

# **Test Driven Development**

in Java with Daniel Hinojosa



*The Addison-Wesley*

Book

# 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

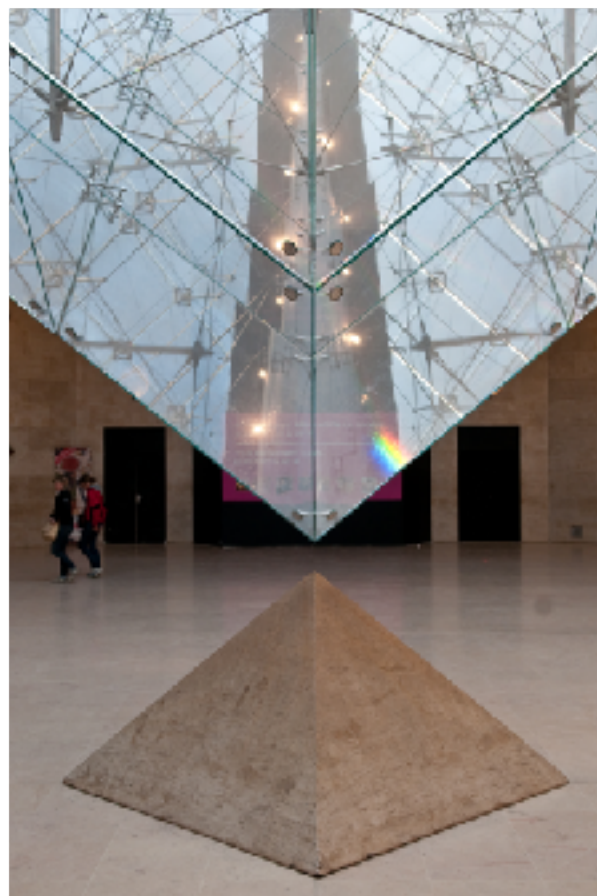
“

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.


”

# 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





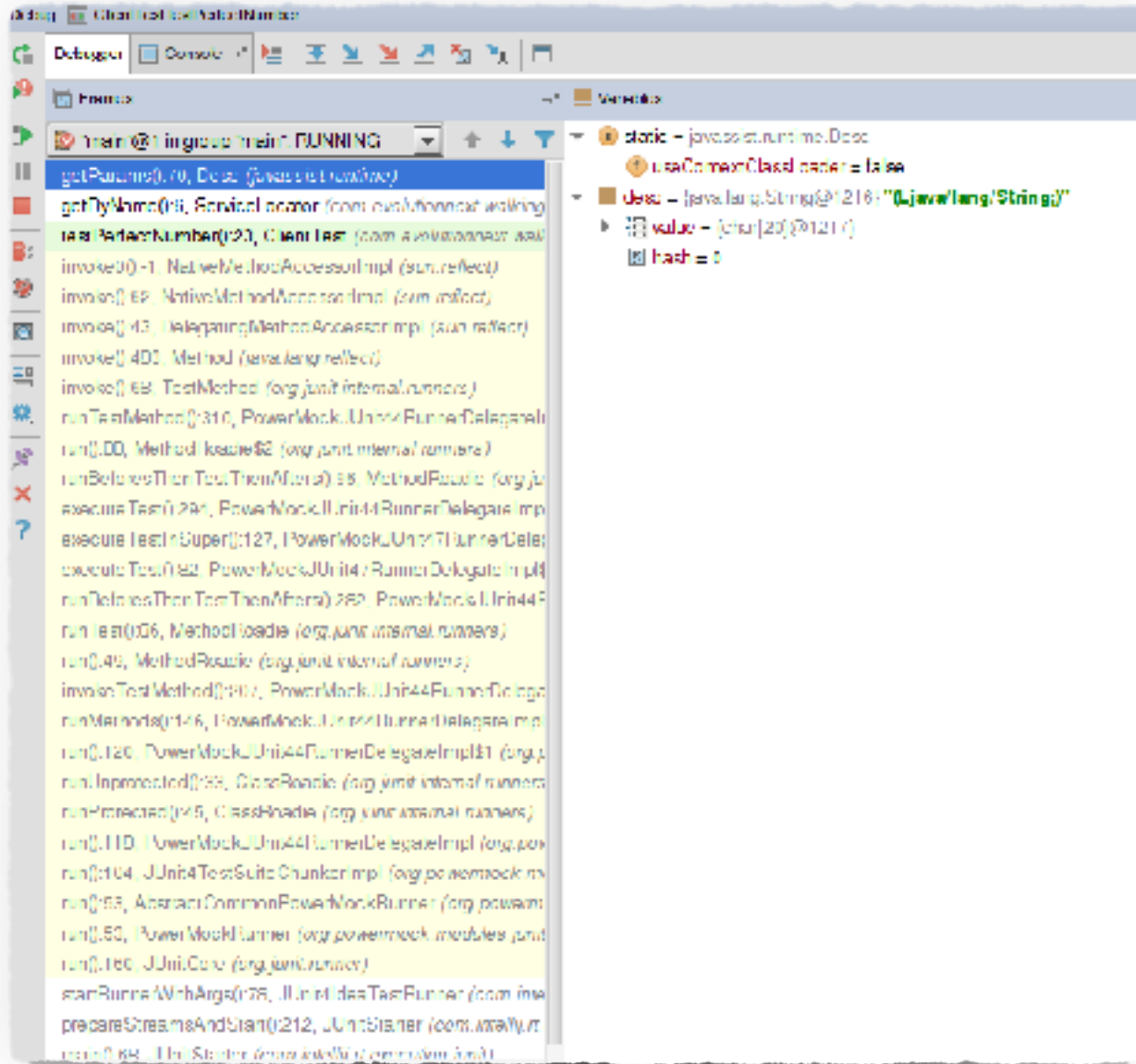


```
public class TaxCode {
    public void applyTaxAndSurcharge(Order order) {
        Money total = order.total();
        if (order.after(LocalDate.of("2001-01-01"))) {
            if (order.customer.getState().equals("FL") ||
                order.customer.getState().equals("NY"))
                order.applySurcharge(new Money(10))
                order.applyTax(new Money(total *
                    TaxWS.findTaxRate(order.customer.getState())));
            else
                order.applyTax(new Money(total));
        } else {
            order.applyTax(0)
            order.applySurcharge(0)
        }
    }
}
```

