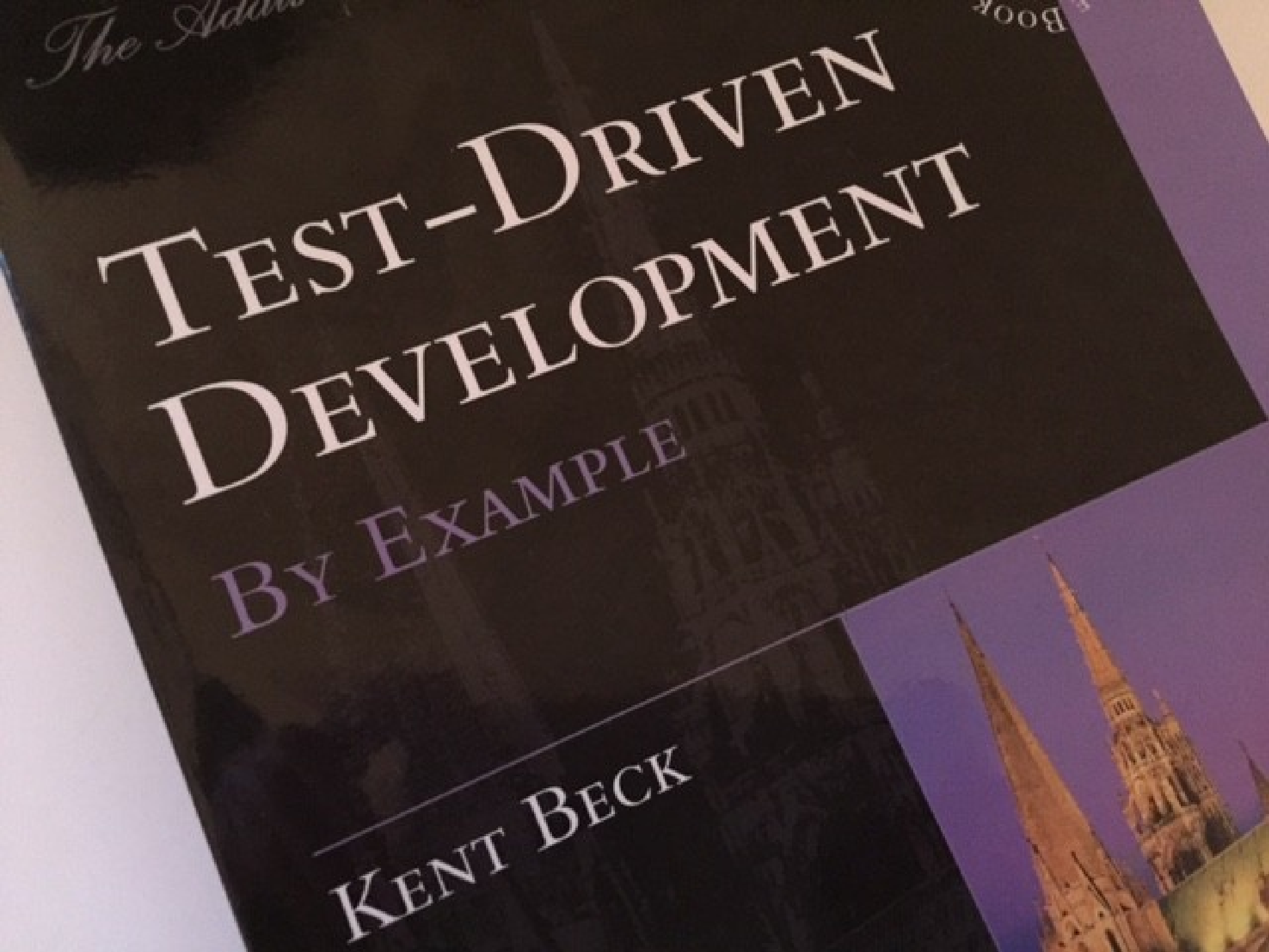
Test Driven Development

Daniel Hinojosa


Test-Driven Development
By Example
Kent Beck

The Process

“

1. Quickly add a test.
2. Run all tests and see the new one fail.
3. Make a little change.
4. Run all tests and see them all succeed.
5. Refactor to remove duplication.

”

Kent Beck – Test Driven Development By Example 2003

“

1. Write a failing test.
2. Write code to make it pass.
3. Repeat steps 1 and 2.
4. Along the way, refactor aggressively.
5. When you can't think of any more tests, you must be done.

