Модульное тестирование

Владимир Поляков
vladimir.p.polyakov@gmail.com

https://github.com/drxaos-edu/lecture-unit-testing

Модульное тестирование

- Проверка работы программы на уровне отдельных модулей (классов, методов)
- Система как белый ящик
- Модуль как черный ящик
- Пишется программистами.

Интеграционное тестирование

- Проверка совместной работы нескольких модулей.
- Попытка совместно запустить несколько модулей и проверить их взаимодействие.
- Программисты или QA.

Системное тестирование

- Проверка работы системы в целом
- Вся система собрана и запущена
- Проверяется что система решает свою задачу - функциональные требования.
- Юзабилити. QA.

Приемочное тестирование

- Проверка на соответствие требованиям заказчика

Юнит-тесты

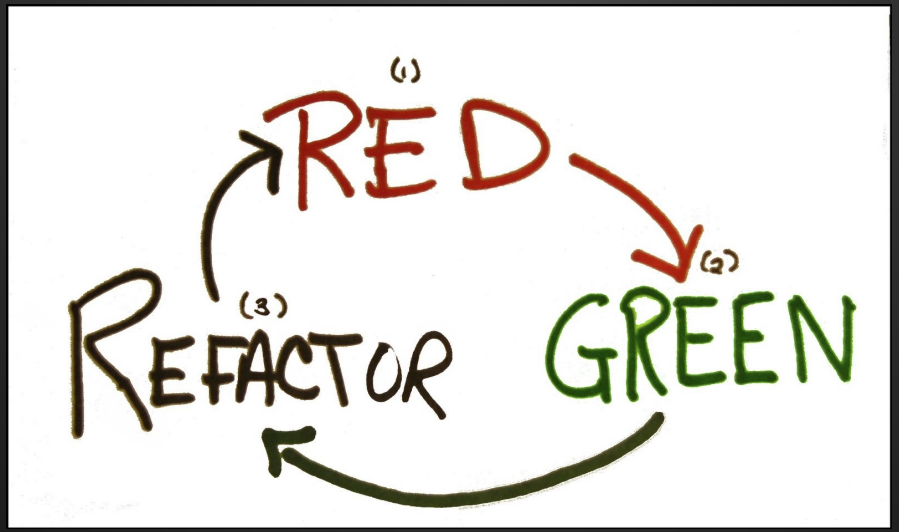
- Часть программного продукта
- может быть столько же, сколько основного кода. Поддерживаются и развиваются параллельно с кодом продукта.
- Хранятся вместе с кодом. Выполняются после каждой сборки.

Юнит-тесты

Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны.

TDD

Test Driven Development - хороший способ начать писать тесты. Идея - начинать все с тестов.



TDD

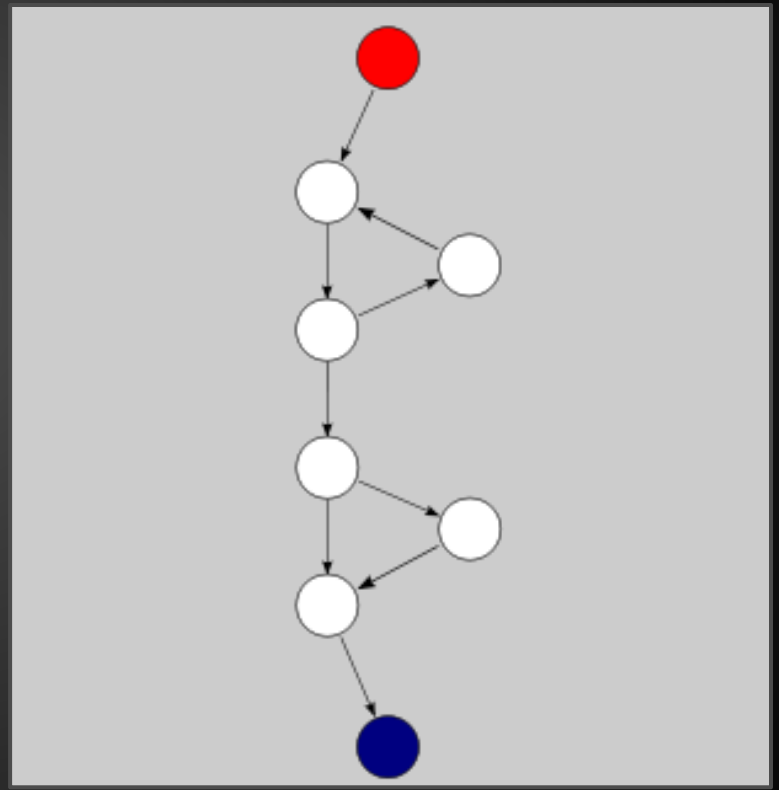
1. Пишем простейший тест, ломающий программу
2. Пишем простейшую реализацию, достаточную для прохождения теста
3. Улучшаем написанный код, не ломая тесты.
4. Возвращаемся к пункту 1

