Лекция 3. Функциональное тестирование

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

2016

Павел Степанов

Старший преподаватель

Кафедра компьютерной математики и программирования ГУАП

1. Содержание

- Понятие и виды функционального тестирования
- Критерии выбора тестов. Методы черного и белого ящика
- Системы управления тестами
- Оценка качества тестов

2. Понятие функционального тестирования

- функция (function): Реализация в программе алгоритма, по которому пользователь или программа могут частично или полностью выполнять решаемую задачу. (ГОСТ Р ИСО/МЭК 12119/2000)
- Функциональное тестирование – тестирование на предмет реализованности функциональных требований.
- Так как, потенциально, в функциональные требования можно записать почти все, что угодно, то и функционально тестировать можно почти что угодно.
- Нефункциональное тестирование – определение количественных метрик ПО (например, скорости отклика, производительности, времени работы на отказ и пр).
  - Но если эта метрика вписана в функциональные требования, то проверка на соответствие становится уже функциональным тестом (удовлетворяет/не удовлетворяет)
- Функциональное тестирование – скорее процесс, чем тестовый набор. Один и тот же тест может быть одновременно функциональным и каким-то еще.

3. Какое еще бывает тестирование?

- Вообще говоря, единой классификации нет

- Сертификационное
- Производительности
- Юзабилити
- Безопасности
- Локализации
- Конфигурационное
- Тестирование восстановления при отказе
- Регрессионное
- CI-тестирование (тестирование при непрерывной интеграции)
- И пр. и пр.

4. Уровни тестирования

- Модульное
- Интеграционное
- Системное
  - Smoke/BAT/SWAT
  - Альфа
  - Бета
- Приемочное тестирование

5. Черный ящик

- Мы ничего не знаем о внутренней организации программы
  - Тесты пишутся на основе спецификаций. Обычно используются два критерия выбора тестов
    - Проверка классов эквивалентности
    - Проверка граничных условий
  - Выбор классов эквивалентности – на основе эвристик

6. Пример тестирования черным ящиком

- Тестируем функцию long plus(int a, int b)
  - Как она реализована?
    - Например, return a+b
      - Какие проблемы могут быть у этого кода?
        - Переполнение
        - Неверная работа с отрицательными числами
        - Что-то еще?

7. Пример тестирования черным ящиком

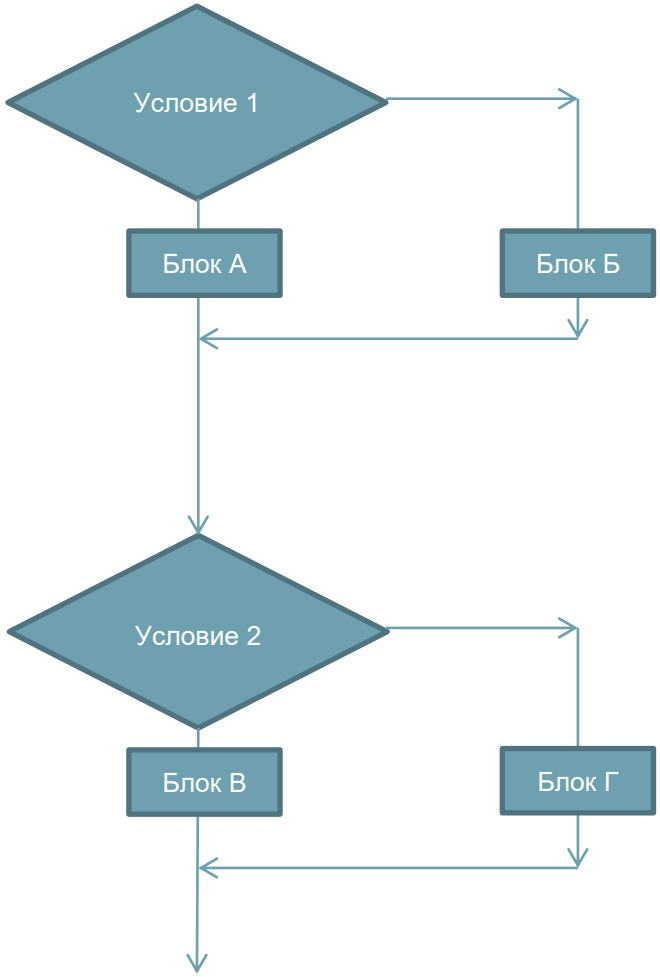
- Тестируем функцию long plus(int a, int b)
  - a=1 b=2
  - a=1 b=-2
  - a= -2 b=1
  - a=-1 b=-2
  - a=MAXINT b=MAXINT
  - a=MININT b=MININT
  - a=MAXINT b=MININT
  - a=MININT b=MAXINT
  - продолжить самостоятельно

8. Формальные требования черного ящика

- Тестирование функций
- Тестирование классов входных данных
- Тестирование классов выходных данных
- Тестирование области допустимых значений (тестирование границ класса)
- Тестирование длины набора данных

