Тестирование программного обеспечения

«Тестирование программ может использоваться для демонстрации наличия ошибок, но оно никогда не покажет их отсутствие.» Дейкстра, 1970 г.

Доклад подготовил: Харитонов Федор, ВМИ-336

История развития тестирования программного обеспечения

- Первые программные системы разрабатывались в рамках программ научных исследований или программ для нужд министерств обороны.
- Тестирование таких продуктов проводилось строго формализовано с записью всех тестовых процедур, тестовых данных, полученных результатов.
- Тестирование выделялось в отдельный процесс, который начинался после завершения кодирования, но при этом, как правило, выполнялось тем же персоналом.

История развития тестирования программного обеспечения

- В 1960-х много внимания уделялось «исчерпывающему» тестированию, которое должно проводиться с использованием всех путей в коде или всех возможных входных данных.
- В начале 1970-х тестирование ПО обозначалось как «процесс, направленный на демонстрацию корректности продукта»
- В 1980-х тестирование расширилось таким понятием, как предупреждение дефектов.
- В начале 1990-х в понятие «тестирование» стали включать планирование, проектирование, создание, поддержку и выполнение тестов и тестовых окружений, и это означало переход от тестирования к обеспечению качества, охватывающего весь цикл разработки ПО.
- В середине 1990-х с развитием Интернета и разработкой большого количества веб-приложений особую популярность стало получать «гибкое тестирование»

Контекст ролей

- Команда заказчика включает в себя: владельцев продукта, экспертов предметной области, менеджеров продукта – т.е. всех тех, кто находится на «бизнес-стороне» проекта.
- Команда разработчиков объединяет всех людей, участвующих в поставке кода.

  В команде разработчиков также имеются тестировщики, которые следят за качеством продукции от имени заказчика и помогают добиться максимальной потребительской ценности.

Взаимодействие ролей

Десять принципов гибкого тестирования

- Обеспечение постоянной связи
- Принесение пользы заказчику
- Готовность к личным контактам
- Смелость
- Простота
- Постоянное совершенствование практики
- Реакция на изменение
- Самоорганизация

Квадранты гибкого тестирования

Принципы планирования тестов

- Ценность каждого приема зависит от контекста.
- Существуют хорошие приемы, но не лучшие.
- Люди, работающие вместе – наиболее важная часть контекста любого проекта.
- С течением времени проекты часто развиваются непредсказуемым образом.
- Продукт – это решение. Если проблема не решена, значит, продукт не работает.
- Только вооружившись опытом и знаниями, можно делать правильные вещи в нужное время, чтобы эффективно тестировать разрабатываемые продукты.

Регрессионное тестирование

- Регрессионное тестирование— собирательное название для всех видов тестирования программного обеспечения, направленных на обнаружение ошибок в уже протестированных участках исходного кода.
- Такие ошибки появляются, когда после внесения изменений в программу перестает работать то, что должно было продолжать работать, и их называют регрессионными ошибками.

Регрессионное тестирование

После внесения изменений в очередную версию программы, регрессионные тесты подтверждают, что сделанные изменения не повлияли на работоспособность остальной функциональности приложения. Регрессионное тестирование может выполняться как вручную, так и средствами автоматизации тестирования.

Виды тестирования по уровню знания системы

- Тестирование чёрного ящика
- Тестирование белого ящика

Тестирование чёрного ящика

- Под «чёрным ящиком» понимается объект исследования, внутреннее устройство которого неизвестно.
- Целью тестирования ставится выяснение обстоятельств, в которых поведение программы не соответствует спецификации.

Методы стратегии чёрного ящика

- Эквивалентное разбиение.
- Анализ граничных значений.
- Анализ причинно-следственных связей.
- Предположение об ошибке.

Тестирование белого ящика

- "Белый ящик" - тестирование кода на предмет логики работы программы и корректности её работы с точки зрения компилятора того языка на котором она писалась.
- Стратегия тестирования по принципу Белого ящика позволяет проверить внутреннюю структуру программы. Исходя из этой стратегии тестировщик получает тестовые данные путем анализа логики работы программы.

