Advanced testing techniques

When assertThat(you).understandUnitTesting() fails

23 October 2014

Martin Škurla, Equities IT

Speaker's Bio

- Ing. Martin Škurla
  - Certifications
    - Sun Certified Programmer for the Java 2 Platform, Standard Edition 6.0
    - Oracle Certified Professional, Java EE 5 Web Component Developer
  - Open Source Committer
    - Gephi (GSoC 2010), NetBeans (NetCAT), Abego Tree Layout, AST Visualizer, Tyrus
  - Conference/JUG Speaker
    - Fosdem, CZJUG, GeeCON
  - Technical Reviewer
    - CPress, Grada, Manning
  - Previous work experience
    - Sors Technology, ESET, Celum, Atlassian
  - Currently
    - PSEC at Barclays

“More than the act of testing, the act of designing tests is one of the best bug preventers known. The thinking that must be done to create a useful test can discover and eliminate bugs before they are coded – indeed, test-design thinking can discover and eliminate bugs at every stage in the creation of software, from conception to specification, to design, coding and the rest.” [1]

Boris Beizer

[1] Software Testing Techniques, Creating a Software Engineering Culture by Karl Eugene Wiegers, ISBN: 0932633331 , Page: 211

Agenda

- Test face-lifting
  - naming conventions
  - logical grouping
  - test code smells and “refactoring patterns”
  - bad practices and pitfalls
- Custom assertions
- Declarative and Data driven testing
- Advanced techniques
  - TestNG framework integration
  - concurrent test execution
- Results (code reduction, number of tests, execution time)

Naming conventions

- *The most important asset of a unit test is readability.*
- We try to achieve “readable tests”
  - readable vs. understandable tests
- Test method naming conventions:
  - `testNameOfTheMethod()` antipattern
  - method name should contain 3 parts: behavior being tested, inputs and outputs
  - e.g. `attrUsedInRules_ShouldNotBeAbleToRemove()`, `removingAttrUsedInRules_ShouldThrowIllegalArgumentException()`, `ruleEqualsAndHashCodeMethods_ShouldBeCorrectlyImplemented()`, `getAttrVal_ShouldThrowExceptionWhenAmbiguityHappens()`
- Test lifecycle method naming conventions:
  - `setUp()` & `tearDown()` antipatterns
  - e.g. `@BeforeMethod public void rulesInit() {…}`

Logical grouping

- *Don't put all test code related to single class/method always into the same test class.*
  - `NameOfClassTest` antipattern
  - it doesn't scale for anything but trivial examples
- *Instead, you should group tests logically.*
  - AE has attributes, attribute groups, cross rule validation, distinct rules, matching expressions, meta rules, querying API, rule ambiguities handling, rule precedence, target expressions, transactional behavior
  - e.g. `AttrManipulationTest, AttributeEngineTest, AttrGroupTest, CrossRuleValidationTest, DistinctRuleTest, GettingAttrsRulesAndValsAPITest, MatchingAttrExpressionsTest, MetaRuleManipulationTest, MetaRulesTest, RuleAmbiguitiesTest, RuleManipulationTest, RuleMatchValsTest, RulePrecedenceTest, TargetAttrExpressionsTest, TransactionalBehaviorTest`

Test code smells and “refactoring patterns”

- Conditions
  - introducing multiple test execution paths is bad
  - => multiple tests, JUnit assumptions, TestNG `SkipException`
- Loops
  - introducing multiple test cases into a single test is bad
  - => parameterized testing, more powerful (custom) assertions
- Assertions on different objects
  - test should only assert on a single object: SUT
  - => refactoring to multiple tests
- Multiple assignments into the same variable
  - reinitializing the SUT is bad
  - => refactoring to multiple tests

“Yesterday’s best practices is tomorrow’s anti-pattern.” [2]

