TEST-DRIVEN SOFTWARE DEVELOPMENT

JOHN BLUM

2015 JUNE 17

AGENDA

- Misconceptions
- Why Test
- Types of Tests
- What & How to Test
- Using **JUnit**, **Mockito** and **MultithreadedTC**
- Best Practices
- Final Thoughts
- QA

“*No amount of testing can prove a software right, but a single test can prove a software wrong.*”
– Amir Ghahrai

MISCONCEPTIONS

Testing is a *Quality Engineer's* (QE/SDET) job

- *Quality Assurance* (QA) is <u>everyone's</u> responsibility

MISCONCEPTIONS

Automated tests are unnecessary if I manually test and debug my code

✧ Testing is <u>not a replacement</u> for debuggers and other development tools either (e.g. *Profilers*, *FindBugs, etc*)

Not possible to automate all tests

✧ Difficult when the software architecture & design is flawed

MISCONCEPTIONS

Developer tests are "Unit Tests"

JUnit is a "Unit Test" framework

- ✧ JUnit is a framework facilitating the development and execution of automated tests
- ✧ Used in Integration and Functional (Acceptance) testing

MISCONCEPTIONS

Software is correct when there is 100% test coverage and no *FindBugs* (*CheckStyle, PMD*) errors

- ✧ A bug occurs when the software fails to function properly in a prescribed manner
- ✧ A defect occurs after the software is used in a unintentional way and behaves unexpectedly

"*Not everything that can be counted counts and not everything that counts can be counted.*"
- Albert Einstein

Why do software engineers write tests?

Or…

Why don’t software engineers write tests?

TO TEST OR NOT TO TEST?

Time Constraints

- ✧ *Time-based* vs. *Functional-based* releases
- ✧ Quantity (Scope) over Quality (Less is More)

(Cultural) Responsibility

Laziness, Ignorance, Fear

- ✧ Not detail-oriented

WHY TEST

Verifies software is behaviorally correct and functionally complete

- ✧ "*Definition of Done*" (DoD)
- ✧ No "*works on my box*" arguments (we don't ship your box)