”

Neal Ford – Evolutionary Architecture and Emergent Design 2009

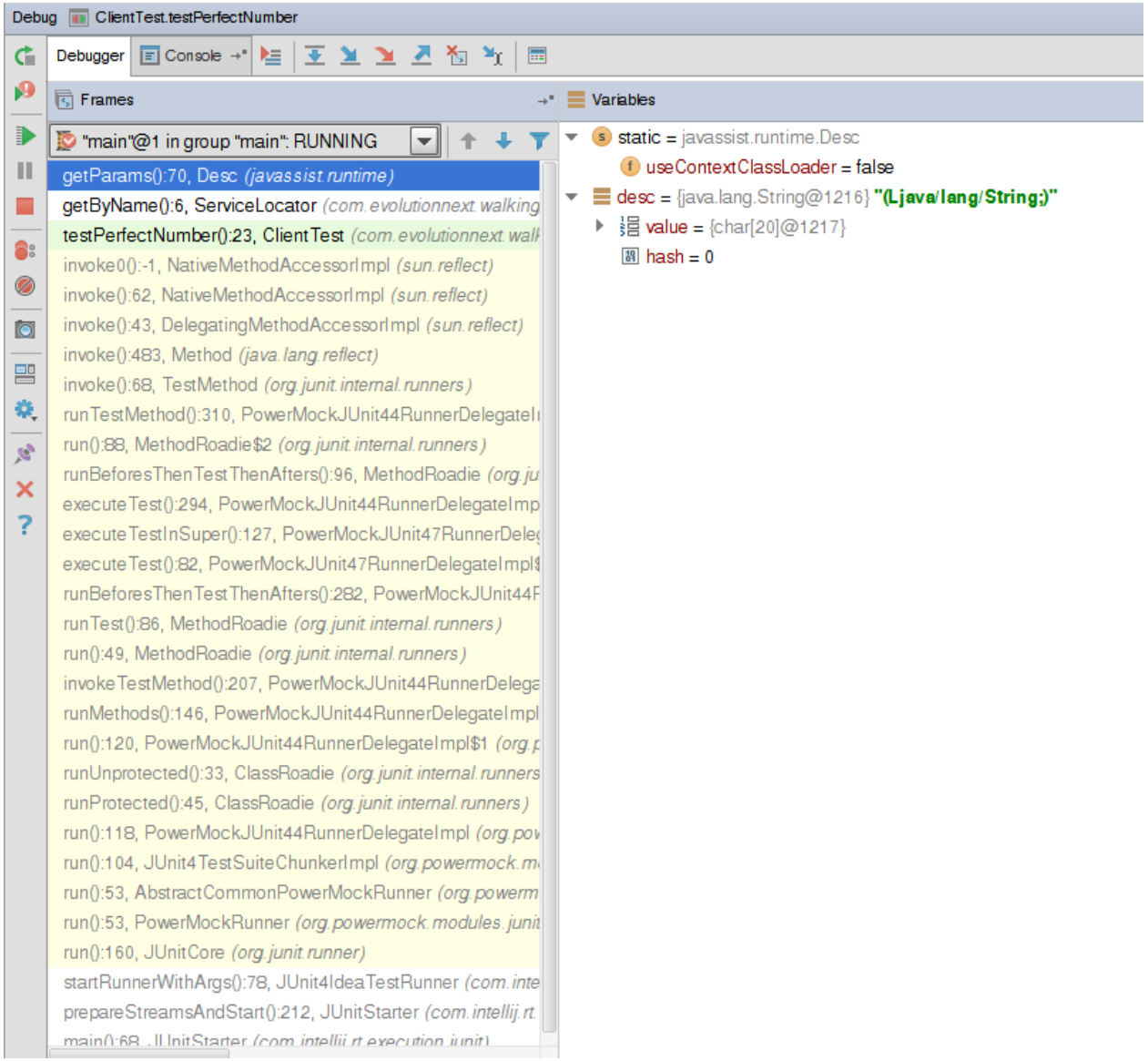
Benefits

- Promotes design decisions up front
- Allows you and your team to understand your code
- Model the API the way you want it to look
- Means of communicating an API before implementation
- Avoids Technical Debt
- Can be used with any programming language

"Game"ifying Development

Consider each fail test a challenge

Less Debugging!



Disadvantages

- People find it difficult and unintuitive at first
- Requires team investment
- Many do not see the advantages until it is too late

Bob Martin's Three TDD Laws

“

You may not write production code until you have written a failing unit test.

”

Bob Martin – Clean Code 2008

“

You may not write more of a unit test than is sufficient to fail, and not compiling is failing.

”

Bob Martin – Clean Code 2008

“

You may not write more production code than is sufficient to pass the currently failing test.

”

Bob Martin – Clean Code 2008

“

You may not write production code until you have written a failing unit test.

”

Bob Martin – Clean Code 2008

Adopting TDD

Adopting TDD As An Individual

- Practice Makes Perfect
- TDD every new method, class, or function
- Contribute to an open source project for fun
- Use testing when learning a new language!