Neal Ford

[2] GeeCON Prague 2014 Conference, This is water, Opening Keynote

Bad practices

- Using `assertTrue()`, `assertFalse()`, `assertEquals()`
  - => use custom assertions
- Sophisticated string assertion description
  - `assertEquals(String message, Object expected, Object actual);`
  - => use custom assertions
- Logging and printing into console
  - => remove it
- Boolean `asserted` aggregation
  - => rewrite to multiple tests
- Sophisticated tests
  - GC tests, memory leak tests, latency tests
  - => remove them

Common pitfalls

- Removing repetition
  - some level of repetition is actually good
  - => don’t do that
- Introducing constants
  - test constants should only be introduced if the particular value is not important for test
  - => don’t introduce constants as in normal code
- SUT initialization in `@BeforeMethod`
  - not conforming to arrange-act-assert
  - not directly obvious how SUT is initialized
  - introduces mutability (stay tuned)
  - => don’t initialize SUT in `@BeforeMethod` if not necessary
- Reusing SUT
  - you might be relying on already modified state
  - => never reuse already used SUT

Agenda

- Test face-lifting
  - naming conventions
  - logical grouping
  - test code smells and “refactoring patterns”
  - bad practices and pitfalls
- Custom assertions
- Declarative and Data driven testing
- Advanced techniques
  - TestNG framework integration
  - concurrent test execution
- Results (code reduction, number of tests, execution time)

Custom assertions

- Because of readability
  - `assertThatAttr("T1").hasDescription("…");`
- Very useful when API is leaking implementation details
  - e.g. returning complicated data structures
  - `assertThatUnsuppressedRules(rules).containRulesOnLines(1, 2);`

    `assertThatSuppressedRules(rules).isNull();`
- Very powerful when custom assertions reuse other custom assertions
  - `assertThatException(e).forRuleOnLine(1).hasDependentCount(1)`

    `.forRuleOnLine(2).hasDependentCount(2);`
- *Think twice before adding custom assertions.*

Agenda

- Test face-lifting
  - naming conventions
  - logical grouping
  - test code smells and “refactoring patterns”
  - bad practices and pitfalls
- Custom assertions
- Declarative and Data driven testing
- Advanced techniques
  - TestNG framework integration
  - concurrent test execution
- Results (code reduction, number of tests, execution time)

Declarative and Data driven testing

- Data driven testing
  - testing the same code, but with different inputs and outputs
- Declarative testing
  - tests are saying what to do and not how

```
@DataProvider private String[][] attrGroupDataProvider() {
    return new String[][] {{"XETRA", "b111", "rule2"}};}

@MatchingAttrDelimiter("!")
@AeColumns("M1        ! M2     || +M1     || T1    ")
@AeRules( {"LSE       !        || EUROPE ||       ",
           "+EUROPE" ! /a111 ||         || rule1")
@Test(dataProvider= "attrGroupDataProvider" )
public void attrGroup_...(String m1, String m2, String expT1) {}
```

Agenda

- Test face-lifting
  - naming conventions
  - logical grouping
  - test code smells and “refactoring patterns”
  - bad practices and pitfalls
- Custom assertions
- Declarative and Data driven testing
- Advanced techniques
  - TestNG framework integration
  - concurrent test execution
- Results (code reduction, number of tests, execution time)

TestNG framework integration

- TestNG listeners
  - for handling declarative testing
  - `@AEColumns, @AEFileResources, @AERules, @MatchingAttrDelimiter, @MetaAEColumns, @MetaAERules, @TrimOnlyLeadingAndTrailingSpaces`
  - listeners should play nicely with data providers
  - listeners should support concurrency
- But how does it feel like to integrate into TestNG?
  - happiness followed by hacking[3]

```
private static final Set<String> UGLY_HACK = new HashSet<String>();
```

  - you have to expect issues
    - internal bugs (Maven surefire plug-in)
    - workarounds (ROCK-581)

[3] *"to devise or modify (a computer program), usually skillfully" according to http://dictionary.reference.com/browse/hacking?s=t*