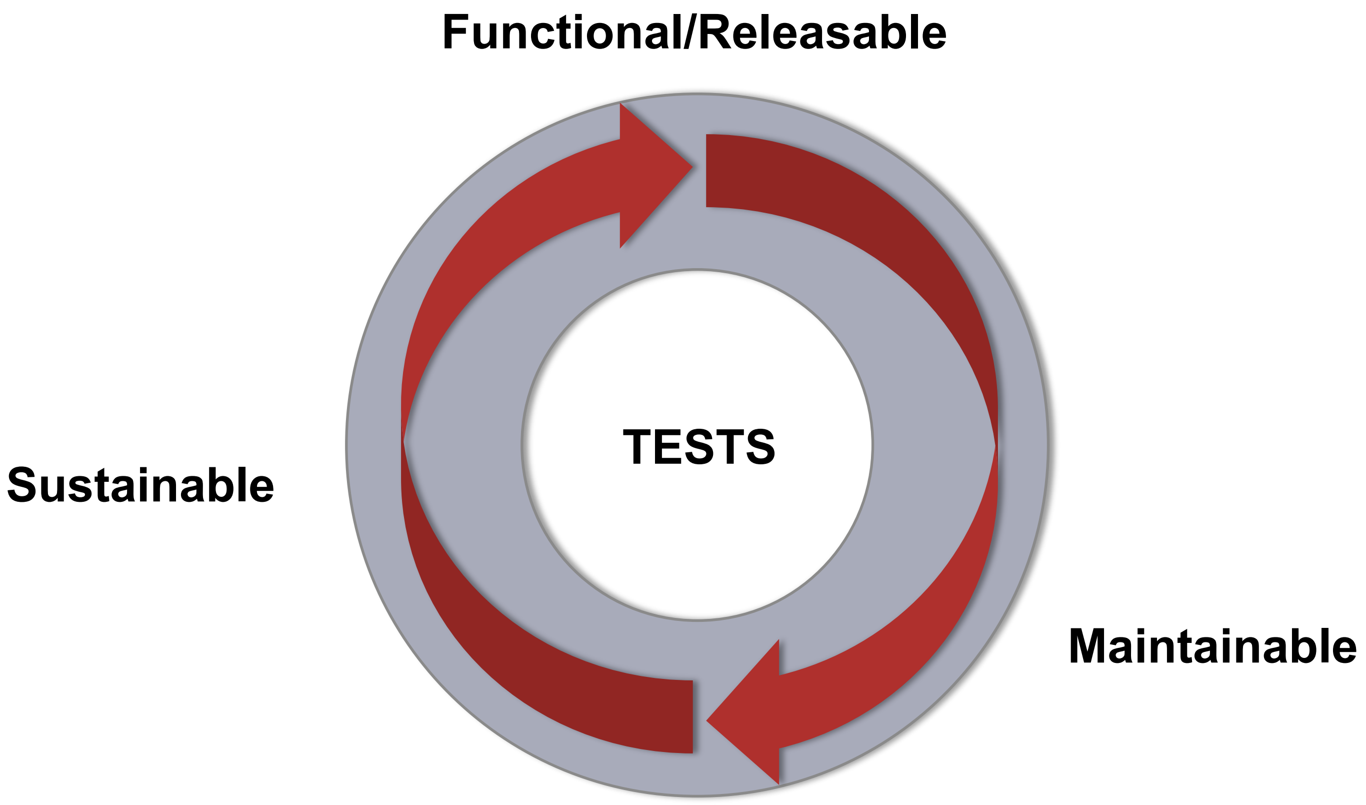
Ensures functional integration

- ✧ Your code plays nicely with others

Increases regression coverage

- ✧ Tests in a suite is money in the bank; ensure what worked yesterday, works today and will work tomorrow

WHY TEST



WHY TEST

Tests are a form of feedback

- Developers get *immediate feedback* on changes to code by running the test(s) to ensure the "*contract is upheld*"

Tests are a form of documentation

- Demonstrates how the code (e.g. API) is properly used

WHY TEST

Focus on the problem that needs to be solved

- ✧ Tests are a "*contract for deliverables*"; avoid "*scope creep*"
- ✧ Tests limits "*over-engineering*" (future coders)
- ✧ TDD

Testing gives developers confidence in their changes

- ✧ And more importantly… to "refactor" and make changes
- ✧ To learn

Testing encourages smaller, more frequent commits

- ✧ And by extension, "releasable" code

WHY TEST

Testing identifies design flaws

- ✧ Hard to test code is flawed and a sure sign of technical debt

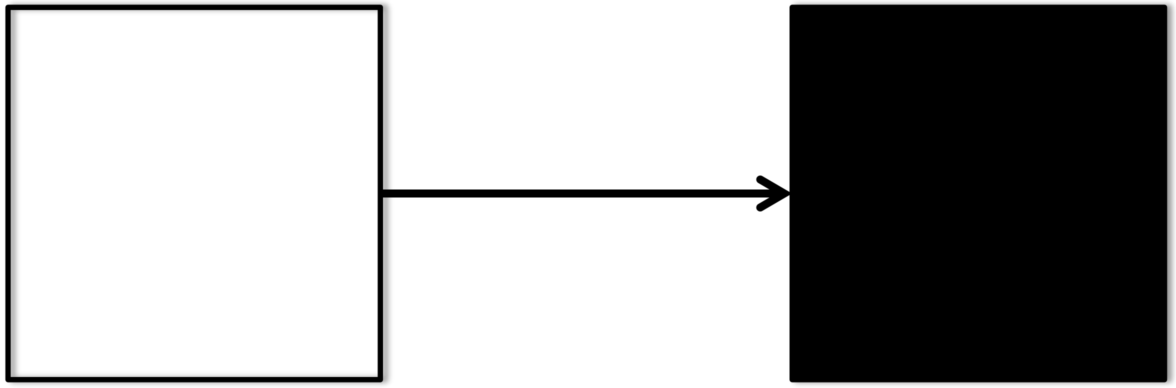
Untested code triggers a domino effect

- ✧ **User** becomes the **Tester** leading to undesirable, costly "*workarounds*" leading to technical debt leading to other bugs leading to more serious issues like data corruption and so on

“*Don’t be that guy!*”

(the guy who doesn’t test his code)

Why are there different types of tests?