Методы стратегии белого ящика

- Покрытие операторов.
- Покрытие решений.
- Покрытие условий.
- Покрытие решений и условий.
- Комбинаторное покрытие условий.

Покрытие операторов

- Критерии покрытия операторов подразумевает выполнение каждого оператора программы, по крайней мере, один раз.

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

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

Покрытие решений.

- В соответствии с этим критерием 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) && (b == 0)) x = x/a;
    if (a == 2 || x > 1) x++;
}
```

a=3, b=0, x=3

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

Покрытие условий.

- Записывается число тестов достаточное для того, чтобы все возможные результаты каждого условия в решении были выполнены, по крайней мере, один раз.

```
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 = 4

a = 1 ; b = 1 ; x = 1

Автоматическое тестирование

Существует два основных подхода к автоматизации тестирования: тестирование на уровне кода и GUI-тестирование. К первому типу относится, в частности, модульное тестирование. Ко второму - имитация действий пользователя с помощью специальных тестовых фреймворков.

Автоматическое тестирование

Одной из главных проблем автоматизированного тестирования является его трудоемкость: несмотря на то, что оно позволяет устранить часть рутинных операций и ускорить выполнение тестов, большие ресурсы могут тратиться на обновление самих тестов.

Автоматическое тестирование.

Для автоматизации тестирования существует большое количество приложений.

- HP LoadRunner.
- Segue SilkPerformer.
- IBM Rational FunctionalTester.
- AutomatedQA TestComplete.
- JUnit

JUnit

JUnit — библиотека для тестирования программного обеспечения на языке Java, созданная Кентом Беком и Эриком Гаммой

Пример JUnit

```
public class MathFunc {

    private int variable;

    public MathFunc() {

        variable = 0;

    }

    public MathFunc(int var) {

        variable = var;

    }

    public int getVariable() {

        return variable;

    }

    public long factorial() {
        long result = 1;
        if (variable > 1) {
            for (int i=1; i<=variable; i++)
                result = result*i;
        }
        return result;
    }

    public long plus(int var) {
        long result = variable + var;
        return result;
    }
}
```

Пример JUnit

Для написания тестового класса нам нужно создать наследника junit.framework.TestCase. Затем необходимо определить конструктор, принимающий в качестве параметра строку (String) и передающую ее родительскому классу.

Пример JUnit

```
public class TestClass extends TestCase {

   public TestClass(String testName) {

      super(testName);

   }


   public void testFactorialNull() {

      MathFunc math = new MathFunc();

      assertTrue(math.factorial() == 1);

   }
```

```
   public void testFactorialPositive() {
      MathFunc math = new MathFunc(5);
      assertTrue(math.factorial() == 120);
   }

   public void testPlus() {
      MathFunc math = new MathFunc(45);
      assertTrue(math.plus(123) == 168);
   }
}
```

Пример JUnit

Метод assertTrue проверяет, является ли результат выражения верным. Присутствуют и следующие методы - assertEquals, assertFalse, assertNull, assertNotNull, assertSame.