# "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.

”



# **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

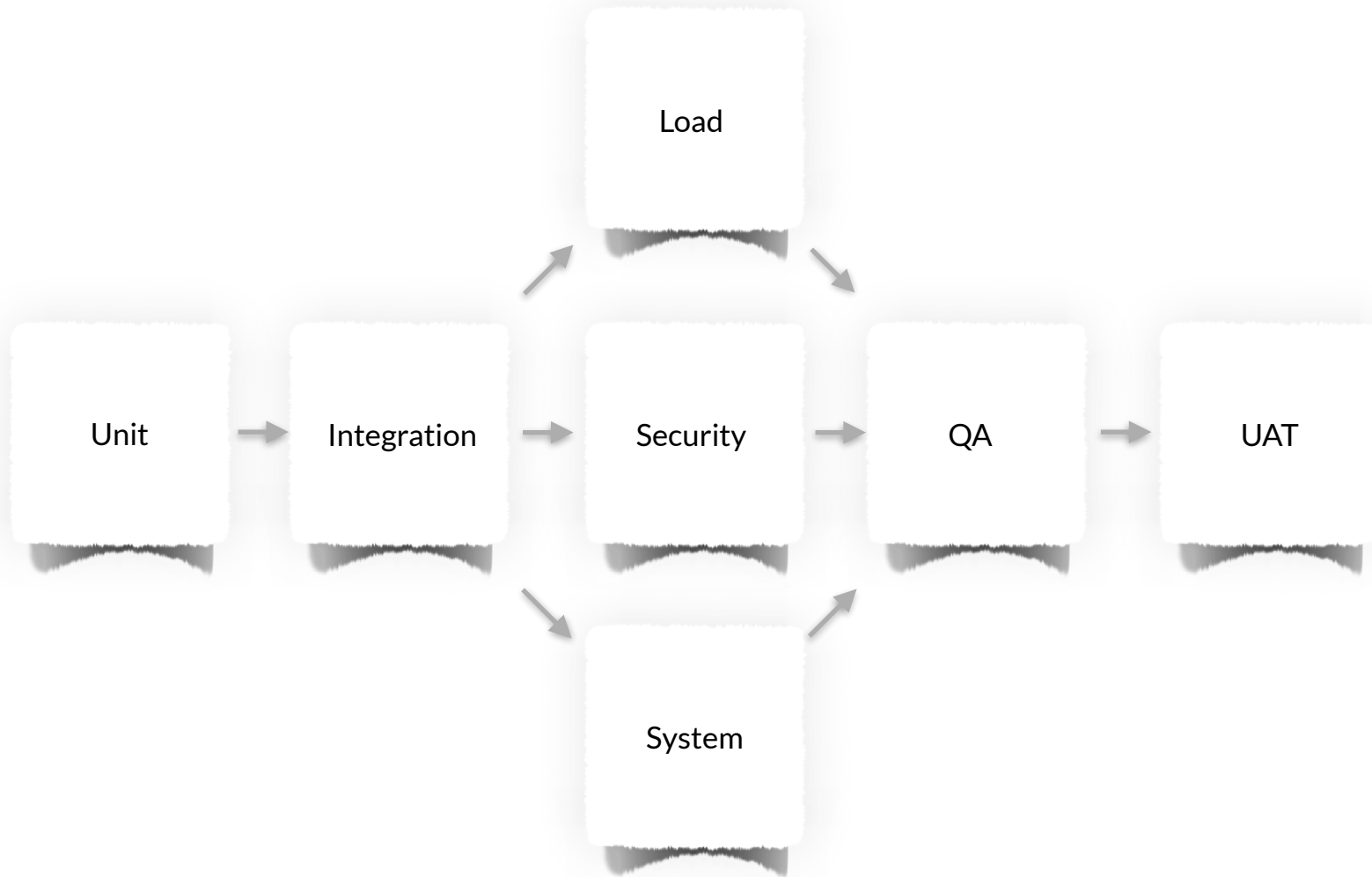
- For Green Field Projects
  - Start Immediately!
  - Prevent Technical Debt
- For Brown Field Projects
  - Adopt "Testing Thursdays or Fridays" if affordable
  - Powermock in Java if necessary\*

