TESTING WITH MOCKS

Macon Pegram

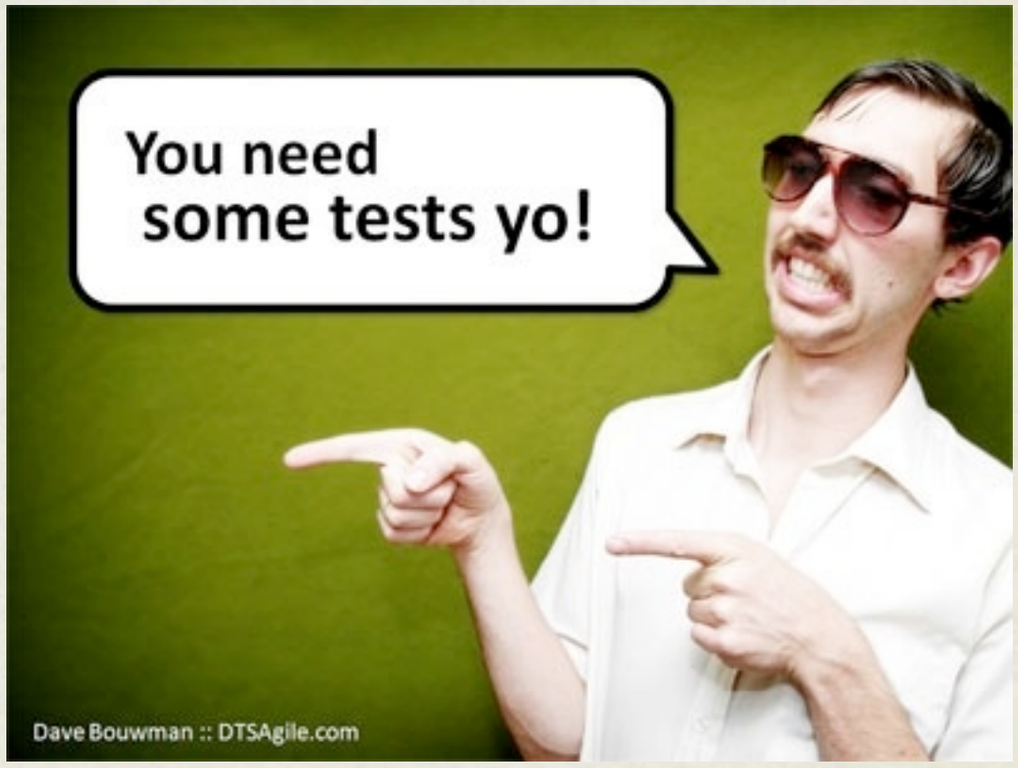
Blog: http://blogs.captechconsulting.com/blog/author/Macon%20Pegram
Code: https://github.com/mpegram3rd/testing-with-mocks

SOUND FAMILIAR?

- I’d rather not change that function
- This defect keeps coming back
- It’s impossible to unit test our project
- The backend/database is not available
- Setting up my test environment takes too long
- I have a testing specific Spring context

UNIT TESTING: WHY BOTHER?

- Detect / Avoid defects
- Validate Bug Fixes
- Refactor without hesitation
- Forces you to think about the design
- Keeps you out of the debugger



WHAT DO WE TEST?