* You should learn a new language every year anyway

Adopting TDD As A Team

- For Green Field Projects
  - Start Immediately!
  - Prevent Technical Debt
- For Brown Field Projects
  - Adopt "Testing Thursdays or Fridays" if affordable
  - Powermock if necessary*

* Powermock, though it is a great product, it is a sign of bad design

Measuring and Monitoring TDD

- Use Code Coverage to Check That Code is being tested
  - Cobertura
  - Emma
- Employ Pair Programming
- Employ Code Review
  - Non-TDD code is easy to spot

Levels of Testing

Unit Testing

- All TDD is Unit Testing
- Isolated
  - No calls to file system, database, network
  - Other objects and services are either
    - Mocked
    - Replaced with a Function (Java 8)

Unit Testing (Continued)

- Fast with Instant Feedback
- Bug Replication
- Done by the Developers, not QA

Integration Testing

- Testing already unit-tested (TDD) tests with each other
- Testing separate team developed components
- Done by Developers, Perhaps other Software Testers

Acceptance Testing

- Stakeholder & Developer Collaborative Testing
- Integration Test of Services
- Under the UI Layer
- Cucumber-like Frameworks
- Done by Developer, Technical Manager, other testing teams, or maybe QA

System Testing

- How does the system test as a whole?
- Does it meet the entire criteria including UI
- Generally done by QA or other testing teams

Security Testing

- Is the system secure
- Can it be hacked or compromised
- No holds barred testing
- Results in creation of system tests and Unit Test to plug up
- Done by developers, technical managers, testers, or QA

Load Testing

- Can the system as a whole handle extreme demand?
- Typically employs Load Testing Software like Apache JMeter
- Done typically by Developers, Testers, QA

Robustness Testing

- Testing whether a system in general can cope with errors
- Ability to continue to operate normally despite failure
- At times, fault injection is performed to cause failure

Keep in Mind...

Advanced Testing Levels Creates unit Tests!

Infrastructure Changes Required

- TDD Adherence among all developers
- Continuous Integration Server:
  - Performs unit testing every hour
  - Breaks the build if agreed testing metrics are not met

Testing Libraries On The JVM

JUnit

- Kent Beck
- Original Testing Framework on the JVM
- Most popular
- Plugins easy available for Eclipse and IntelliJ

TestNG

- Developed by Cedric Beust
- Multiple Features for Testing
  - Groups (also now available in JUnit)
  - Providers
  - Ordered Testing
  - JUnit Integration

Cucumber JVM

- Specification Testing Framework
- Used to primarily to test features demanded by stakeholder
- Non-Programmer Readable Style Tests
- Can be used for Unit Testing

Code Coverage Libaries On The JVM

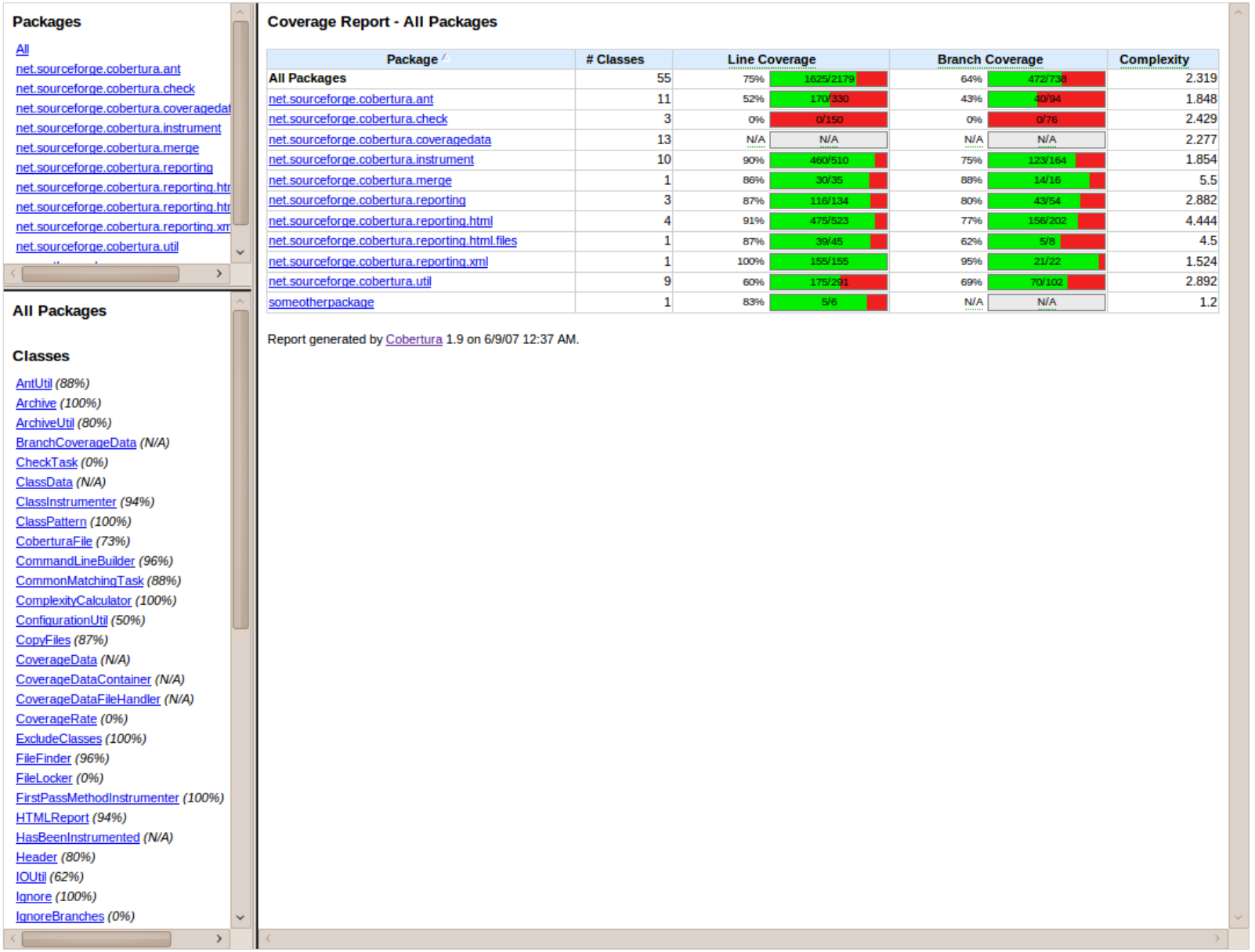
What is Code Coverage?

