

Agile Firestarter



Introduction to TDD



Alex Hung

Developer (a.k.a. Employee #2)

Ballyhoo Software

The Rationale for Unit Tests

Debatable Software Engineering 'Facts'

(Agile/XP/SCRUM/etc.) is better than (Agile/XP/SCRUM/etc.)

TDD (test-first) is better than Test-after

Non-Debatable Software Engineering Facts:

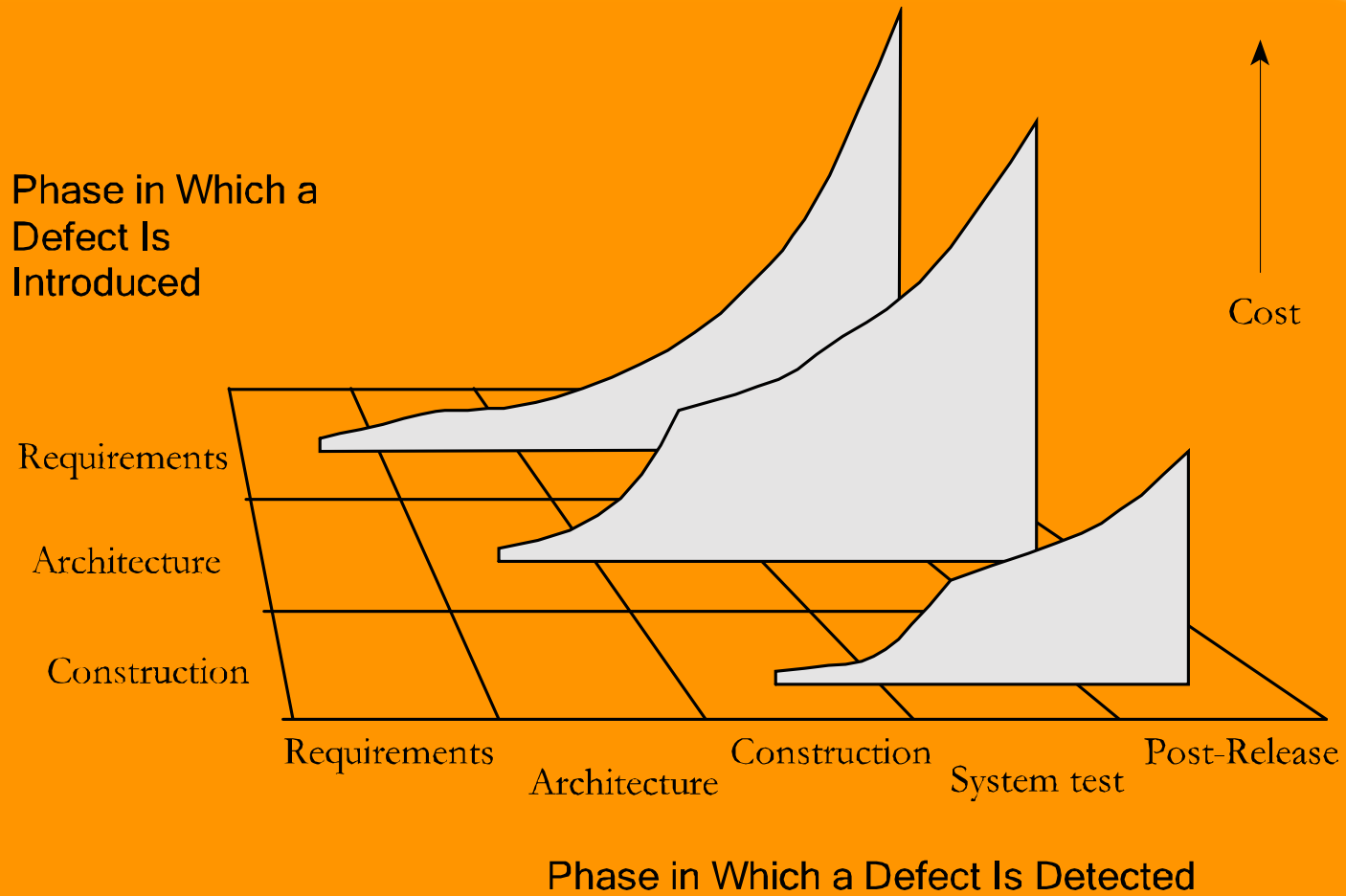
There will always be bugs

Complex programs have more bugs than simple programs

Code is more maintainable when its divided into bite-sized chunks

The cost of fixing a bug escalates non-linearly over time as the project progresses