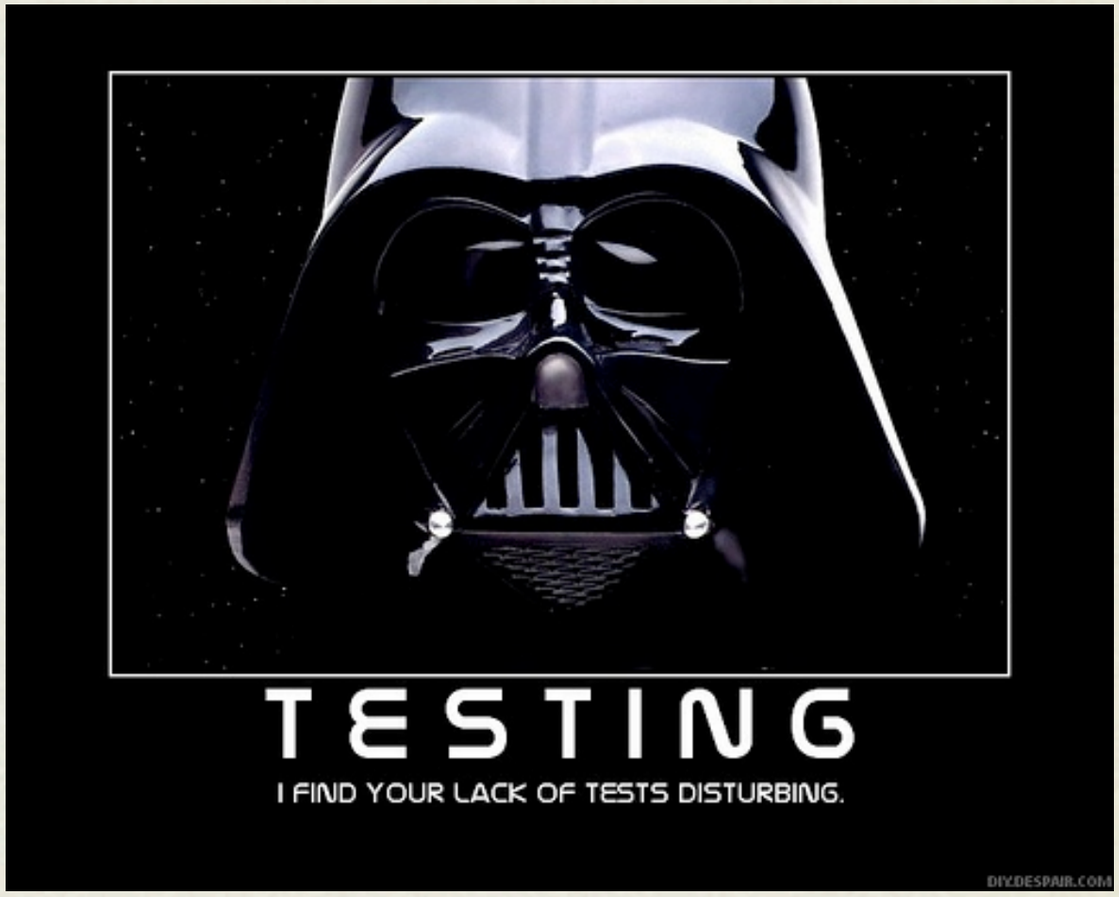
- Testing State
  - Does it have the expected values after execution?
    - Usually via “assertXXX”
- Verifying Behavior
  - Did the Method Under Test do what I expected?

OBSTACLES TO TESTING

- Method under test:
  - Has void return type
  - Runs slow
  - Has complicated dependencies (IE: EJB, Servlet APIs, Persistence APIs)
- Hard to Create scenarios (Network Exception)

TYPICAL SOLUTIONS

- Run Integration Tests
- In container testing
- Roll your own “stub”
- Don’t test



