Тестирование

Юрий Литвинов
yurii.litvinov@gmail.com

??.??.2017г

Внезапно, тестирование

- Любая программа содержит ошибки
- Если программа не содержит ошибок, их содержит алгоритм, который реализует эта программа
- Если ни программа, ни алгоритм ошибок не содержат, такая программа даром никому не нужна

Тестирование не позволяет доказать отсутствие ошибок, оно позволяет лишь найти ошибки, которые в программе присутствуют

Виды тестов

- Модульные
- Интеграционные
- Системные

- Регрессионные
- Приёмочные
- Дымовые (smoke-test)

- UI-тесты
- Нагрузочные тесты
- ...

Модульные тесты

- Тест на каждый отдельный метод, функцию, иногда класс
- Пишутся программистами
- Запускаются часто (как минимум, после каждого коммита)
- Должны работать быстро
- Должны всегда проходить
- Принято не продолжать разработку, если юнит-тест не проходит
- Помогают быстро искать ошибки (вы ещё помните, что исправляли), рефакторить код (“ремни безопасности”), продумывать архитектуру (мешанину невозможно оттестировать), документировать код (каждый тест — это рабочий пример вызова)

Почему модульные тесты полезны

- Помогают искать ошибки
  - Особо эффективны, если налажен процесс Continuous Integration
- Облегчают изменение программы
  - Помогают при рефакторинге
- Тесты — документация к коду
- Помогают улучшить архитектуру
- НЕ доказывают отсутствие ошибок в программе

Best practices

- Независимость тестов
  - Желательно, чтобы поломка одного куска функциональности ломала один тест
- Тесты должны работать быстро
  - И запускаться после каждой сборки
    - Continuous Integration!
- Тестов должно быть много
  - Следить за Code coverage
- Каждый тест должен проверять конкретный тестовый сценарий
  - Никаких try-catch внутри теста
    - @Test(expected = NullPointerException.class)
    - Любая нормальная библиотека юнит-тестирования умеет ожидать исключения
- Test-driven development

Hamcrest

```
assertThat(someString, is(not(equalTo(someOtherString))));
assertThat(list, everyItem(greaterThan(1)));
assertThat(cat.getKittens(), hasItem(someKitten));
assertThat("test",
   anyOf(is("testing"), containsString("est")));
assertThat(x,
   allOf(greaterThan(0), lessThanOrEqualTo(10)));
```

Mock-объекты

- Объекты-заглушки, симулирующие поведение реальных объектов и контролирующие обращения к своим методам
  - Как правило, такие объекты создаются с помощью библиотек
- Используются, когда реальные объекты использовать
  - Слишком долго
  - Слишком опасно
  - Слишком трудно
  - Для добавления детерминизма в тестовый сценарий
  - Пока реального объекта ещё нет
  - Для изоляции тестируемого объекта
- Для mock-объекта требуется, чтобы был интерфейс, который он мог бы реализовать, и какой-то механизм внедрения объекта

Пример: Mockito

```
@Test
public void test() throws Exception {
    // Arrange, prepare behaviour
    Helper aMock = mock(Helper.class);
    when(aMock.isCalled()).thenReturn(true);
    // Act
    testee.doSomething(aMock);
    // Assert - verify interactions (optional)
    verify(aMock).isCalled();
}
```

Соотношение тестов