TYPES OF TESTS

Different *types of tests* cover different “aspects” (concerns) of the software under test…

- ✧ Different test types serve to “*close the feedback loop*” at different intervals in the software development lifecycle…

TYPES OF TESTS

Unit Tests

- Tests a single, contained "unit" of functionality
- Objects adhere to their contract (specified by the "*interface*")
- Class invariants are upheld as object state changes

Integration Tests

- Tests interactions between "*collaborators*"
- Actual "*dependencies*" used

TYPES OF TESTS

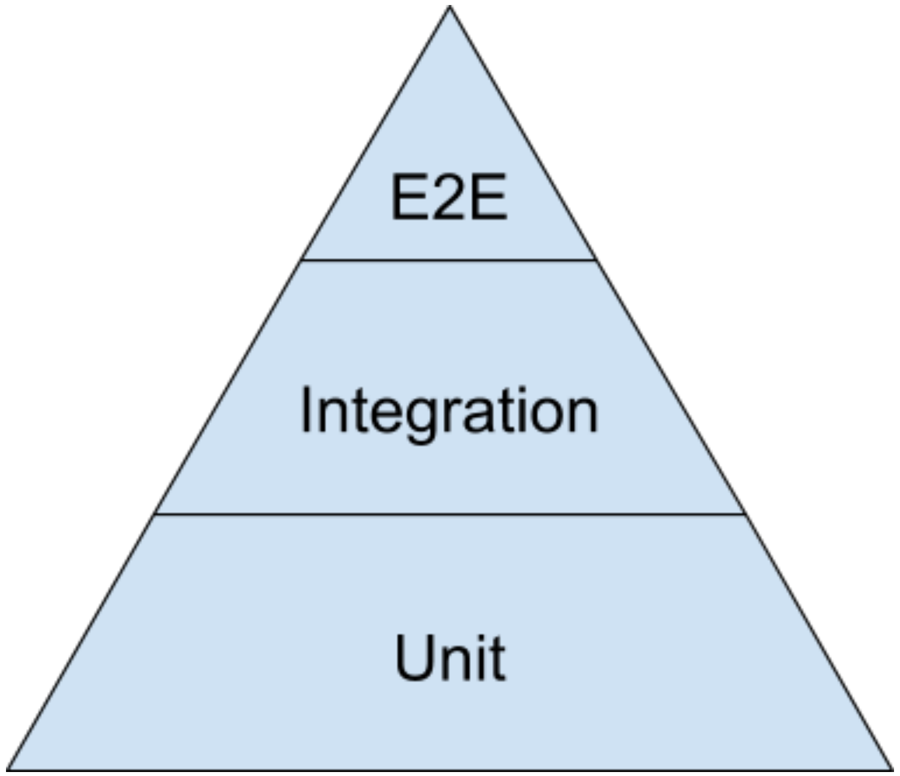
Functional, Acceptance-based Tests (End-To-End)

- ✧ Tests user/software interaction based on predefined workflows (Use Cases) and Functional Requirements
- ✧ *Acceptance Tests* are defined by Acceptance Criteria (based on *Functional Requirements*)
- ✧ Usability Testing (UAT)

Performance-based Tests

- ✧ load/stress testing

TYPES OF TESTS



http://googletesting.blogspot.com/2015/04/just-say-no-to-more-end-to-end-tests.html

“*What*” and “*How*” do you test?

WHAT TO TEST

Test everything (or as much as you can)!

- ✧ Constructors, methods, input, output, Exception conditions, all code paths, invariants/object state, thread safety…
- ✧ Test the unexpected

Don’t make assumptions; verify with tests

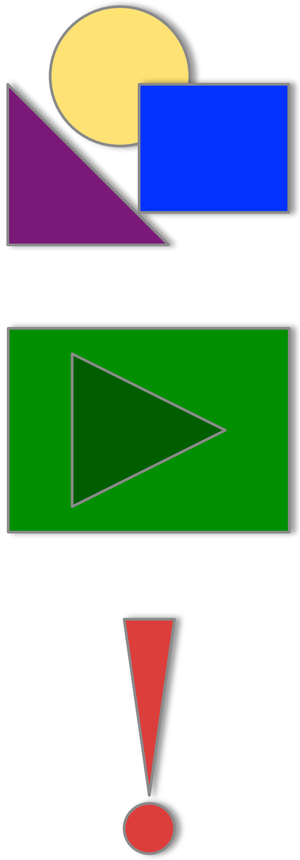
- ✧ “*It is not only what you don’t know that gets you in trouble, it is what you thought you knew that just isn’t so!*”

ANATOMY OF A TEST

Arrange

Act

Assert (Verify)