- Analyzes what production code has been covered by Unit Testing
- Line Coverage tests which line have been covered by tests
- Branch Coverage tests if all conditions have been tested in your `if`, `else`, `else`, `while`, `do`
- Cyclomatic Complexity
  - Algorithm to determine code complexity
  - Aim for 5 (even if the max should be 10)

How does code coverage work?

- Code that is compiled is then instrumented at byte code level
- Detects when a line of code is process by thread

Example Coverage Report



Packages

All
net.sourceforge.cobertura.ant
net.sourceforge.cobertura.check
net.sourceforge.cobertura.coveragedat
net.sourceforge.cobertura.instrument
net.sourceforge.cobertura.merge
net.sourceforge.cobertura.reporting
net.sourceforge.cobertura.reporting.htr
net.sourceforge.cobertura.reporting.htr
net.sourceforge.cobertura.reporting.xm
net.sourceforge.cobertura.util

All Packages

Classes

AntUtil *(88%)*
Archive *(100%)*
ArchiveUtil *(80%)*
BranchCoverageData *(N/A)*
CheckTask *(0%)*
ClassData *(N/A)*
ClassInstrumenter *(94%)*
ClassPattern *(100%)*
CoberturaFile *(73%)*
CommandLineBuilder *(96%)*
CommonMatchingTask *(88%)*
ComplexityCalculator *(100%)*
ConfigurationUtil *(50%)*
CopyFiles *(87%)*
CoverageData *(N/A)*
CoverageDataContainer *(N/A)*
CoverageDataFileHandler *(N/A)*
CoverageRate *(0%)*
ExcludeClasses *(100%)*
FileFinder *(96%)*
FileLocker *(0%)*
FirstPassMethodInstrumenter *(100%)*
HTMLReport *(94%)*
HasBeenInstrumented *(N/A)*
Header *(80%)*
IOUtil *(62%)*
Ignore *(100%)*
IgnoreBranches *(0%)*

Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	55	75%	1625/2179	64%	472/738	2.319
net.sourceforge.cobertura.ant	11	52%	170/330	43%	40/94	1.848
net.sourceforge.cobertura.check	3	0%	0/150	0%	0/76	2.429
net.sourceforge.cobertura.coveragedata	13	N/A	N/A	N/A	N/A	2.277
net.sourceforge.cobertura.instrument	10	90%	460/510	75%	123/164	1.854
net.sourceforge.cobertura.merge	1	86%	30/35	88%	14/16	5.5
net.sourceforge.cobertura.reporting	3	87%	116/134	80%	43/54	2.882
net.sourceforge.cobertura.reporting.html	4	91%	475/523	77%	156/202	4.444
net.sourceforge.cobertura.reporting.html.files	1	87%	39/45	62%	5/8	4.5
net.sourceforge.cobertura.reporting.xml	1	100%	155/155	95%	21/22	1.524
net.sourceforge.cobertura.util	9	60%	175/291	69%	70/102	2.892
someotherpackage	1	83%	5/6	N/A	N/A	1.2

Report generated by Cobertura 1.9 on 6/9/07 12:37 AM.