Concurrent test execution

- *The point of adding concurrency into tests was surprisingly not performance.*
  - the reason was the independence (isolation)
- Different granularities
  - test suites, test classes, test methods
- *Achievable by removing all unnecessary and guarding the necessary shared state.*
  - application of thread-safety and functional principles
- Test framework integration
  - solid understanding of the testing framework thread model is essential
- But what to do when thread-safety cannot be achieved?
  - `@Test(singleThreaded = true)`
  - JVM forking (last resort solution)

Agenda

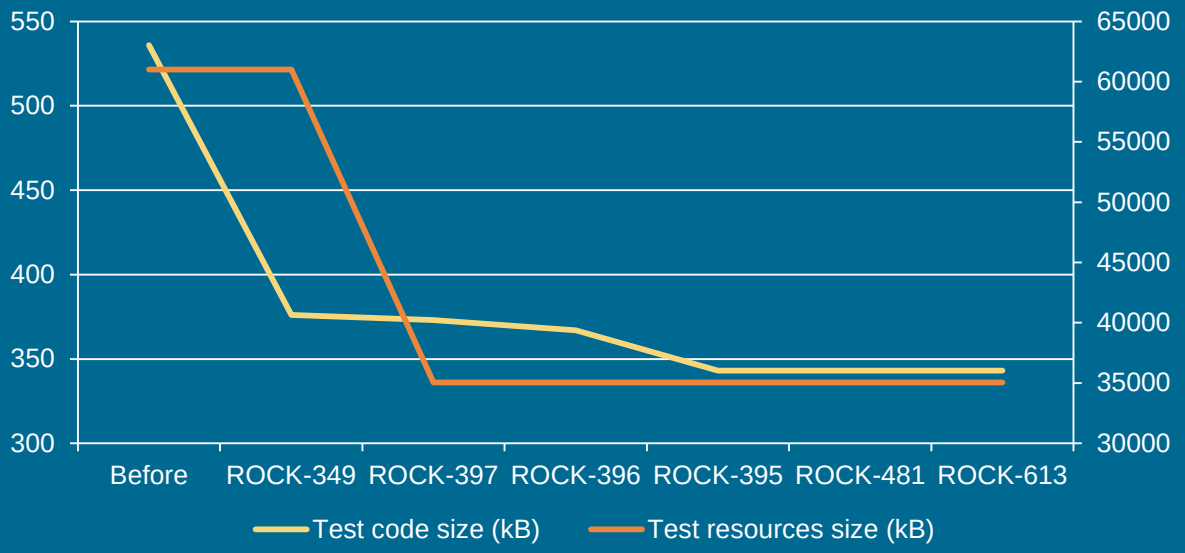
- Test face-lifting
  - naming conventions
  - logical grouping
  - test code smells and “refactoring patterns”
  - bad practices and pitfalls
- Custom assertions
- Declarative and Data driven testing
- Advanced techniques
  - TestNG framework integration
  - concurrent test execution
- Results (code reduction, number of tests, execution time)

Results

- JIRA issues resolved:
  - specifically created: ROCK-349, ROCK-395, ROCK-396, ROCK-397, ROCK-481, ROCK-613
  - resolved as a side-effect of refactoring: ROCK-295
  - TestNG workarounds: ROCK-581
  - won’t fix: ROCK-511
  - introduced bug: ROCK-585
  - pending: ROCK-495
- Testable documentation
- This is in line with Barclay’s Values including Stewardship

Results

- Number of tests:
  - 373.38%
- Execution time:
  - 42.64%
- Number of lines:
  - 61.57%
- Test code size:
  - 63.99%
- Test resources size:
  - 57.45%