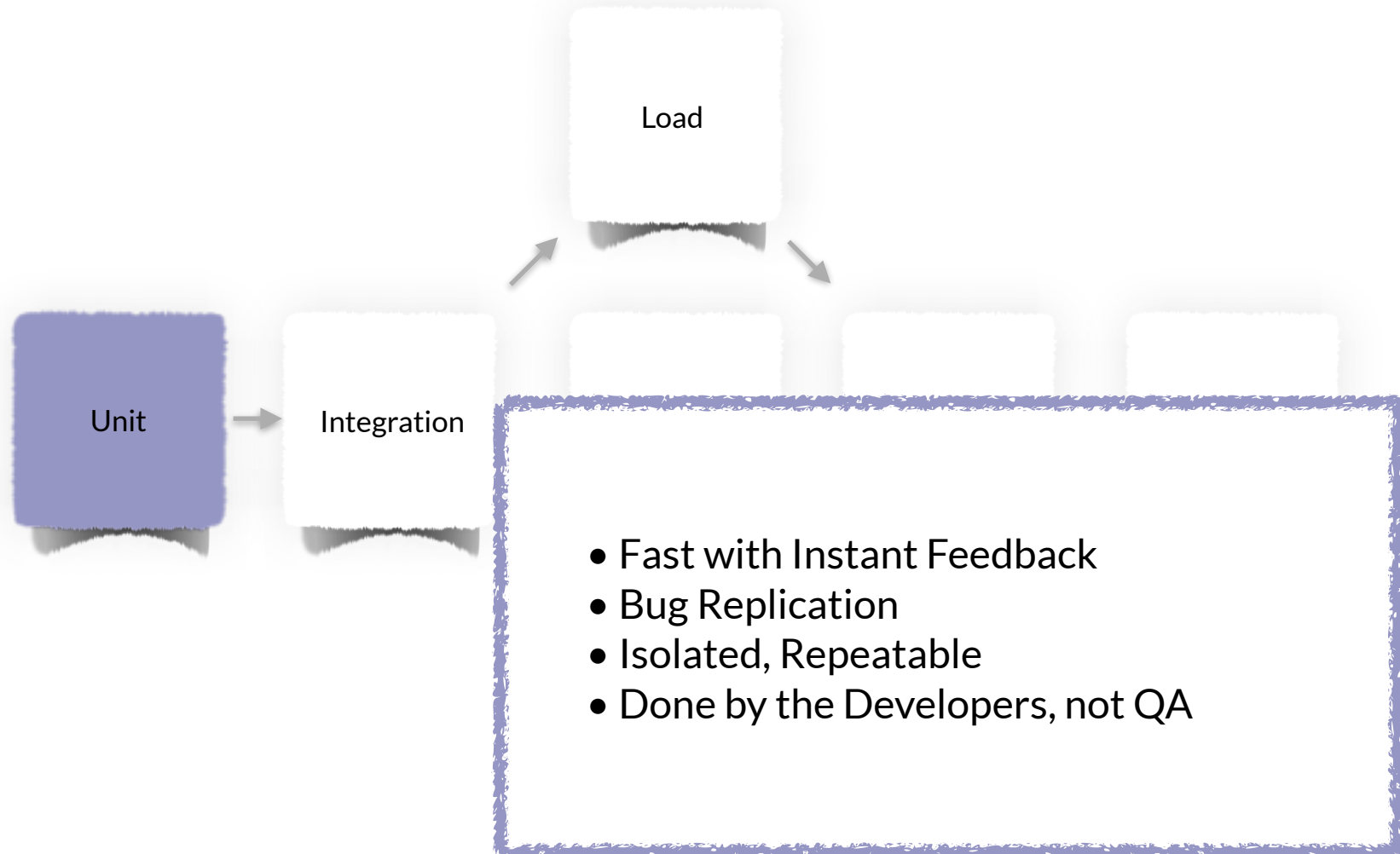
\* Powermock in Java, or any class manipulation utility is usually a sign of bad design

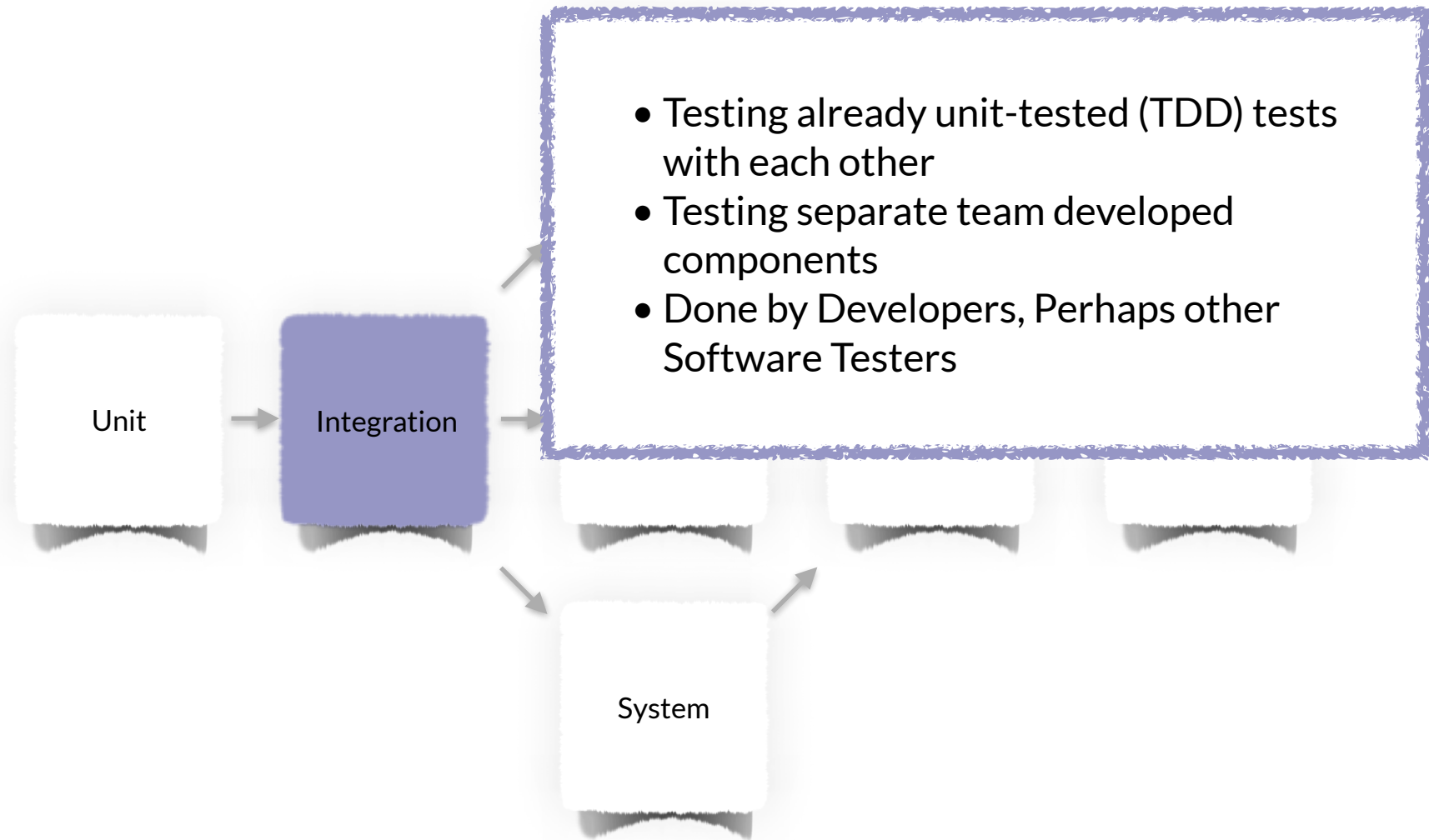
# Measuring and Monitoring TDD

- Use Code Coverage to check that code is being tested
  - Cobertura
  - Emma
  - Jacoco
- Employ Pair Programming
- Employee Code Review
  - Non-TDD Code is easy to spot