Пример JUnit

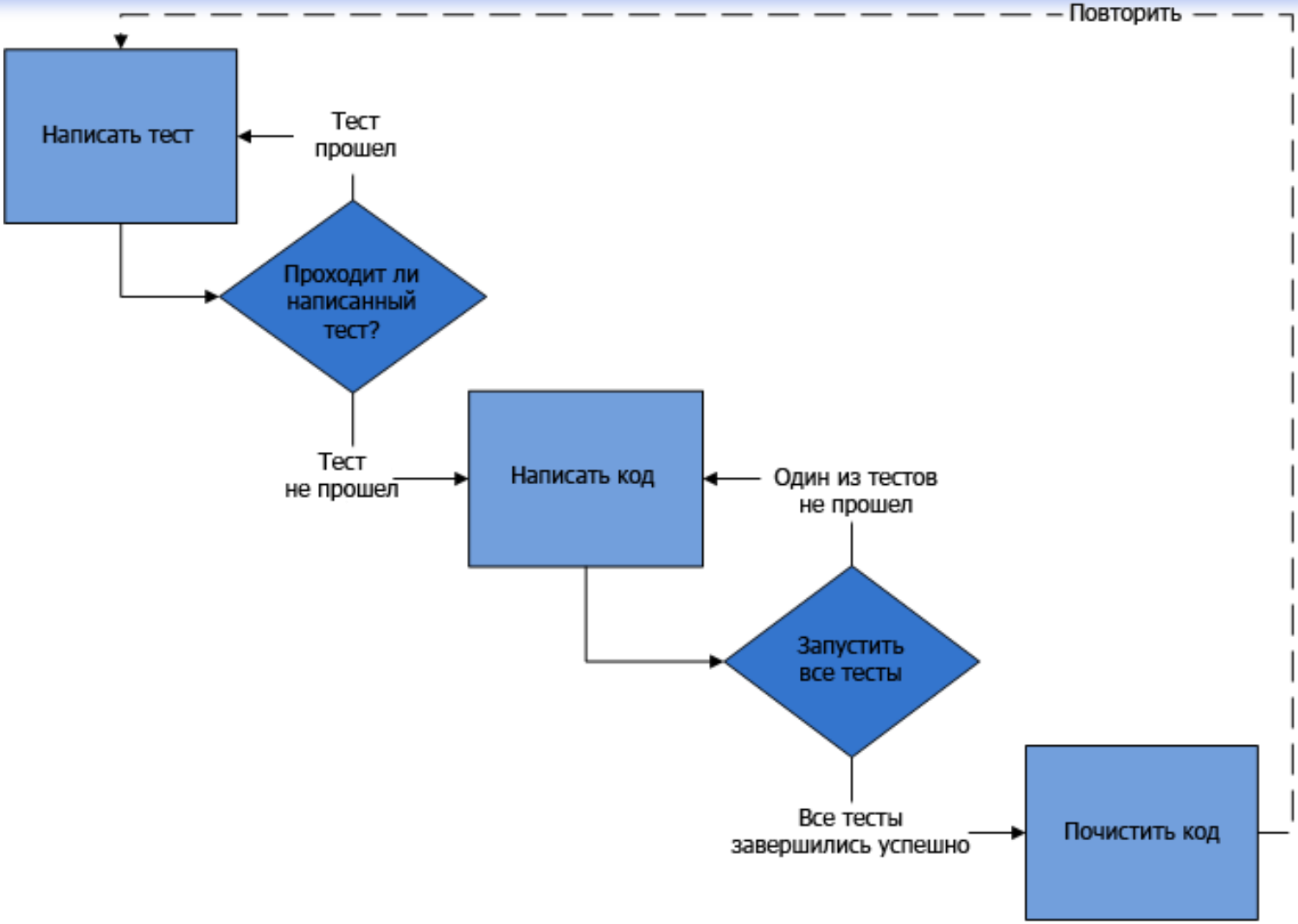
Для того, чтобы объединить тесты, можно воспользоваться классом TestSuite с его методом addTest.

```
public static void main(String[] args) {
    TestRunner runner = new TestRunner();
    TestSuite suite = new TestSuite();
    suite.addTest(new TestClass(“testFactorialNull”));
    suite.addTest(new TestClass(“testFactorialPositive”));
    suite.addTest(new TestClass(“testPlus”));
    runner.doRun(suite);
}
```

Разработка через тестирование

Разработка через тестирование

- Разработка через тестирование (test-driven development, TDD) — техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест и под конец проводится рефакторинг нового кода к соответствующим стандартам


Повторить
Написать тест
Тест прошел
Проходит ли написанный тест?
Тест не прошел
Написать код
Один из тестов не прошел
Запустить все тесты
Все тесты завершились успешно
Почистить код