How-To Unit Test Demonstration

JUnit

JUNIT - ASSERTIONS

With `assertThat(..)`

- ✧ More Readable (natural/DSL) and Strongly-typed
- ✧ Assert subject (actual), verb, object (expected)

Example

`assertThat(x is(3));` // assert "x is 3" or "x == 3"

vs…

`assertEquals(3, x);` // assert "equals 3 x" or "== 3 x"

Examples…

JUNIT - ASSUMPTIONS

Useful to make test dependencies (pre-conditions) explicit

✧ Examples: Environment Configuration, Resource Availability

With `assumeTrue(..)` **(JUnit) or the DSL-friendly** `assumeThat(..)` **(Hamcrest)**

Examples…

JUNIT - EXCEPTIONS

With `Try/Catch` **Idiom (JUnit 3.x)**

With `@Test(expected = Class<? extends Throwable>)`

With `ExpectedException` **Rule (JUnit 4)**

Examples…

JUNIT - PARAMETERIZED

Parameterized test class instances created for the cross-product of test case methods and test data elements.

- ✧ Useful for large volume of data

Run test class with the `Parameterized` Test Runner.

```
@RunWith(Parameterized.class)
public class ParameterizedTest {
  @Parameters
  public Iterable<Object[]> data = …;

  public ParameterizedTest(..) {
  }
}
```

Example…

JUNIT - RULES

~~**ErrorCollector**~~ – allows execution of test to continue after the first problem is encountered (Fail-Fast!)

ExpectedException – specify expected error conditions in test

ExternalResource – setup external resource (File, Socket, Database Connection, …)

TemporaryFolder – creation of files and folders that are deleted after test case (method) finishes

TestName – captures the name of a test inside test methods

Timeout – applies same timeout to all test case methods in test class

Example…

JUNIT - TIMEOUTS

With `@Test(timeout = ms)`

```
@Test(timeout = 500)
public void testWithTimeout() {
}
```

With `Timeout` **Rule…**

```
public class TestsWithGlobalTimeout {
  @Rule
  public Timeout globalTimeout = Timeout.seconds(20);
}
```

Example…

JUNIT – FIXTURES

With *`@BeforeClass`*, *`@Before`*, *`@AfterClass`*, *`@After`*

With (optionally) *`@ClassRule`* **and** `ExternalResource` **Rule**

Examples…

JUNIT – EXECUTION ORDER

By design, JUnit <u>does not</u> specify test execution order

- ✧ Test case method invocation determined by Reflection API
- ✧ JDK 7 is more or less random

Use `MethodSorters` **and** *`@FixMethodOrder`* **annotation**

- ✧ E.g. `@FixMethodOrder(MethodSorters.NAME_ASCENDING)`

Example…

JUNIT – AGGREGATION

Enables suites of tests to be grouped and built from existing test classes using…

```
@RunWith(Suite.class)
@Suite.SuiteClasses({
  ClientCommandsTest.class,
  DiskStoreCommandsTest.class,
  FunctionCommandsTest.class,
  IndexCommandsTest.class,
  MemberCommandsTest.class,
  QueueCommandsTest.class,
  …
})
public class GfshTestSuite {
}
```

JUNIT – ADDITIONAL FEATURES

- **Test Runners** (e.g. `Categories`, `Parameterized`, `MockitoJUnitRunner`, `SpringJUnit4ClassRunner`)
- **Categories** (e.g. `UnitTests`, `IntegrationTests`, `AcceptanceTests`)
- **Theories** – more flexible/expressive assertions combined with ability to state assumptions
- **Rule Chaining** – with `RuleChain` to control test rule ordering
- Multithreaded Code and Concurrency (support)
  - Eh, MultithreadedTC is better!

JUNIT – EXTENSIONS

HttpUnit - http://httpunit.sourceforge.net/

HtmlUnit - http://htmlunit.sourceforge.net/

Selenium - http://www.seleniumhq.org/

JUext - http://junitext.sourceforge.net/

http://www.tutorialspoint.com/junit/junit_extensions.htm

Unit Testing with *Mocks* using Mockito

Why use Mocks in *testing*?