До тех пор пока все требования к программе не будут выполнены

Когда юнит-тесты не работают

Сложный код

Цикломатическая сложность
(Томас Мак-Кейб)
M = e(дуги) - n(вершины) + 2
= 9 - 8 + 2 =3



Когда юнит-тесты не работают

“For each module, either limit cyclomatic complexity to [the agreed-upon limit] or provide a written explanation of why the limit was exceeded.” - Thomas J. McCabe, Sr.

“Для каждого модуля, либо ограничивай цикломатическую сложность (до оговоренного предела), либо предоставляй запись, поясняющую, почему ограничение должно быть превышено.”

Когда юнит-тесты не работают

Результат известен лишь приблизительно

Например, научные программы, где результат становится известным только благодаря написанной программе

Когда юнит-тесты не работают

Ошибки интеграции и функциональности

Инфраструктура

(то как оформляется код)

JUnit (-> xUnit -> cUnit, phpUnit, NUnit), TestNG

JUnit 3.x - test*

JUnit 4.x - аннотации

Проверки

Библиотеки проверок (наборы assert’ов)

FEST Assert, Hamcrest, XMLUnit, HttpUnit

```
assertThat(yoda).isInstanceOf(Jedi.class)
        .isEqualTo(foundJedi).isNotEqualTo(foundSith);
assertThat(Math.sqrt(-1), is(notANumber()));
assertXpathExists("//planet[@name='Earth']", mySolarSystemXML);
assertEquals( "links", 1, table.getTableCell( 0, 2 ).getLinks().length );
```

Stub / Mock

Библиотеки создания тестовых дублеров

Mockito, JMock, EasyMock

«тестовые дублеры» заменяют реальные модули упрощенными, которые обладают более высоким быстродействием и уменьшают количество зависимостей (отрезают тестируемый модуль от других модулей)

Пример

```
import org.junit.Test;
import static org.junit.Assert.*;
public class StringTest {
    @Test
    public void substring () {
        assertEquals("llo", "Hello".substring(3));
    }
}
```

Пример

org.junit.ComparisonFailure: expected:<l[l]o> but was:<l[]o>

at org.junit.Assert.assertEquals(Assert.java:125)

at org.junit.Assert.assertEquals(Assert.java:147)

at ru.compscicenter.java2013.testing.StringTest.substring(StringTest.java:10)

Проверки

assertTrue, assertFalse

assertEquals, assertArrayEquals, assertNotEquals

assertSame, assertNotSame

fail

Варианты с текстом ошибки и без

Проверки

Встроенный assert

```
assert "llo".equals("Hello". substring (3));
assert 1 == 1 : "Arithmetics broken";
```

Поддерживаются только булевские условия

В исключении нет описания проблемы

Надо включать флагом JVM -ea

Структура теста

(Given) Подготовка тестового окружения

(When) Выполнение тестового сценария

(Then) Проверки

Жизненный цикл теста

@BeforeClass

Для каждого @Test-метода:

- создание экземпляра тестового класса
- @Before
- @Test
- @After

@AfterClass

Demo

junit.ExampleTest

Сложные случаи тестирования

В большинстве случаев модульное тестирование не представляет большой проблемы

Особенно в случае TDD, когда целевой код с самого начала проектируется с учетом необходимости тестирования.

Сложные случаи тестирования

Если сначала пишется целевой код, а тестирование производится потом, когда он уже готов, тестирование становится более сложным и трудоемким и сталкивается серьезными проблемами, для решения которых разработаны специальные средства.

Сложные случаи тестирования

- работа с оборудованием;
- работа в реальном времени;
- работа с базами данных;
- работа с удаленными сервисами (например, WWW).

Сложные случаи тестирования

Инкапсуляция — один из столпов, на которых базируется ООП

Затруднено тестирование вспомогательных (приватных) методов класса.