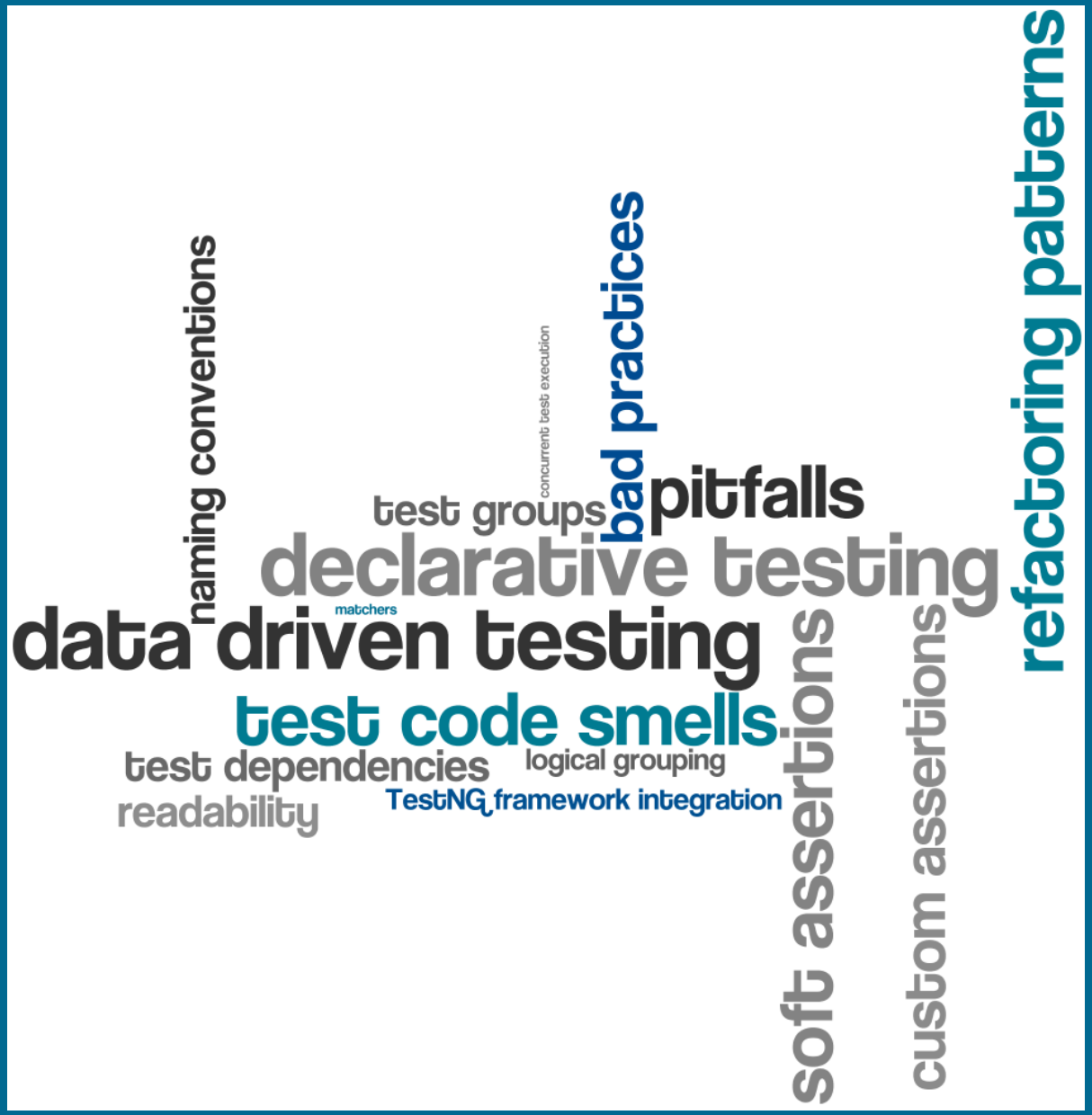
BARCLAYS

More on testing libraries, books

- Links[4]:
  - http://testng.org/doc/index.html
  - http://code.google.com/p/hamcrest/
  - http://joel-costigliola.github.io/assertj/
  - http://code.google.com/p/catch-exception/
  - http://www.manning.com/koskela2/
  - http://www.manning.com/osherove2/