Cobertura

- Open Source Code Coverage
- Spanish/Portuguese for "Coverage"
- Easy Use

Emma

- Open Source Coverage Tool
- Easy Use

JaCoCo

- Another Open Source Coverage Tool
- Easy Use

SCCM

- Open Source
- Scala based Code Coverage for Scala programs

Build Tools On The JVM

What is a build tool?

- Tool that manages a project
- Set of commands that allow to describe build a project
  - Compiling
  - Testing
  - Packaging
  - Cleaning Binaries
  - Deployment
- Any build tool can be used with TDD

Ant

- First Popular JVM Build Tool
- XML Based
- Developed By James Duncan Davidson
- Flexible
- Contains Multiple Tasks
- Disadvantage: Lots of XML and too much configuration!

Maven

- An Apache Open Source project
- Development began in 2001
- Grew out of unwieldy Ant build files for other Apache projects
- Has gone through many iterations
- Current version is Maven 3.x
- Multiple Plugins

Gradle

- Open Source
- Groovy based Build Tool
- Feature Rich and Friendly
- Contains same features as Maven
- Allow scriptability
- Multiple Plugins

Buildr

- Open Source
- Ruby Based Build Tool
- Small following
- Flexible

SBT

- Official Name: Simple Build Tool
- Scala Build Tool
- Flexible
- Out of the box is simple
- Creating your tasks can be difficult

Leinigen

- Clojure Based Build Tool
- Open Source

Lab: Setting up Project

- Set up a project in Maven called "basic-tdd"
- Set up a dependency for "junit" by searching maven central and place inside of `pom.xml`
- Set up a dependency for "junit" by searching maven central and place inside of `pom.xml`

Setting up your IDEs

Quick Note About IDES

- Integrated Development Environment
- They are tremendous for full projects
- They are terrible for simple file editing
- Two popular ones for the JVM
  - Eclipse
  - IntelliJ IDEA

* NetBeans still has some Mind Share, Not Sure it is Popular

Eclipse

- http://www.eclipse.org
- Variant Eclipse versions depending on focus (Spring STS, Jboss Tools, Scala-IDE, etc)
- Open Source
- Pluggable Features
- Most popular IDE among JVM developers

IntelliJ IDEA

- http://www.jetbrains.com/idea
- Community Edition (Free)
- Ultimate Edition (Various Pricing Packages)
- Has more keyboard shortcut bindings than Eclipse
- Pluggable Features
- Easy to Use

* NetBeans still has some Mind Share, Not Sure it is Popular

Learning Shortcuts for IDEs

Why Keyboard Shortcuts?

- You waste an enormous amount of time using a mouse
- Efficient development requires you know most if not all keyboard shortcuts
- Learn one or two keyboard shortcuts a day (It adds up)
- If you need to perform a task, look up the shortcut until it is committed to memory
- It is essential for successful Test Driven Development

Essential Shortcuts (Windows/Linux)

- `CTRL+S` - **Save**
- `CTRL+C` - **Copy**
- `CTRL+X` - **Cut**
- `CTRL+V` - **Paste**
- `CTRL+Z` - **Undo**
- `CTRL+SHIFT+Z or CTRL+R` - **Redo**
- `ALT+TAB` – **Switch Applications**

Essential Shortcuts (Mac OS X)

- ⌘+S - **Save**
- ⌘+C - **Copy**
- ⌘+X - **Cut**
- ⌘+V - **Paste**
- ⌘+Z – **Undo**
- ⌘ + SHIFT(⇧) + Z – **Redo**
- ⌘ + TAB – **Switch Applications**

Essential Eclipse Shortcuts (Linux/Windows)

- `CTRL + SHIFT + L` – **Toggle Keyboard Shortcut Help**
- `CTRL + E` - **Recent Files**
- `CTRL + M` – **Toggle Fullscreen**
- `CTRL + D` – **Delete Line**
- `SHIFT + CTRL + F` – **Format Code**
- `CTRL + 1` – **Context Help**

Essential Eclipse Shortcuts (Mac OSX)

- ⌘ + SHIFT(⇧) + L – **Toggle Keyboard Shortcut Help**
- ⌘ + E - **Recent Files**
- ⌘ + M – **Toggle Fullscreen**
- ⌘ + D – **Delete Line**
- ⌘ + SHIFT(⇧) + F – **Format Code**
- ⌘ + 1 – **Context Help**

MoreUnit for Eclipse

- Eclipse Plugin that allows you to:
  - Switch between Test and Production Easily
  - Run Tests
- **`CTRL + J` – Switch to Test/Production Code**
- **`CTRL + R` – Run the Test**