DON'T BE THESE GUYS

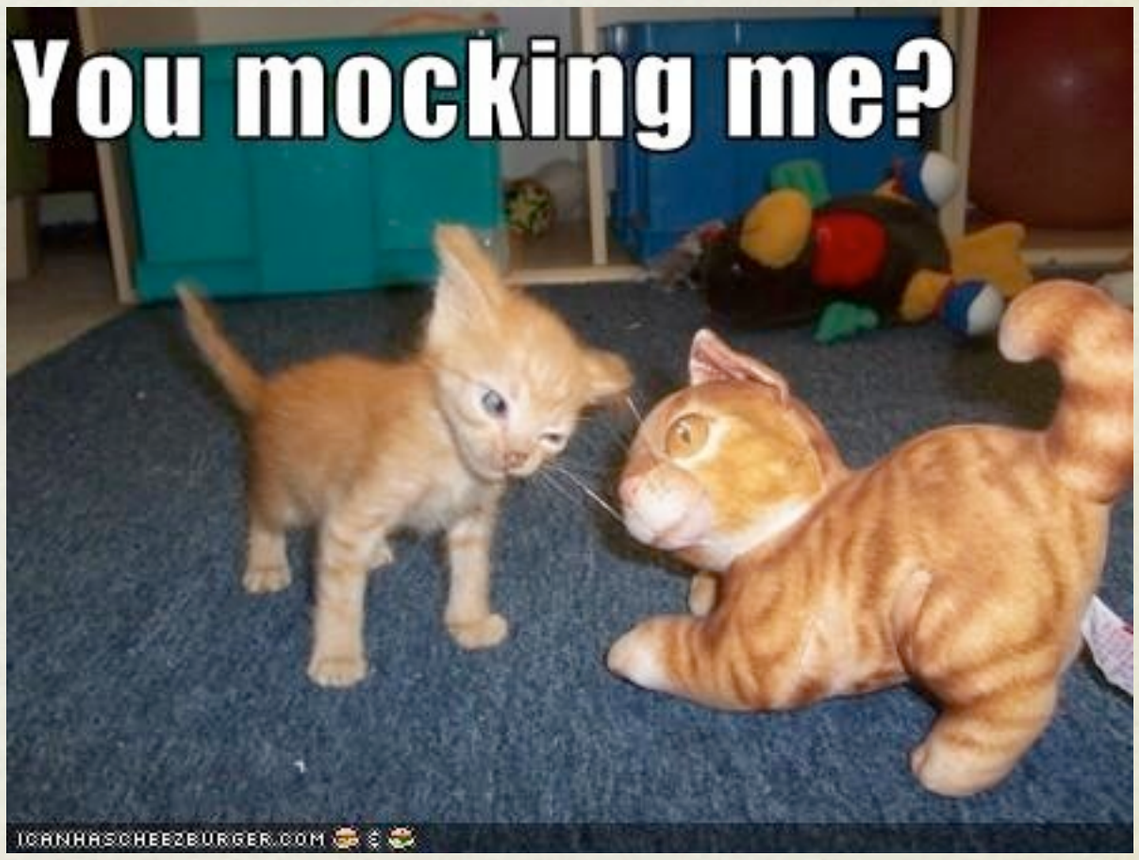
TERMINOLOGY

- test double - pretend object used for testing (encompasses all of the below)
- dummy - object passed around but not used
- fake - simple working implementation of a dependency
- stub - canned answers to calls within tests.
- mock - objects pre-programmed with expectations.
  - Used to verify “behavior”

