут покрыты.

CASE BRANCHES – количество веток ветвления операторов выбора (select). Ветка считается покрытой, если выражение, стоящее в условии оператора select, принимает значение, соответствующее данному варианту выбора (case), в результате чего ветка выполняется.

COVERAGE EVENTS – количество точек входа/выхода в функциях. Точка входа в функцию считается покрытой, если эта функция была вызвана хотя бы один раз. Точками выхода являются точка окончания функции (}), когда управление передается в место вызова этой функции, а также операторы возврата (return), после выполнения которых управление также передается в место вызова функции.

Затем выводится информация для первого модуля, а именно: имя модуля, путь к файлу на диске, дата последнего изменения, контрольная сумма. После этого выводится суммарная информация о покрытии только для этого модуля (для всех его функций, количество которых также указывается).

После этого происходит поочередный вывод суммарной информации для каждой из функций в отдельности, а также всех ключевых участков функции, влияющих на покрытие, а именно: точки входа/выхода, логические выражения, логические условия, операторы выбора.

ПОКРЫТИЕ НА УРОВНЕ МАШИННЫХ КОДОВ

В некоторых случаях инструментальные средства сбора покрытия анализируют покрытие программного кода тестами не на уровне исходных текстов системы, а на уровне машинных инструкций. В этом случае степень покрытия зависит и от того, какой исполняемый код генерируется компилятором.

Сбор информации о покрытии на уровне исполняемого кода наиболее часто применяется в высоко-критичных программных системах, в которых не допускается наличия “мертвого” исполняемого кода, который потенциально может привести к сбою или отказу во время работы системы. К таким системам в

первую очередь можно отнести авиационные бортовые системы, медицинские системы и системы обеспечения безопасности информации.

ЛИТЕРАТУРА

[Синицын 2006] – Синицын С.В., Налютин Н.Ю. Верификация программного обеспечения. Курс лекций. Московский инженерно-физический институт. М. 2006.