Installing MoreUnit for Eclipse

- **`Help > Eclipse Marketplace...`**
- Search for **`MoreUnit`**
- Click Install
- Accept License Agreement
- Restart Eclipse if necessary

MoreUnit

Essential IntelliJ Shortcuts (Linux/Windows)

- `CTRL + SHIFT + A` – **Keyboard Shortcut Lookup**
- `CTRL + E` - **Recent Files**
- `CTRL + SHIFT + F12` – **Maximize Screen**
- `CTRL + Y` – **Delete Line**
- `SHIFT + ALT + L` – **Format Code**
- `ALT + ENTER` – **Context Help**
- `CTRL + SHIFT+ T` – **Toggle between Test and Class**
- `CTRL + SHIFT + F10` – **Run**

https://www.jetbrains.com/idea/docs/IntelliJIDEA_ReferenceCard.pdf

Essential IntelliJ Shortcuts (Mac OSX)

- **SHIFT(⇧) + ⌘ + A – Toggle Keyboard Shortcut Help**
- **⌘ + E - Recent Files**
- **⌘ + SHIFT(⇧) + F – Toggle Fullscreen**
- **⌘ + DELETE(⌫) – Delete Line**
- **⌘ + OPTION(⌥) + L – Format Code**
- **OPTION(⌥) + ENTER(⏎) – Context Help**

https://www.jetbrains.com/idea/docs/IntelliJIDEA_ReferenceCard_Mac.pdf

Assertion Language Libraries

Standard JUnit

- `assertEquals(expected, actual)`
- `assertEquals(message, expected, actual)`
- `assertNotEquals(unexpected, actual)`
- `assertNotEquals(message, unexpected, actual)`
- `assertNull(object)`
- `assertNull(message, object)`
- `assertNotNull(object)`
- `assertNotNull(message, object)`

Standard JUnit

- `assertSame(expected, actual)`
- `assertSame(message, expected, actual)`
- `assertTrue(boolean)`
- `assertTrue(message, boolean)`
- `assertArrayEquals(expectedArray, actualArray)`
- `assertArrayEquals(message, expectedArray, actualArray)`
- `fail()`
- `fail(message)`

Standard TestNG

- `assertEquals(actual, expected)`
- `assertEquals(actual, expected, message)`
- `assertNotEquals(actual, unexpected)`
- `assertNotEquals(actual, unexpected, message)`
- `assertNull(object)`
- `assertNull(message, object)`

Standard TestNG

- `assertSame(actual, expected)`
- `assertSame(actual, expected, message)`
- `assertTrue(boolean)`
- `assertTrue(message, boolean)`
- `assertEquals(actualArray, expectedArray)`
- `assertEquals(actualArray, expectedArray, message)`
- `fail()`
- `fail(message)`

Hamcrest

- Fluent Assertion Language
- Uses Matchers to Assertions
- Allows Essentially Wildcard Matching using Matchers

Simple Hamcrest Matchers

- `assertThat(cheese, is(equalTo(smelly)))`
- `assertThat("", isEmptyString())`
- `assertThat(((String)null), isEmptyOrNullString())`
- `assertThat(cheese, is(not(equalTo(smelly))))`
- `assertThat(cheese, is(nullValue())`
- `assertThat(cheese, is(notNullValue()))`

Descriptive Testing with Hamcrest

- **describedAs**("a big decimal equal to %0",
  equalTo(myBigDecimal), myBigDecimal.toPlainString())

Comparison Hamcrest Matchers

- `assertThat(1, comparesEqualTo(1))`
- `assertThat(2, greaterThan(1))`
- `assertThat(1, greaterThanOrEqualTo(1))`
- `assertThat(1, lessThan(2))`
- `assertThat(1, lessThanOrEqualTo(1))`
- `assertThat("Foo", equalToIgnoringCase("FOO"))`
- `assertThat("  my\tfoo  bar ", equalToIgnoringWhiteSpace(" my foo bar"))`

Instance Hamcrest Matchers

- `assertThat(cheese,is(instanceOf(Cheddar.class)))`
- `assertThat(cheese, isA(Cheddar.class))`
- `assertThat(cheese, is(sameInstance(cheddar))`
- `assertThat(cheese, is(theInstance(cheddar))`
- `assertThat(Integer.class,`
  `is(typeCompatibleWith(Number.class)))`
- `assertThat(cheddar, is(any(Cheese.class)))`