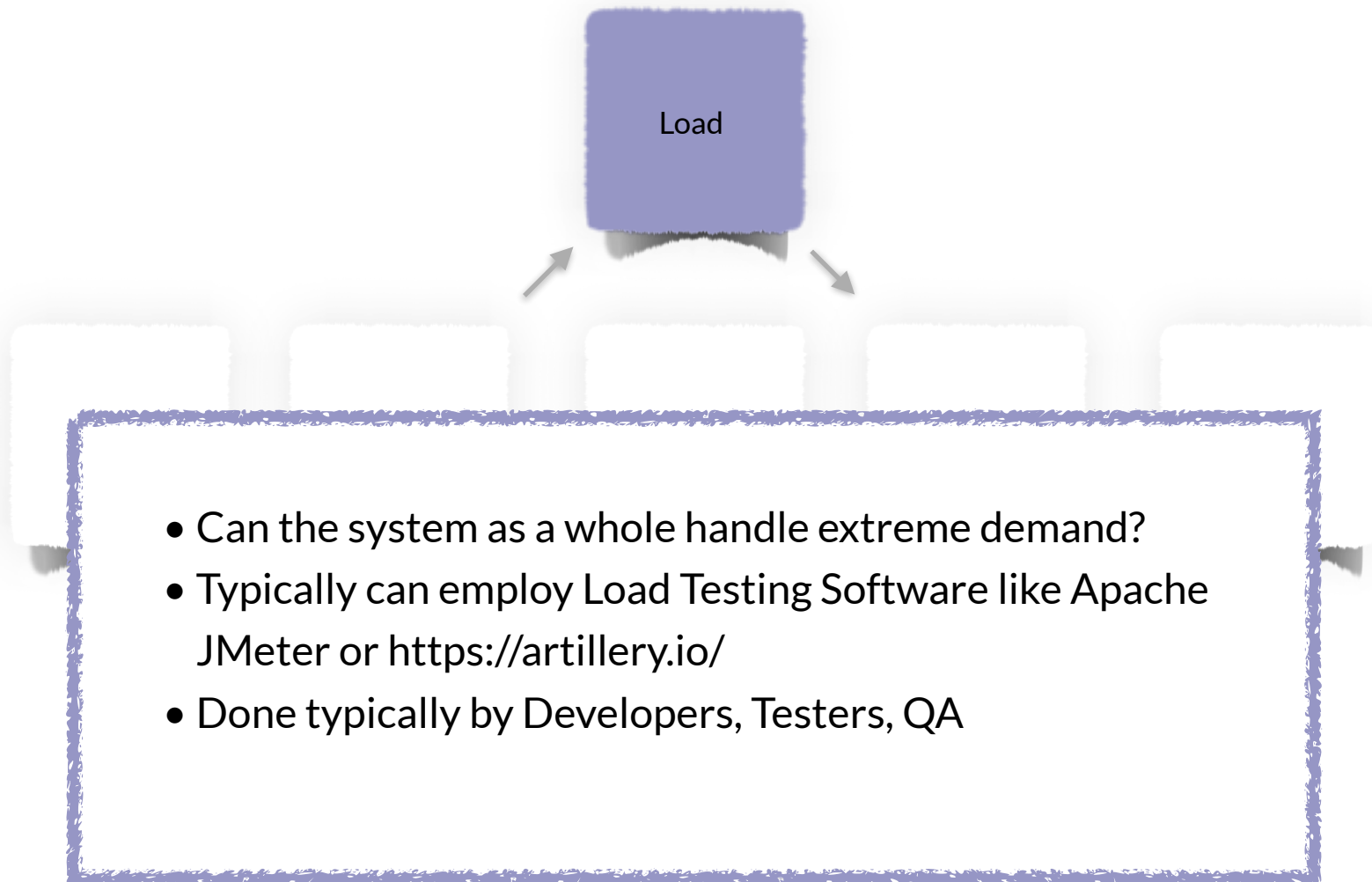
# **Levels of Testing**











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

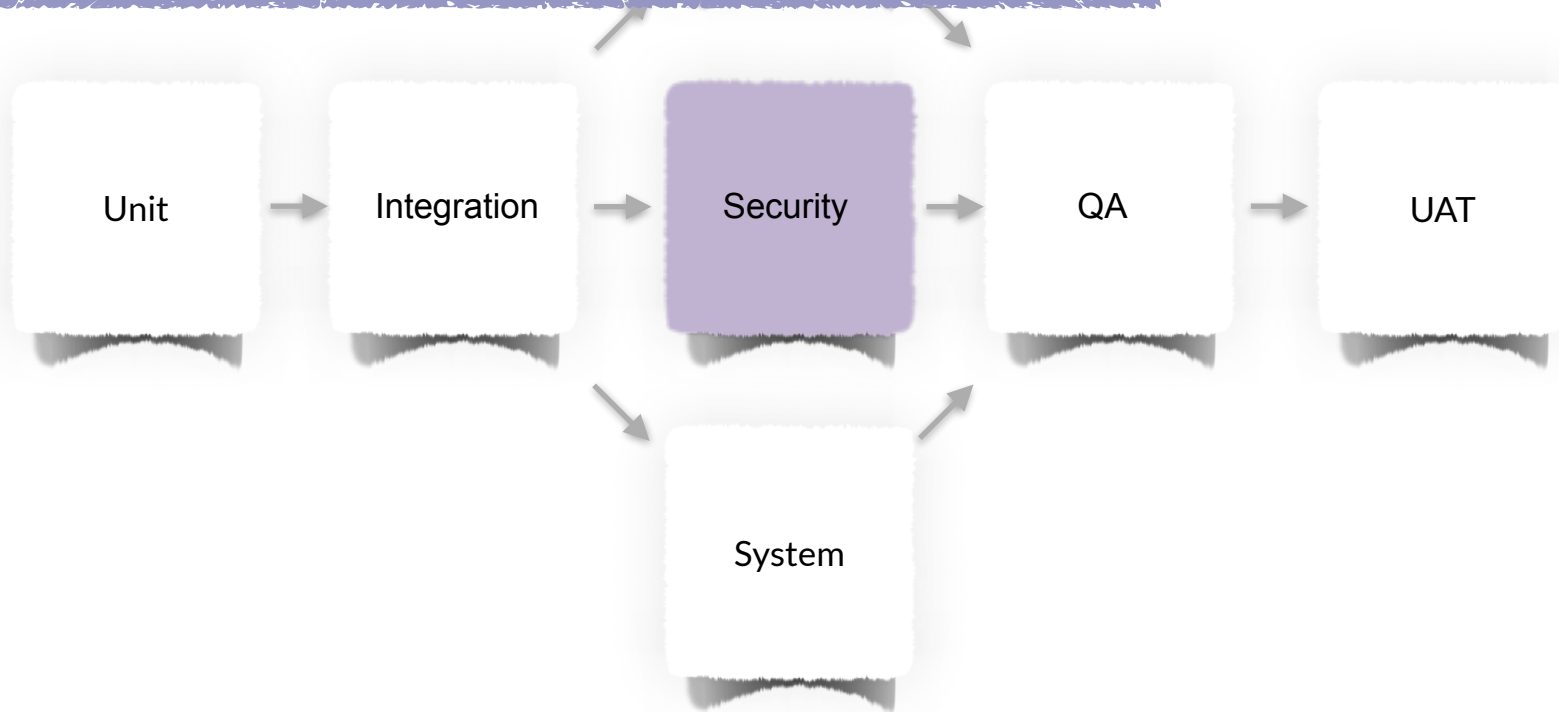
Unit

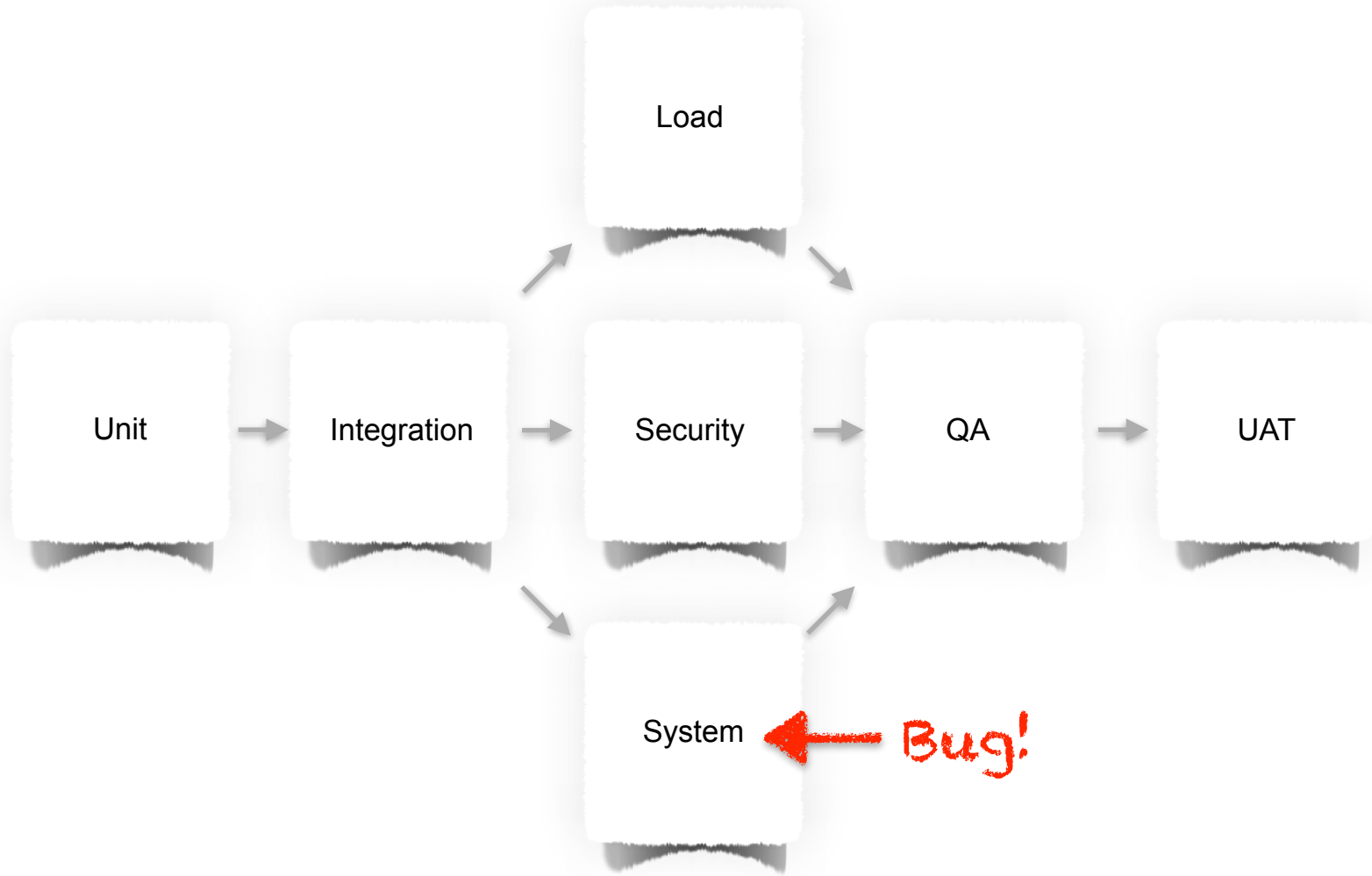
UAT

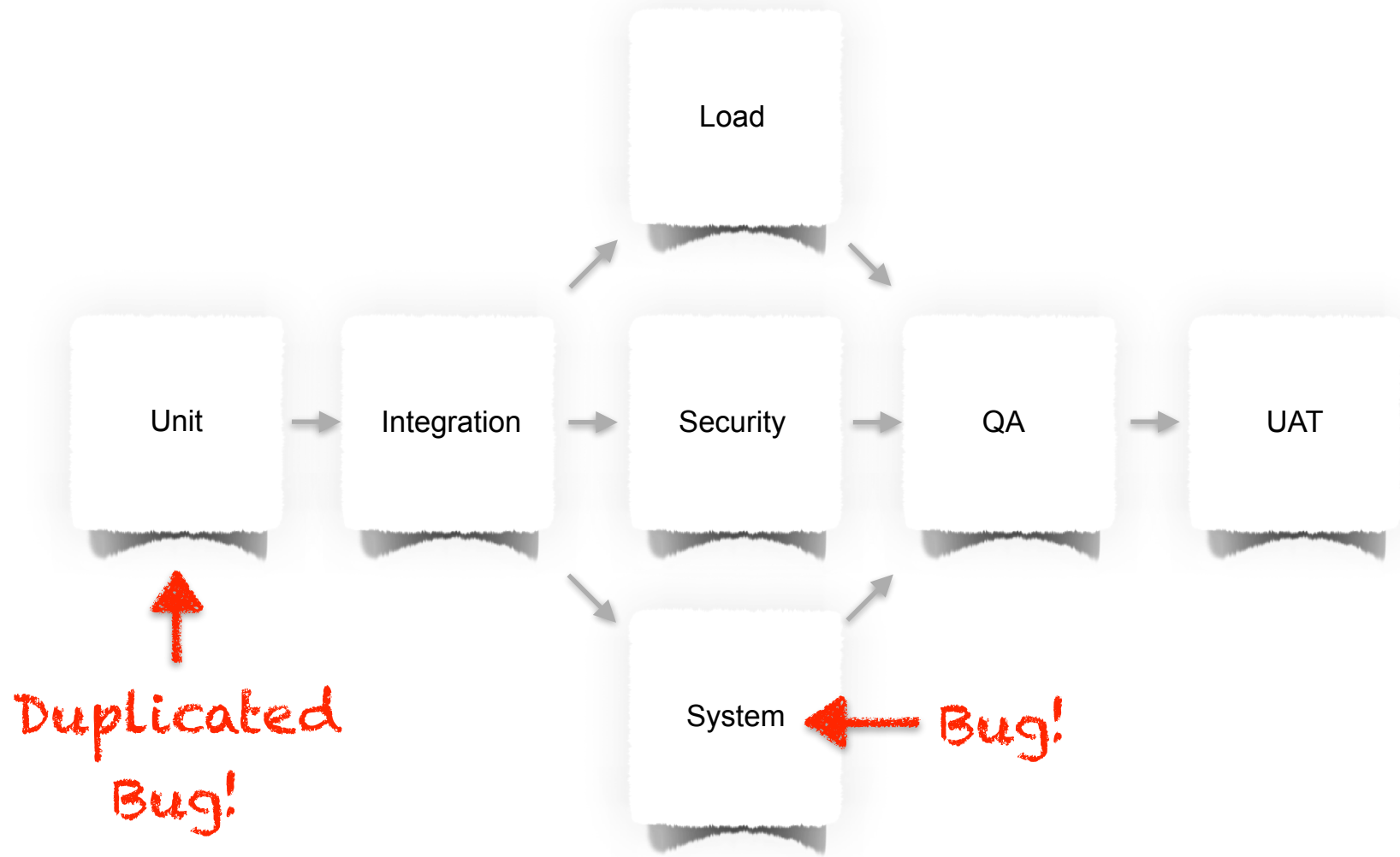
System

The diagram illustrates the relationship between different levels of testing. At the top, a large white box with a blue border contains three bullet points: 'How does the system test as a whole?', 'Does it meet the entire criteria including UI', and 'Generally done by QA or other testing teams'. Below this box, the word 'Unit' is on the left and 'UAT' is on the right. At the bottom center is a blue box labeled 'System'. Two arrows point from the 'Unit' and 'UAT' boxes towards the 