9. Белый ящик

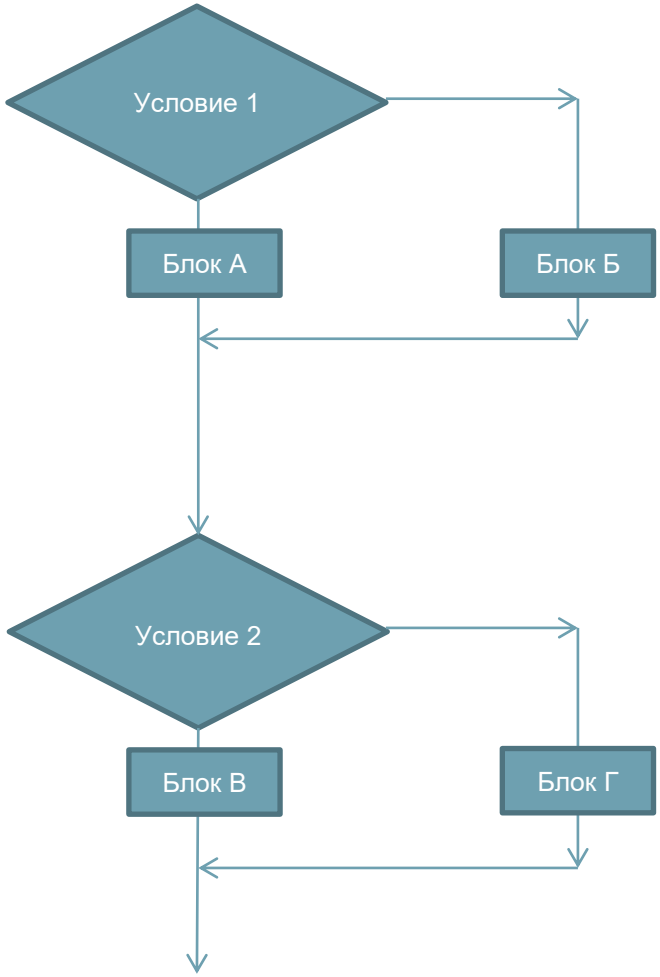
- Текст программы известен
  - Все операторы
    - Тест 1 А В
    - Тест 2 Б Г
  - Все пути
    - Тест 1 А В
    - Тест 2 А Г
    - Тест 3 Б В
    - Тест 4 Б Г



10. Еще стратегии белого ящика

- покрытие решений
- покрытие условий
- покрытие условий и решений
- комбинаторное покрытие условий

- Например
  - Условие 1
    - M>0 & K<5
  - Условие 2
    - M<=0



11. Test Harness

- Среда управления тестами
- Java
  - JUnit
  - TestNG
- C++
  - CppUnit

12. JUnit 4.0 test

```
@Test
public void shouldReturnZero() {


}
```

13. testNG test

- По синтаксису очень похож на Junit
- Больше аннотаций и возможностей
- Другая лицензия (junit - eclipse license, testng – apache license)

14. CppUnit (GNU)

```
#include "stdafx.h"
#include <cppunit/CompilerOutputter.h>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/ui/text/TestRunner.h>


int main(int argc, char* argv[])
{
  // Get the top level suite from the registry
  CppUnit::Test *suite = CppUnit::TestFactoryRegistry::getRegistry().makeTest();

  // Adds the test to the list of test to run
  CppUnit::TextUi::TestRunner runner;
  runner.addTest( suite );

  // Change the default outputter to a compiler error format outputter
  runner.setOutputter( new CppUnit::CompilerOutputter( &runner.result(),
                                                       std::cerr ) );
  // Run the tests.
  bool wasSucessful = runner.run();

  // Return error code 1 if the one of test failed.
  return wasSucessful ? 0 : 1;
}
```

15. CppUnit (Visual Studio)

```
#include "stdafx.h"
#include "CppUnitTest.h"
#include "MyProjectUnderTest.h"
using namespace Microsoft::VisualStudio::CppUnitTestFramework;
namespace MyTest
{
  TEST_CLASS(MyTests)
  {
  public:
      TEST_METHOD(MyTestMethod)
      {

      }
  };
}
```

16. Лучшие практики

- Тест должен быть максимально коротким
- Тесты должны быть взаимонезависимы и, одновременно, самодостаточны
- Тесты должны иметь понятные имена и сообщения об ошибках

17. Демонстрация

18. Метрика покрытия кода

- Автоматический анализ покрытия кода. Покрытие кода по XXX не ниже YY%.
  - Обычно XXX это все строки, все ветви или все пути

19. Покрытие кода - инструменты

- Java
  - Jcov
  - Emma
  - Cobertura
- C++
  - Gcov
  - Tcov
- C#
  - OpenCover

20. Покрытие кода - демонстрация

21. Проверка качества тестовой базы

- Метатестирование
  - Тесту подкладываются эталонные данные (метатесты), про которые известно, должен на них тест проходить или нет, после чего результат проверяется
- Bug Injection
  - Все значения, возвращаемые методами, меняются на неверные (например, противоположные), и если соответствующие тесты не начинают падать, то они проверяются

22. Q&A