МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное

Учреждение высшего образования

**«Гжельский государственный университет»** (ГГУ)

Колледж ГГУ

Специальность 09.02.07 Информационные системы и программирования

**Реферат**

**по дисциплине «МДК 01.01 Технология разработки программного обеспечения »**

**на тему « Тестовое покрытие. Оценка необходимого количества тестов»**

ВЫПОЛНИЛА:

Студентка группы ИСП-О-17

Янгабозова Анастасия

ПРОВЕРИЛА:

Прокуронова А.Ю.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

п. Электроизолятор

2019 г.

Самый лучший способ оценить, хорошо ли мы протестировали продукт – **проанализировать пропущенные дефекты**. По ним можно многое оценить: что мы проверили недостаточно тщательно, каким областям продукта стоит уделить больше внимания, какой вообще процент пропусков и какова динамика его изменений.

Сложность современного программного обеспечения и инфраструктуры сделало невыполнимой задачу проведения тестирования со 100% тестовым покрытием. Поэтому для разработки набора тестов, обеспечивающего более менее высокий уровень покрытия можно использовать специальные инструменты либо техники тест дизайна.

**Существуют следующие подходы к оценке и измерению тестового покрытия:**

1. [Покрытие требований](http://www.protesting.ru/testing/testcoverage.html#requirements) - оценка покрытия тестами функциональных и нефункциональных требований к продукту путем построения матриц трассировки .
2. [Покрытие кода](http://www.protesting.ru/testing/testcoverage.html#code)  - оценка покрытия исполняемого кода тестами, путем отслеживания непроверенных в процессе тестирования частей программного обеспечения.

[Тестовое покрытие на базе анализа потока управления](http://www.protesting.ru/testing/testcoverage.html#flow) - **оценка покрытия**, основанная на определении путей выполнения кода программного модуля и создания выполняемых тест кейсов для покрытия этих путей.

Метод покрытия требований сосредоточен на проверке соответствия набора проводимых тестов требованиям к продукту, в то время как анализ покрытия кода - на полноте проверки тестами, разработанной части продукта (исходного кода), а анализ потока управления - на прохождении путей в графе или модели выполнения тестируемых функций.

Перед тем как проводить работу по внедрению выбранной метрики, нужно понять, как вы будете ее использовать. Сначала ответьте именно на этот вопрос – и вы сразу поймете, как лучше всего поступить.

На практике есть два наиболее масштабных и действенных подхода к оценке степени покрытия продукта автоматическими тестами: методика покрытия установленных предписаний (другими словами, что-то схожее с проверкой черного ящика) и работа с программным кодом (процесс тестирования классического белого ящика).

**Покрытие требований** – уровень оценки покрытия продукта теста для проверки как производительных, так и нефункциональных требований, устанавливаемых перед разрабатываемым ПО путем создания матриц трассировки.

**Покрытие кода** – способ анализа покрытия разработанного кода тестовыми кейсами путем поиска непроверенных на стадии тестирования частей разрабатываемого программного продукта.

Деятельность покрытия тестами программного кода, бесспорно, является наиболее важной метрикой для максимального обеспечения качества при работе с тестируемой средой, особенно когда речь заходит о проверке продукта с очень сложной логикой или большим объемом написанного кода.

Работы для проверки кода, как правило, выполняются с помощью широкого перечня доступного функционального инструментария, благодаря которому можно качественно отследить, какие ветки программного кода были проверены, а какие остались незамеченными при проведении автотрестов.

На сегодняшний день существует сразу несколько типов измерения тестового покрытия, базовыми из которых являются:

1. Работа с операторами – каждая ли строчка программного кода была выполнена согласно ТЗ и максимально протестирована на работоспособность?
2. Плотность покрытия установленных условий – все ли решения и наработки были выполнены и протестированы?
3. Тестирование путей – все ли потенциально допустимые пути выполнения программного кода были проверены и максимально качественно протестированы?
4. Работа с функциями – все ли допустимые значения были проанализированы и выполнены?
5. Тестирование программных комбинаций – все ли заданные условия и технические комбинации были проверены и выполнены согласно установленной технологической документации?

Пример:

1 int gcd(int a, int b)

2 {

3 if(a == 0)

4 return b;

5 if(b == 0)

6 return a;

7 if(a > 0 && b < 0 || a < 0 && b > 0)

8 b = -b;

9

10 while(b != 0)

11 {

12 if(b > a && a > 0 || b < a && a < 0)

13 {

14 a = b-a;

15 b = b-a;

16 a = a+b;

17 }

18

19 b = a-b;

20 a = a-b;

21 }

22

23 return a;

24 }

Приведенная функция вычисляет наибольший общий делитель своих аргументов. При вызове этой функции с аргументами 0 и 1 выполняются только инструкции в строках 3 и 4. При вызове с аргументами 1 и 0 будут выполнены строки 3, 5, 6. При вызове с аргументами 1 и –2 выполняются строки 3, 5, 7, 8, 10, 12, 14, 15, 16, 19, 20, 23. Таким образом, набор тестовых данных, состоящий из пар , , обеспечивает полное покрытие инструкций этой функции. Заметим, что вместо можно было бы использовать два набора аргументов, например, и , первый из которых покрывает инструкцию 8, но не покрывает 14, 15, 16, а второй — покрывает эти три инструкции. С точки зрения получаемого покрытия все равно, какой набор тестовых данных выбрать. Однако могут быть существенны другие аспекты, например, время работы тестового набора и удобство анализа результатов тестирования. Время выполнения тестов обычно сокращается при уменьшении их количества, но сложный тест, эквивалентный по покрытию нескольким простым, в ряде случаев может выполняться дольше, чем все они вместе взятые. С точки зрения удобства анализа результатов, чем проще тесты, тем лучше, поскольку меньше различных факторов приходится рассматривать при локализации ошибки, найденной таким тестом. Для обнаружения всех ошибок покрытия 100% инструкций недостаточно. В следующем примере приведен код функции, которая должна по значению целого числа печатать его простую характеристику — ноль это, четное или нечетное число, положительное или отрицательное. В этом коде пропущена вставка слова «нечетное» в описание нечетных чисел. Однако тесты с входными данными 0, 2 и –2 дадут 100% покрытия строк и не обнаружат никаких ошибок.

1 String classifier(int n)

2 {

3 StringBuffer s = new StringBuffer();

4

5 if(n == 0)

6 return "ноль";

7

8 if(n%2 == 0)

9 s.append("четное ");

10

11 if(n < 0)

12 s.append("отрицательное");

13 else

14 s.append("положительное");

15

16 return s.toString();

17 }