[4] *These links were accessed on 23/09/2014. The websites are those of third parties and Barclays is not responsible for any information stated to be obtained or derived from these third party sources.*

Q & A


naming conventions
concurrent test execution
bad practices
refactoring patterns
test groups
pitfalls
declarative testing
matchers
data driven testing
soft assertions
custom assertions
test code smells
test dependencies
logical grouping
readability
TestNG framework integration

Thank you for your attention

BARCLAYS

Disclaimer

CONFLICTS OF INTEREST BARCLAYS IS A FULL SERVICE INVESTMENT BANK. In the normal course of offering investment banking products and services to clients. Barclays may act in several capacities (including issuer, market maker, underwriter, distributor, index sponsor, swap counterparty and calculation agent) simultaneously with respect to a product, giving rise to potential conflicts of interest which may impact the performance of a product.

NOT RESEARCH This document is from a Barclays Trading and/or Distribution desk and is not a product of the Barclays Research department. Any views expressed may differ from those of Barclays Research.

BARCLAYS POSITIONS Barclays, its affiliates and associated personnel may at any time acquire, hold or dispose of long or short positions (including hedging and trading positions) which may impact the performance of a product.

FOR INFORMATION ONLY THIS DOCUMENT IS PROVIDED FOR INFORMATION PURPOSES ONLY AND IT IS SUBJECT TO CHANGE. IT IS INDICATIVE ONLY AND IS NOT BINDING.

NO OFFER Barclays is not offering to sell or seeking offers to buy any product or enter into any transaction. Any transaction requires Barclays' subsequent formal agreement which will be subject to internal approvals and binding transaction documents.

NO LIABILITY Barclays is not responsible for the use made of this document other than the purpose for which it is intended, except to the extent this would be prohibited by law or regulation.

NO ADVICE OBTAIN INDEPENDENT PROFESSIONAL ADVICE BEFORE INVESTING OR TRANSACTING. Barclays is not an advisor and will not provide any advice relating to a product. Before making an investment decision, investors should ensure they have sufficient information to ascertain the legal, financial, tax and regulatory consequences of an investment to enable them to make an informed investment decision.

THIRD PARTY INFORMATION Barclays is not responsible for information stated to be obtained or derived from third party sources or statistical services.

PAST & SIMULATED PAST PERFORMANCE Any past or simulated past performance (including back-testing) contained herein is no indication as to future performance.

OPINIONS SUBJECT TO CHANGE All opinions and estimates are given as of the date hereof and are subject to change. Barclays is not obliged to inform investors of any change to such opinions or estimates.

NOT FOR RETAIL This document is being directed at persons who are professionals and is not intended for retail customer use.

IMPORTANT DISCLOSURES For important regional disclosures you must read, click on the link relevant to your region. Please contact your Barclays representative if you are unable to access.

EMEA EMEA Disclosures APAC APAC Disclosures U.S. US Disclosures

IRS CIRCULAR 230 DISCLOSURE: Barclays does not provide tax advice. Please note that (i) any discussion of US tax matters contained in this communication (including any attachments) cannot be used by you for the purpose of avoiding tax penalties; (ii) this communication was written to support the promotion or marketing of the matters addressed herein; and (iii) you should seek advice based on your particular circumstances from an independent tax advisor.

CONFIDENTIAL This document is confidential and no part of it may be reproduced, distributed or transmitted without the prior written permission of Barclays.

ABOUT BARCLAYS Barclays Bank PLC offers premier investment banking products and services to its clients through Barclays Bank PLC. Barclays Bank PLC is authorised by the Prudential Regulation Authority and regulated by the Financial Conduct Authority and the Prudential Regulation Authority and is a member of the London Stock Exchange. Barclays Bank PLC is registered in England No. 1026167 with its registered office at 1 Churchill Place, London E14 5HP.

BARCLAYS