'System' box, indicating that both unit and user acceptance testing contribute to the overall system testing process.

- 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







# 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



**What to test?**





[illegible]

```

#include <stdio.h>
#include <string.h>
// Exercise the symbol table mechanism of the compiler in an effort to
// take a unreasonable amount of time compiling

#define PRINT(x) { \
printf("CMD[%d]: ", x); \
printf("NAME[%s], ADDR[%s], SIZE[%s], \
FNAME[%s], FILENAME[%s], \
SYMBOL[%s]\n", \
x, \
NAME, \
ADDR, \
SIZE, \
FNAME, \
FILENAME, \
SYMBOL); \
}

// Forward declaration of repeated 0 method pointers
typedef void (*ctor_ptr) (void);
typedef void (*dctor_ptr) (void);

// Assumes two var-args: classfile(pointer), ctor, dctor
void main (var_ptr, var_ptr, ...) {
    // Fill in variable lists
    var_list_t *var_list;
    var_list_t *var_list2;
    int timesvar_arglist_len;
    var_ptr ctor_var_arglist;
    var_ptr dctor_var_arglist;
    var_ptr var_arglist;
    var_ptr var_arglist2;
    // process
    if (timesvar_arglist == malloc(sizeof(int)))
        return;
    if (timesvar_arglist2 == malloc(sizeof(int)))
        return;
    // var-args declared to install the ctor and dctor in the write places */
    return;
}

```

[illegible][illegible][illegible]

- We break things up into logical sections
- We also want to reuse those sections
- Those things must form a cohesive unit
- Single Responsibility Principle