'Code Complete' Defect Cost Graph



Types of Tests

Human-Based testing

Load the app, click the buttons

Time-consuming

Error-prone

Difficult to reliably reproduce results

Different inputs → Different outputs

Types of Tests

Automated Testing

Computer does what its good at (mass-repetition)

High-speed

Hundreds (1000s?) of automated tests in the same time to execute a single human-based test

Reproducible

Same inputs → Same outputs

Types of Automated Tests

User Interface Tests

Primary purpose is to test user interaction with the application as a whole

Typically run via a user-interface-runner (NUnitForms, NUnitASP, Selenium, Watin, Watir, White, etc.)

How is TDD different from
User Interface Testing

How is TDD different from

User Interface Testing
Exercise at Application level

Feedback loop is longer

Does not lead straight to code

Types of Automated Tests

Integration Tests

Primary purpose is to test interaction between components

Often mistaken for unit tests

Just as valuable as unit tests, every bit as automatable as unit tests

Typically run via a unit test framework but with more complex pre-test setup and post-test teardown steps

No point in running until unit tests pass!

How is TDD different from
Integration Testing

How is TDD different from

Integration Testing

Tend to be difficult to setup

Inconsistent

Does lead straight to code,
sometimes...

Types of Automated Tests

Unit Tests

Primary purpose is to test public interface of one or more classes

As Isolated as possible (hence 'unit')

Independent of other system components

If your test touches something else you didn't develop but your app depends on to run (database, AutoCAD, etc.) then its not a unit test

Sandboxed

No side-effects to the environment allowed

Typically run via a unit test framework (Nunit, MbUnit, etc.)

What is Test-Driven Development?

Testing *while* coding

not *before* or *after*

What is Test-Driven Development?

Think “Test-Driven Design”

or

“Specification-Driven-Development”

Consider your design from the perspective of the consumers of your methods, classes, etc.

Outside-in design instead of Inside-Out design

What is Test-Driven Development?

Think “Test-Driven Design”

or

“Specification-Driven-Development”

For every line of code you write, ask yourself a simple (but powerful) question:

“HOW WILL I TEST THAT?”

The Process

REFACTOR

...mercilessly!

What about...

Costs

Additional time spend on code that isn't part of delivery.

What about...

Costs

We are programmers, not testers! We should concentrate on developing features, not testing!

What about...

Benefits

Every class and method immediately has at least **TWO** consumers of it: your app and your test!

What about...

Benefits

Better-isolated code leads to easier to extend, enhance, replace, maintain (lifecycle costs)

What about...

Benefits

Waiting until code is written to write the tests often means hard (impossible?) to test code!

What about...

Benefits

Less likely to write code that you eventually don't need

Write nothing that you *think* you will need (YAGNI)

Now its your turn!

CALCULATING PRIMES

Problem Statement

Calculate Prime Numbers between 1-10,000

A positive integer divisible only by 1 and itself

By definition, 0 and 1 are *non-prime*

Write results to the console as follows:

Multiple lines of 5 comma-separated values each

Every 10 lines, insert a line indicating the count of the primes output thus far

n,n,n,n,n

n,n,n,n,n

n,n,n,n,n

n,n,n,n,n

n,n,n,n,n

n,n,n,n,n

n,n,n,n,n

n,n,n,n,n

n,n,n,n,n

n,n,n,n,n

Count: 50

n,n,n,n,n

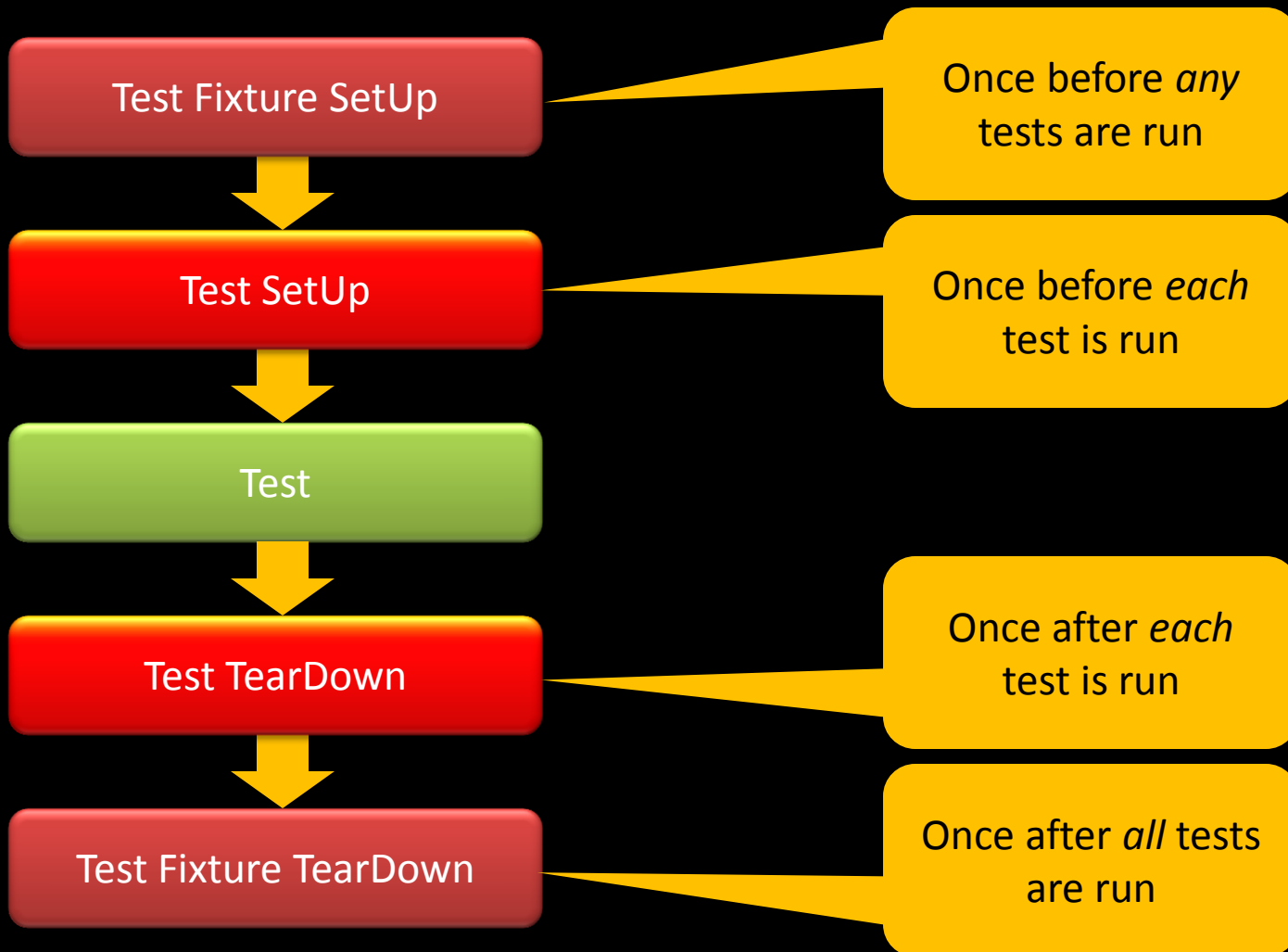
n,n,n,n,n

....