Array Hamcrest Matchers

- `assertThat("foo", isIn(new String[]{"bar", "foo"}))`
- `assertThat(new Integer[]{1,2,3},`
  `is(array(equalTo(1), equalTo(2), equalTo(3))))`
- `assertThat(new Integer[]{1,2,3}, hasItemInArray(equalTo(3)))`
- `assertThat(new String[] {"foo", "bar"},`
  `hasItemInArray(startsWith("ba")))`
- `assertThat(new String[]{"foo", "bar"},`
  `contains(Arrays.asList(equalTo("foo"), equalTo("bar"))))`
- `assertThat(new String[]{"foo", "bar"},`
  `containsInAnyOrder("bar", "foo"))`
- `assertThat(new String[]{"foo", "bar"}, arrayWithSize(equalTo(2)))`
- `assertThat(new String[0], emptyArray())`

Collection Hamcrest Matchers

- `assertThat(Arrays.asList("bar", "baz"), everyItem(startsWith("ba")))`
- `assertThat(Arrays.asList("foo", "bar"), hasItem("bar"))`
- `assertThat(Arrays.asList("foo", "bar"), hasItem(startsWith("ba")))`
- `assertThat(Arrays.asList("foo", "bar", "baz"), hasItems("baz", "foo"))`
- `assertThat(Arrays.asList("foo", "bar", "baz"), hasItems(endsWith("z"), endsWith("o")))`
- `assertThat(Arrays.asList("foo", "bar"), hasSize(equalTo(2)))`
- `assertThat(new ArrayList<String>(), is(empty()))`

Map Hamcrest Matchers

- `assertThat(myMap, hasKey(equalTo("bar")))`
- `assertThat(myMap, hasKey("bar"))`
- `assertThat(myMap, hasValue("foo"))`
- `assertThat(myMap, hasValue(equalTo("foo")))`

Floating Point Hamcrest Matchers

- `assertThat(1.03, is(`**`closeTo`**`(1.0, 0.03)))`
- ```
  assertThat(new BigDecimal("1.03"),
    is(closeTo(new BigDecimal("1.0"),
       new BigDecimal("0.03"))))
  ```
  ```

String Hamcrest Matchers

- `assertThat("myStringOfNote", containsString("ring"))`
- `assertThat("myStringOfNote", startsWith("my"))`
- `assertThat("myStringOfNote", endsWith("Note"))`

Property Hamcrest Matchers

- `assertThat(myBean, hasProperty("foo"))`
- `assertThat(myBean, hasProperty("foo", equalTo("bar"))`
- `assertThat(myBean, samePropertyValuesAs(myExpectedBean))`

Wildcard Matchers

- `assertThat("myValue", allOf(startsWith("my"), containsString("Val")))`
- `assertThat("myValue", anyOf(startsWith("foo"), containsString("Val")))`
- `assertThat("fab", both(containsString("a")).and(containsString("b")))`
- `assertThat("fan", either(containsString("a")).and(containsString("b")))`
- `assertThat(cheese, is(anything())`

Fest Assert

- Fluent Assertions based on type.
- Filtering based on type provided
- Very little need to look up matchers since IDE will provide guidance
- Supplanted by AssertJ

AssertJ

- Update to Fest Assert
- Contains assertions for Guava, Joda-Time, Swing
- `import static org.assertj.core.api.Assertions.*;`

Instructor & Group Lab: Medium Difficulty Puzzle from cyber-dojo.org

Individual Lab! Fizz Buzz

Isolated Testing

Isolation

- The key to unit testing is isolation
- Mock, Stub, Dummy, or Fake dependencies
- Prefer to investigate an interface of the object you wish to inject.

Evidence of Dependencies

No evidence of a Dependency

```
public void method1() {
      new Employee("Roger", "Moore");
}
```

No evidence of a static dependency

```
public void method2() {
    Resource resource =
        ServerInstance.find("/accounts/resource");
    resource.addDeposit(3000.00);
}
```

Static Dependency Hard to Control

```
public Resource method3() {
    Resource resource =
        ServerInstance.find("/accounts/resource");
    resource.addDeposit(3000.00);
    return resource;
}
```

Full Evidence of a Dependency

```
public Resource method4(Resource resource) {
    resource.addDeposit(3000.00);
    return resource;
}
```

Full Evidence of a Dependency

```
public Employee method4() {
  return new Employee("Sean", "Connery");  //OK
}
```

Full Evidence of a Dependency