Затруднена проверка внутреннего состояния класса, поскольку оно также надежно спрятано в приватных переменных.

Сложные случаи тестирования

- Если нет возможности использовать внешние сервисы, их нужно чем-то заменить
- Если пользователей интересует не состояние объекта (скрытое) а выполняемые им операции, то давайте и мы будем проверять именно эти операции

Опосредованный ввод

Альтернатива непосредственному вводу.

Тестируемый метод не использует входные параметры, а вместо этого обращается за входными данными к другому методу

Нужно заставить этот самый другой метод вернуть именно те данные, которые нужны для нашего теста

Опосредованный вывод

Иногда результат работы модуля передается другим модулям для дальнейших действий.

Типичный пример: данные передаются модулю интерфейса базы данных с целью их записи в таблицу.

В случаях опосредованного ввода/вывода может оказаться полезным применение тестовых двойников (дублеров).

Тестовая заглушка (Stub)

Назначение — обеспечить наш тестируемый модуль входными данными, требуемыми для данного теста.

Может быть реализована как в фиксированном варианте (выдаваемые данные жестко зашиты в коде), так и в настраиваемом (выдаваемые данные задаются в фазе инициализации теста).

Генератор ответов

Разновидность тестовой заглушки, которая поставляет «хорошие» данные, которые должны корректно обрабатываться в штатном режиме. Генератор ответов обычно используется в тестах основного сценария прецедента, то есть в тестах успешности.

Диверсант

Назначение - имитировать аварийную ситуацию (например, генерируя исключение) или поставлять «плохие» данные, которых не должно быть в штатном режиме. Диверсанты полезны при тестировании отказоустойчивости модуля.

Временная тестовая заглушка

Назначение — предоставить возможность тестирования модуля, который зависит от еще не реализованных модулей

В дальнейшем, когда дело дойдет до заглушенного модуля, он постепенно будет обрастать рабочим кодом и в конечном итоге перестанет быть заглушкой.

Тестовый агент

Назначение - запись всех данных, приходящих от модуля, чтобы потом предоставить полный отчет тестовой системе, которая анализирует эти данные и принимает решение об успешности теста.

Подставной объект (Mock)

Подставной объект соединят в себе функциональность тестового агента и тестовой заглушки.

Он может использоваться для контроля как опосредованного ввода, так и опосредованного вывода.

Подставной объект (Mock)

Перед применением подставного объекта он подвергается настройке, в ходе которой задается ожидаемый сценарий его взаимодействия с тестируемым модулем.

Поддельный объект

Поддельный объект — это облегченная реализация настоящего объекта, которая предоставляет клиенту сходную функциональность.

Например, «поддельная СУБД» может сохранять небольшое количество данных прямо в оперативной памяти, выполнять поиск и другие операции над этими данными.

Объект-заглушка

Используется для передачи в методы, требующие объект с определенной сигнатурой, если этот объект сложно создать в коде теста.

Объект-заглушка имеет схожий интерфейс, но упрощенную реализацию и прост для операций в коде теста.

Stub / Mock

тестовые дублеры не способны заменить реальные модули при нормальной работе приложения.

Подробнее

http://googletesting.blogspot.ru/2008/06/tott-friends-you-can-depend-on.html

Demo

stub для конструктора - save + throw unimplemented

шпион - был ли вызван save

mock - запись порядка вызовов и сверка

возврат значений - total

имитация - бд в памяти

Какие тесты писать?

- Основной сценарий
- Крайние случаи
- Code coverage

Сколько тестов писать?

- “Вы никогда не найдете 100% ошибок”
- Подход 100% покрытия тестами
- Подход выделения части времени на написание тестов
- Регрессионные тесты - пишутся перед исправлением найденной ошибки

Именование тестов

```
class HtmlLinkRewriterTest ... {
 void testAppendsAdditionalParameterToUrlsInHrefAttributes(){…}
 void testDoesNotRewriteImageOrJavascriptLinks(){…}
 void testThrowsExceptionIfHrefContainsSessionId(){…}
 void testEncodesParameterValue(){…}
 }
```

Именование тестов

HtmlLinkRewriter appends additional parameter to URLs in href attributes.

HtmlLinkRewriter does not rewrite image or JavaScript links.

HtmlLinkRewriter throws exception if href contains session ID.

HtmlLinkRewriter encodes parameter value.

Вопросы

vladimir.p.polyakov@gmail.com
https://github.com/drxaos-edu