# Production

# Test

## We test the logical sections

**What not to test?**

# What not to test?

- The main method, this is where wiring takes place
- GUIs
  - There are varying testing frameworks to test GUIs
    - Selenium Web Driver (Web, JavaScript)
    - Fest-Swing (Java Swing Applications)
    - TestFX (Java FX Applications)
    - Code that calls external services and APIs (Maybe)
- What about getters and setters? Yes\*

\* Strong Opinion

# **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

# Testing Libraries In Node/JS



# **Code Coverage Libraries On The JVM**

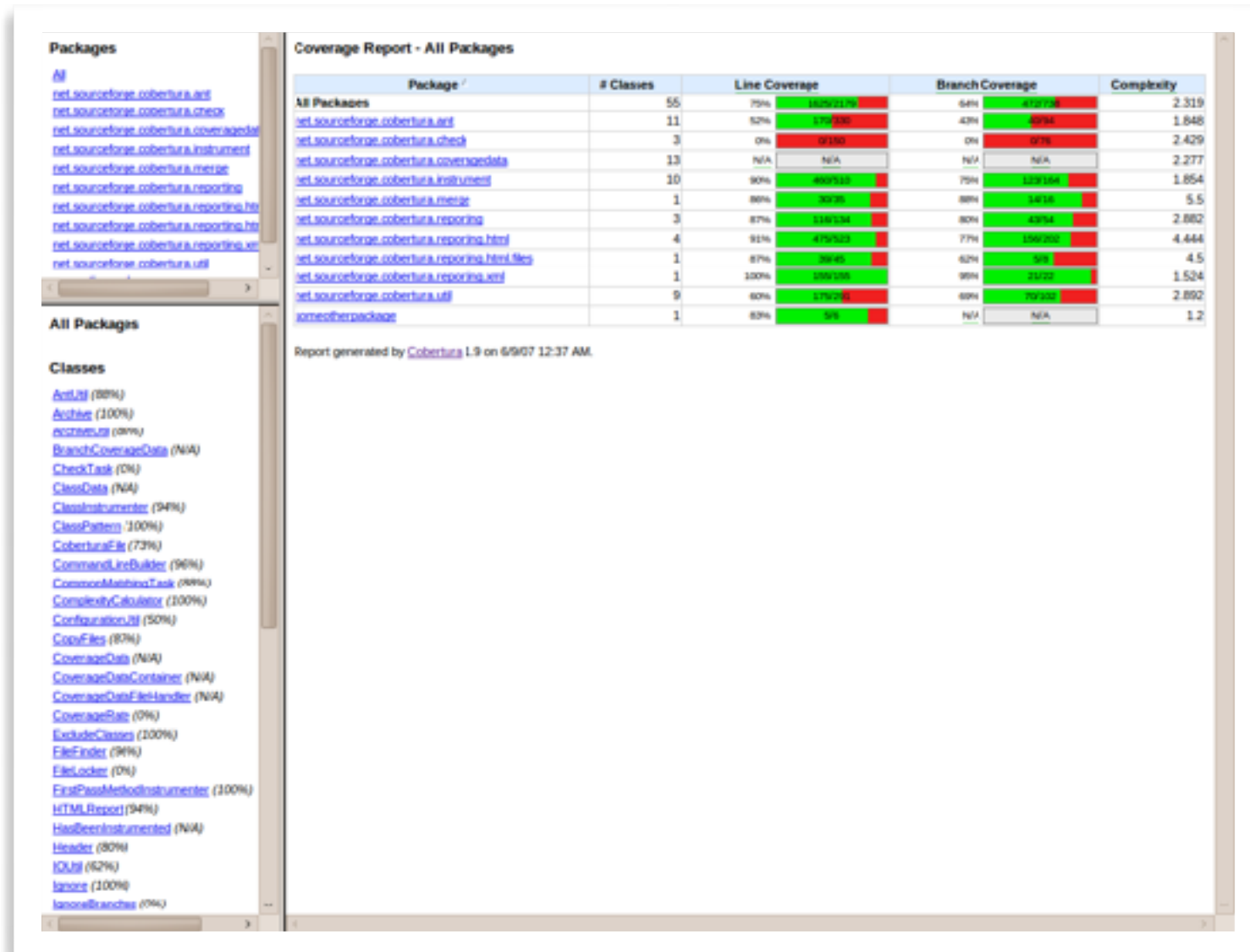
# What is Code Coverage?

- 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 JVM Coverage Report



# JaCoCo

- Another Open Source Coverage Tool
- Easy Use
- Excellent Java 8 Use

# Lab: Retrieve Project

- For Java Based Project:

<http://www.github.com/dhinojosa/nfjs-java-tdd>

Either clone or  
download project  
of your choice:

 Clone 

HTTPS

SSH

GitHub CLI

`git@github.com:dhinojosa/nfjs-java-tdd.git` 

Use a password-protected SSH key.

 Open with GitHub Desktop

 Download ZIP

# Lab: Run A Quick Test

- For Java Based Project:
  - `cd java-tdd`
  - `mvn test`

# Setting up your IDEs



# Quick Note About Java 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
- Full fledged Node/JS IDEs are since editors are usually used
  - WebStorm

\* 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



# **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**
- **CTRL** + 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



# 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\\_Mac.pdf](https://www.jetbrains.com/idea/docs/IntelliJIDEA_ReferenceCard_Mac.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](https://www.jetbrains.com/idea/docs/IntelliJIDEA_ReferenceCard_Mac.pdf)