http://xunitpatterns.com/Test%20Double.html

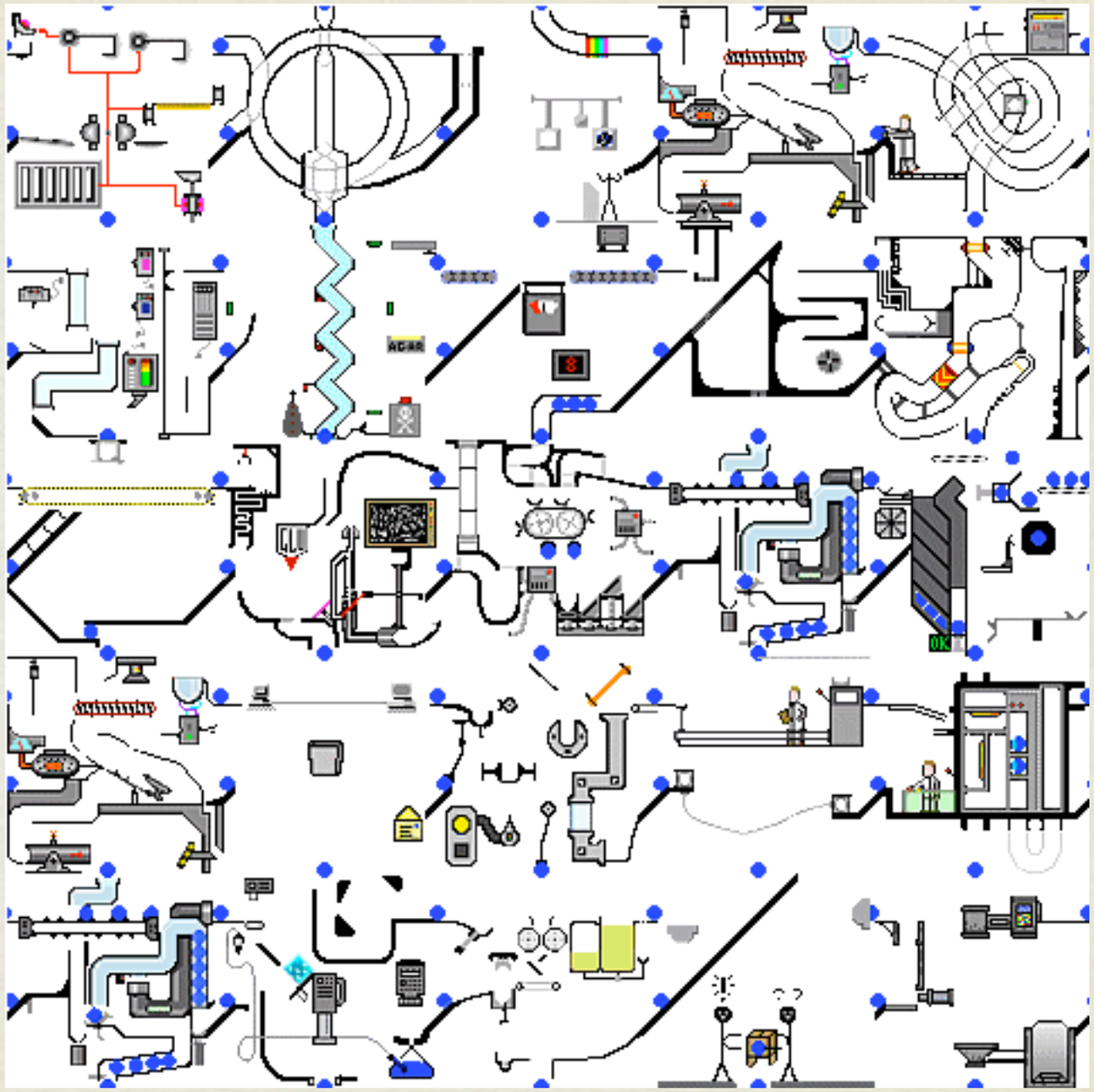
STUB VS. MOCK

- Stubs
  - Preserve and test state
  - You write the implementations.
- Mocks
  - Simulate Behavior
  - Validate behavior



BEHAVIOR MATTERS

- How many of you have worked with a system like this?
- How many of you have seen a method like this?
- Did you test it?
- Why not?



WHY MOCK: ISOLATION

- Application is complex
- Real object may be slow
- Real object is difficult to setup
- Writing isolated tests is hard

WHY MOCK: PROJECT CONSTRAINTS

- Real implementation not available
  - 3rd party code
    - Not yet developed
- Limited time for test development
- Fluid Requirements

MOCKS MAKE YOU THINK

- I have to create a lot of mocks to test.
  - Class / method has too many responsibilities?
- My Mocks need mocks
  - Is the method too dependent on internal details?

MOCKS MAKE YOU THINK

- I can’t mock that dependency
- Does your class have tightly bound dependencies, instead of dependency injection?
- Are statics being used?

MOCKRUNNER

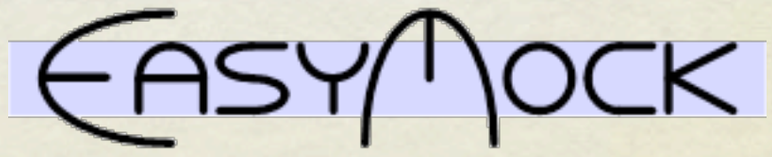
- Does not require a container!
- Supports JavaEE 1.3 - 5
  - Servlets, filters, tags, JNDI, JMS, JDBC, EJBs
  - Does not support EJB 3.0
- Supports Struts 1.1 - 1.3
- Manual Dependency Management :-(

SOAP-UI "MOCKS"

- Generate web service stub from WSDL
  - Note: This is a stub.. not a mock!
- Use Groovy to script responses
- Export as WAR file.
  - These seem to work well for testing/code validation
  - Does not hold up well under extended load!

EASYMOCK

- The original dynamic mocking framework.
- Active development
- v3.0 added many features that addressed it's shortcomings when compared with Mockito.
  - Extend EasyMockSupport (replayAll(), verifyAll())
- Hamcrest argument matchers via bridge library
- Good documentation



MOCKITO

- Next Generation mocking framework
- Already supports Hamcrest for argument matchers.
- No replay
- Claims to produce more “readable” (BDD style) code and failures
- Great documentation with examples

EASYMOCK VS MOCKITO

- Easymock
  - All interactions must be set as expectations
    - Explicit “replay” step
- Mockito
  - Mocks are “nice” by default (verify what you want)
  - No replay

POWERMOCK

- Extends EasyMock and Mockito
- Uses custom classloader
- Mock: static methods, constructors, final methods/classes, and private methods

ASIDE: HAMCREST

- Library of Matchers
  - contains(), closeTo(), endsWith(), equalTo(), typeCompatibleWith()

- Makes Assertions and Mock argument matchers more literate
- Works directly with Mockito argument matchers
- EasyMock supported via a bridge Matcher

http://code.google.com/p/hamcrest/

NICE VS STRICT MOCKS

- Nice Mocks:
  - Provides default behaviors/returns if none defined for a method
    - IE: nulls for Objects, 0 for numbers
- Strict Mocks:
  - Undefined/expected behavior fails the test
  - Variant: Order matters

STRICTS ARE BETTER(??)

- Developer is required to understand the method being tested
- Creates the perception of a “brittle” test case.
  - This brittleness is a positive
  - Failure is success!
- Nice Mocks allow tests to pass when they shouldn’t.