```
public List<Employee> method5() {
  return Arrays.asList(
                  new Employee("Sean", "Connery"),
                  new Employee("George", "Lazenby"),
                  new Employee("Pierce", "Brosnan"),
                  new Employee("Roger", "Moore"),
                  new Employee("Timothy", "Dalton"),
                  new Employee("Daniel", "Craig"));
}
```

Fakes

“

Objects that actually have working implementations, but usually take some shortcut which makes them not suitable for production (an in memory database is a good example).

”

Martin Fowler - http://martinfowler.com/articles/mocksArentStubs.html

Dummies

“

Objects that are passed around but never actually used. Usually they are just used to fill parameter lists.

”

Martin Fowler - http://martinfowler.com/articles/mocksArentStubs.html

Stubs

“

Stubs provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test. Stubs may also record information about calls, such as an email gateway stub that remembers the messages it 'sent', or maybe only how many messages it 'sent'.

”

Martin Fowler - http://martinfowler.com/articles/mocksArentStubs.html

Mocks

“

Mocks are ... objects pre-programmed with expectations which form a specification of the calls they are expected to receive.

”

Martin Fowler - http://martinfowler.com/articles/mocksArentStubs.html

Java 8 Functions

- Lambda Expressions are available now on Java 8
- Potential to minimize the use of mocks & stubs, save time

Java 8 Functions

“

A functional interface is any interface that contains only one abstract method. (A functional interface may contain one or more default methods or static methods.) Because a functional interface contains only one abstract method, you can omit the name of that method when you implement it.

”

Functional Interface Definition - https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

Mocking Frameworks

EasyMock

- One of the first mocking frameworks
- http://www.easymock.org
- Strict by default

JMock

- Mocking Framework
- No current release since 2012

Mockito

- Flexible Mocking Framework
- Most popular of the mocking framework
- Lenient

Instructor & Group Lab: Larger Application Development With Mocking using Mockito

Individual Lab: Finding overdue library books.

Individual Lab: Quick Small Challenges

Best Practices and Advice

Code Reviewer Guide

http://misko.hevery.com/code-reviewers-guide/

For PDF: 
http://misko.hevery.com/attachments/Guide-Writing%20Testable%20Code.pdf

Exception Handling

- One Exception can be thrown for multiple reasons
- It is best to check the messages to avoid false positives

AntiPattern: Mocks returning Mocks

- Mocks returning Mocks shows bad form
- Having more than 2 mocks can possibly show bad form
- Shows that a class is multipurpose
- Likely broke the "Single Responsibility Principle"

“

Perhaps a better rule is that we want to test a single concept in each test function. We don’t want long test functions that go testing one miscellaneous thing after another.

”

Bob Martin – Clean Code 2008

“

Every class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class. All its services should be narrowly aligned with that responsibility

”

Source: http://en.wikipedia.org/wiki/Single_responsibility_principle

Class Cohesion

- Methods should support most if not all private encapsulated member variables
- A few can go unused in some methods, but should have a very good reason to do so.
- Any private member variables that are not entirely supports by encapsulating variables should be removed or refactored

In VCS We Trust

- Commenting out code that you do think you need anymore is bad form
- If you don't need it, **delete it**
- Trust in your version control system
- Commit green and clean code constantly, so that you can recover it
- Take time or take training to know and understand your VCS very well

Fail Fast

- Definition: A fail-fast system is designed to immediately report at its interface any failure or condition that is likely to lead to failure.
- Do not hide exceptions
- Go wrong fast and upfront or go wrong in production

What about Powermock?

The naysayers

"It takes a lot time"

"In the beginning, it does, but anything worthwhile takes time. It is a great practice, and the initial time investment up front will provide faster code maintenance later."

"Mocking is kind of an inane practice"

"There should only be two mocks or less used per test. If there are more, you may have to reevaluate your design"

"Seems I am going to spend my whole life testing!"

"Test your core. Test what you believe is critical to the project. Test also what you believe will have a negative impact if given the wrong input"

"Testing in general sucks when my boss asks me to make a change"

"Don't change your class that you worked so hard on. Respect your code. There are many design patterns that you can use to change the behavior of your code. Look up Adapter Pattern, Decorator Pattern, and the Strategy Pattern"

"We weren't taught that way"

"Agreed, but we are doing more than Hello World apps now. We are also paid to maintain what we create."

Quotes from the Pros

“

As a programmer, do you deserve to feel confident?" (Can you sleep at night knowing your code works)

”

Kent Beck, Is TDD Dead? You Tube Video Series

“

The primary benefit of TDD is self testing code

”

Kent Beck, Is TDD Dead? You Tube Video Series

“

Testing extends what the compiler does, check against your domain to ensure what you are doing is accurate.

”

Daniel Hinojosa -- Yes, I am quoting myself

“

I know this sounds strident and unilateral, but given the record I don’t think surgeons should have to defend hand-washing, and I don’t think programmers should have to defend TDD.

”

Robert Martin, The Clean Coder: A Code of Conduct for Professional Programmers 2011

“

To me, legacy code is simply code without tests.

”

Michael Feathers, Working Effectively with Legacy Code 2004

Recap

Recap

- Test First.. Always
- Have an editor **and** an IDE of choice, and learn its keymap very well
- Learn your version control very well.
- Speed in TDD is key.
- Perhaps a mock can be replaced by a function!
- "Game"ify your development with testing