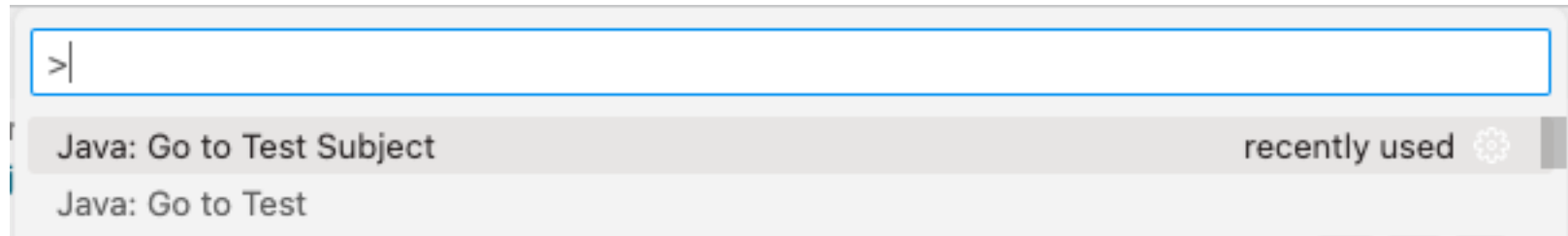


# Editor Can't Toggle?

Consider using split pane



# Visual Studio Code Toggling



# Atom Keyboard Shortcuts



<https://github.com/nwinkler/atom-keyboard-shortcuts>



# Sublime Keyboard Shortcuts



[http://docs.sublimetext.info/en/latest/reference/keyboard\\_shortcuts\\_win.html](http://docs.sublimetext.info/en/latest/reference/keyboard_shortcuts_win.html)

# Sublime Keyboard Shortcuts



[http://docs.sublimetext.info/en/latest/reference/keyboard\\_shortcuts\\_osx.html](http://docs.sublimetext.info/en/latest/reference/keyboard_shortcuts_osx.html)

# AssertJ

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

**cyber-dojo.org**

**the place to practice programming**



## setup a new practice session

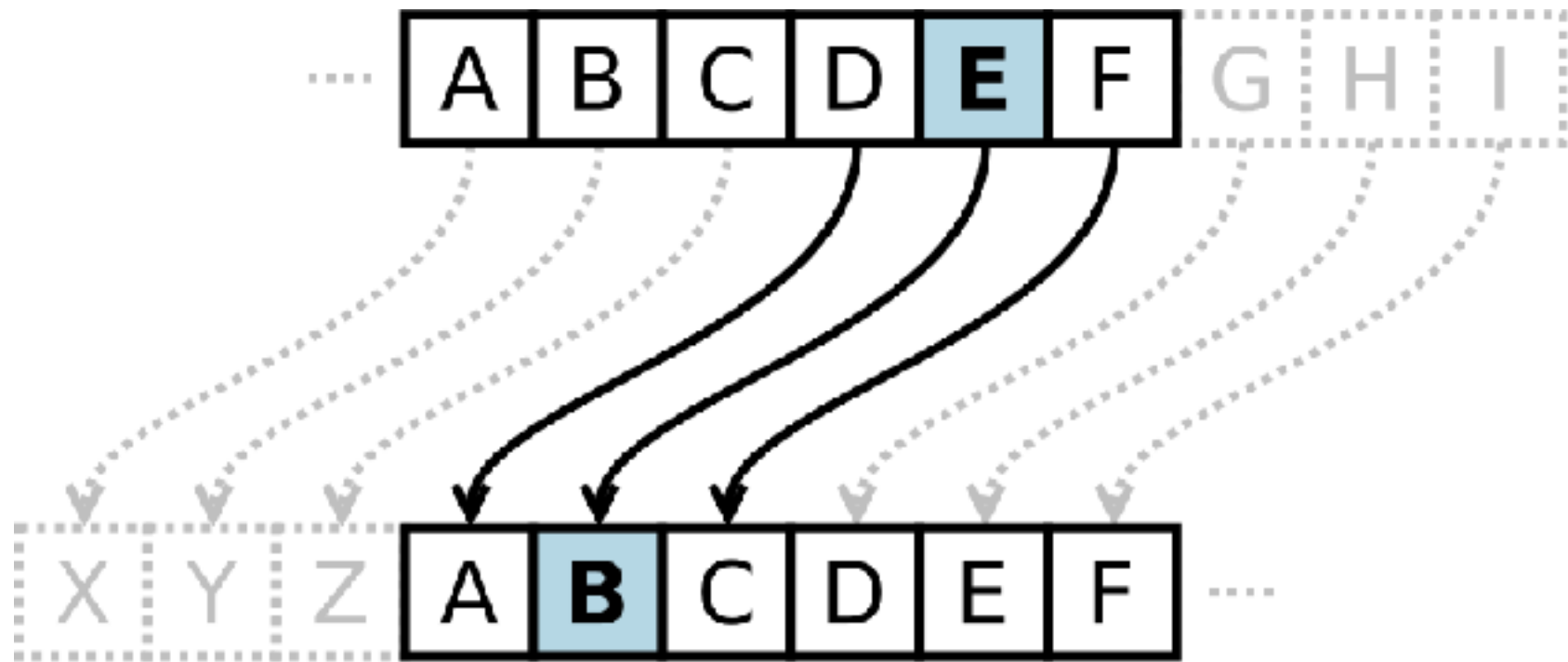
## enter a practice session

## review a practice session

**100% of your donation buys  
Raspberry Pi computers to  
help children learn to program**

**please  
donate**

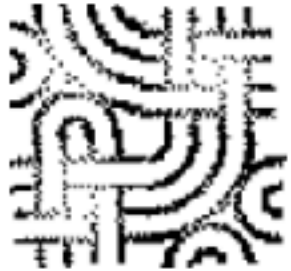
# **Group Lab: Caesar Cipher**



“Foo”  $\Rightarrow +5 \Rightarrow$  “Ktt”

# **Individual Lab/Homework!**

## **Fizz Buzz**



## Fizz Buzz Test

The "Fizz-Buzz test" is an interview question designed to help filter out the 99.5% of programming job candidates who can't seem to program their way out of a wet paper bag. The text of the programming assignment is as follows:

*"Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz"."*

Don't believe it,  
it's more like 60%

<http://wiki.c2.com/?FizzBuzzTest>



# **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).

”

# Dummies

“

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

”

# 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'.

”

# Mocks

“

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

”

# 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>

The thing that you will find is the more you adapt either functions or higher abstractions *the more likely you will not need mocks.*



# 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

# **Best Practices and Advice**

# Code Reviewer Guide

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



**Miško Hevery**  
The Testability Explorer Blog



# 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.

”

“

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](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 supported 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

# **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