Strict Mock Semantics is the Only Way to Mock: http://bit.ly/bj4dSF

MOCK TESTING FLOW

- Create your mocks
- Define Expectations
- Reset/Replay the Mock
- Run your test
- Verification

CREATE MOCKS

- Often done in @Before or setUp() method

```
public class AuthenticationFilterTest extends EasyMockSupport {

    private AuthenticationFilter filter;
    private AuthenticationProvider mockAuthProvider;
    private FilterConfig mockFilterConfig;

    @Before
    public void setUp() throws Exception {
        mockFilterConfig = createMock(FilterConfig.class);
        mockAuthProvider = createMock(AuthenticationProvider.class);

        filter = new AuthenticationFilter(mockAuthProvider);
    }
....
}
```

SET EXPECTATIONS

- Method and Input Parameters
- Outcomes
  - Return values
  - Exceptions
- Invocation Count

```
expect(mockFilterConfig.getInitParameter(eq("authenticationURL")))
        .andReturn(expectedURL);

expect(mockFilterConfig.getServletContext())
        .andThrow (new ServletException("Something broke!"));

expect(mockFilterConfig.getInitParameter(anyObject(String.class)))
            .andReturn(expectedURL)
            .times(1,4);
```

REPLAY/TEST/VERIFY

- Replay - Tell the mock you're ready to test
- Run your Test
- Traditional Asserts and Mock Behavior Validation

```
// Prepare mocks for validation
replayAll();

// Execute test
filter.init(mockFilterConfig);

// Traditional JUnit Assertion (with Hamcrest Matcher)
assertThat(filter.getAuthenticationURL(),
           equalTo(expectedURL));

// Validates expected behavior of mocks occurred.  If getInitParam() on the
// mockFilter did not get called this would fail.
verifyAll();
```

SPYS / PARTIAL MOCKS

- Real methods on class are used unless explicitly mocked
- Useful for:
  - 3rd party libraries
  - Interim refactoring of legacy code
- The need to do this more often than not suggests design issues in the code.

BLACK OR WHITEBOX?

- Traditional Testing tests input/output of publics
  - How do you unit test a void?
- Verifying branching logic in private methods is challenging from the public method.
- Single public method calling multiple privates can be hard to verify.

http://www.quora.com/Should-you-unit-test-private-methods-on-a-class

TESTABLE DESIGN

- True? “Testable Design is Good Design”
  - Most tools don’t support testing private methods or final classes.
    - Should test tools govern your design?
    - Should you write tests to safely refactor code or refactor code to be able to write tests?
- Perhaps: “Good Design is always Testable”
  - http://blog.jayway.com/2009/04/01/questioning-

LET'S LOOK AT SOME CODE

- Simple Mock Setup:
  com.ctv.mocksamples.servlet.UserPrincipalRequestWrapperTest

- Basic Behavior/Assertions:
  com.ctv.mocksample.servlet.AuthenticationFilterTest

- Strict vs Loose:
  com.ctv.mocksample.servlet.AuthenticationFilterTest

- PowerMocking Statics: com.ctv.model.UserTest

- PowerMocking privates: com.ctv.model.OrderPowerMockTest

JAVA FRAMEWORKS

- Mockrunner: http://mockrunner.sourceforge.net/
- SOAP-UI: http://soapui.org/
- EasyMock: http://easymock.org/
- Mockito: http://mockito.org/http://code.google.com/p/mockito/
- Powermock: http://code.google.com/p/powermock/

.NET FRAMEWORKS

- Rhino Mocks:
  - http://www.ayende.com/projects/rhino-mocks.aspx
- EasyMock.net
  - http://sourceforge.net/projects/easymocknet/
- NMock
  - http://www.nmock.org/

RESOURCES

- “Mocks aren’t Stubs” - Martin Fowler
  - http://www.martinfowler.com/articles/mocksArentStubs.html
- Questioning “testable design”
  - http://blog.jayway.com/2009/04/01/questioning-testable-design/
- “Testing the Entire Stack” - Neal Ford
  - http://nealford.com/downloads/conferences/2010/Testing_the_Entire_Stack(Neal_Ford).pdf

RESOURCES

- Java Mocking Framework Comparison
  - http://www.sizovpoint.com/2009/03/java-mock-frameworks-comparison.html
- EasyMock / Mockito Comparisons
  - http://code.google.com/p/mockito/wiki/MockitoVSEasyMock
  - http://blog.octo.com/en/easymock-facts-fallacies/

RESOURCES

- xUnit concepts and definitions
  - http://xunitpatterns.com/Test%20Double.html
- Strict vs Nice Mocks
  - Pro Strict: http://bit.ly/bj4dSF
  - Design vs Maintainability:
    - http://stackoverflow.com/questions/2864796/easymock-vs-mockito-design-vs-maintainability

PRESENTATION CODE

- You can pull down the code from this presentation on Github
  - https://github.com/mpegram3rd/testing-with-mocks