UNIT TESTING WITH MOCKS

Because… "*If you can't make it, fake it*"

Mocks **ensure focus is on the Subject ("unit") of the test by mocking interactions with *Collaborators* to verify appropriate behavior of the Subject, not the *Collaborator(s)***

- *Mocked Collaborators* are "expected to behave" according to their contract

Promotes *programming to interfaces* and delineation of functional responsibility across teams

How-To Mock Demonstration

Mockito

MOCKITO - CALLBACKS

With…

```
when(mock.doSomething(..)).thenAnswer(new Answer<Object>() {
  @Override public Object answer(InvocationOnMock invocation) throws Throwable {
    Object[] args = invocation.getArguments();
    Integer intArg = invocation.getArgumentAt(0, Integer.class);
    Object mock = invocation.getMock();
    return …;
  }
));
```

MOCKITO - STUBBING

Consecutive calls…

```
when(mock.getSomething(..)).thenReturn(“one”, “two”, “three”);
```

MOCKITO – ORDER VERIFICATION

With…

```
InOrder inOrderVerifier = inOrder(firstMock, secondMock);


inOrderVerifier.verify(firstMock, times(2)).doSomething(..);

inOrderVerifier.verify(secondMock, atLeastOne()).doSomething(..);
```

MOCKITO - SPIES

Possible answer to… “*Do not mock code you don’t own*”

```
List<Object> list = new ArrayList<Object>();
List<?> spy = spy(list);

list.add(“one”);

verify(spy, times(1)).add(eq(“one”));
```

MOCKITO - LIMITATIONS

Cannot mock final (non-extensible) classes or final (non-overridable) methods.

Cannot stub Spies in the usual way…

```
List<?> spy = spy(new ArrayList());
// Impossible - actual method is called so spy.get(0) throws
IndexOutOfBoundsException
when(spy.get(0).thenReturn("TEST");
// Use doReturn(..) to do stubbing
doReturn("TEST").when(spy).get(0);
```

??

MOCKITO - EXTENSIONS

PowerMock – enables mocking of static methods, constructors, final classes and methods, private methods, removal of static initializers plus more…

- Uses custom `ClassLoader`
- https://code.google.com/p/powermock/

MultithreadedTC
(sorry)

BEST PRACTICES

Test one thing at a time (per test case)

- ✧ Single code path; one interaction with a collaborator; one user story, and so on…
- ✧ Prefer <u>more</u> test cases rather than <u>bloated</u> test cases

Tests should run quickly providing immediate feedback

* 10-minute build

Tests should fail-fast

Tests should be 100% reliable

BEST PRACTICES

Please <u>do not</u> ignore (@Ignore) or comment out failing tests!

- ✧ Understand test failures, take responsibility, fix the failures and don't commit until all tests pass

Write a test <u>before</u> fixing a bug

- ✧ Without a test it is problematic to verify the fix

BEST PRACTICES

Test cases should be independent and execution order should not matter

Use meaningful test case (method) names

BEST PRACTICES

Ideally, interchanging *Mocks* with actual *Collaborators* does not require any test changes

BEST PRACTICES

Follow the AAA Testing Pattern

- ✧ Arrange -> Act -> Assert

Follow the commit pattern…

1. Update, Create Topic Branch
2. Make Changes
3. Run Tests (if failure(s), goto 2)
4. Update (if changes, goto 3)
5. Merge & Commit

ANATOMY OF A BUG

Title

- short, searchable *summary of the problem*; useful as a commit message

Synopsis

- descriptive *accounts of the problem* (env, conditions, workarounds)

Steps to Reproduce

- data, thread dumps, stack traces, log files; preferably a test case reliably reproducing the issue

Expected Result

Actual Result

FINAL TESTING THOUGHT

Remember…

There is "good code" and then there is "tested code".

REFERENCES

http://www.softwaretestinghelp.com/types-of-software-testing/

http://junit.org/

http://mockito.org/

http://docs.mockito.googlecode.com/hg/org/mockito/Mockito.html

https://code.google.com/p/powermock/

https://www.cs.umd.edu/projects/PL/multithreadedtc/overview.html

http://googletesting.blogspot.com/2015/04/just-say-no-to-more-end-to-end-tests.html

https://zeroturnaround.com/rebellabs/olegs-half-smart-hacks-put-a-timestamp-on-ignore-with-java-8s-new-date-and-time-api/

http://www.codeaffine.com/2013/11/18/a-junit-rule-to-conditionally-ignore-tests/

REFERENCES

https://github.com/codeprimate-software/test-driven-development

Questions