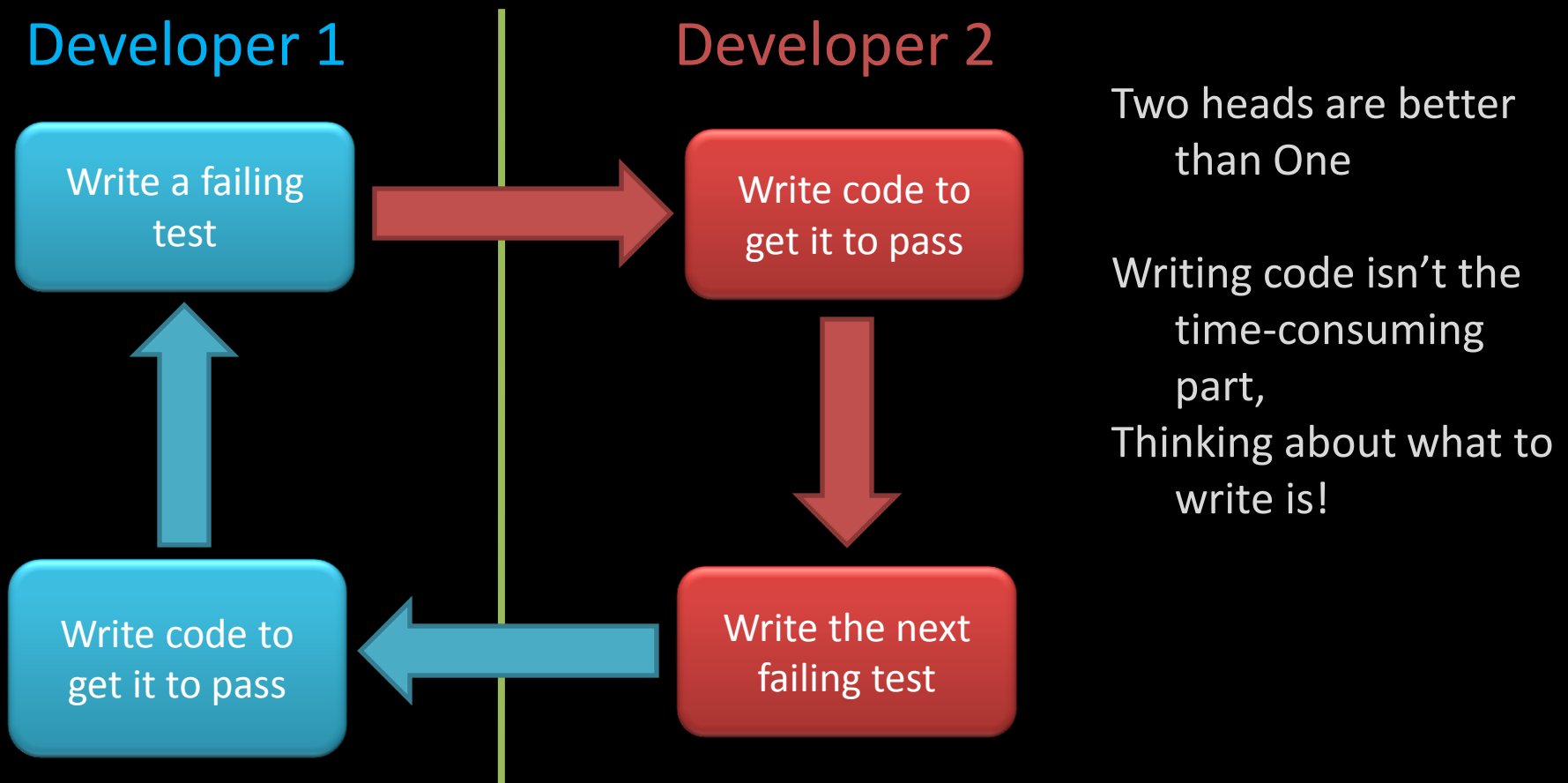
Enough talking, let's write some code!

OUR FIRST TEST

Unit Test Flow



Ping-Pong Pair Programming



Write a Failing Test

Get it to Pass

Refactor!

Now GO!

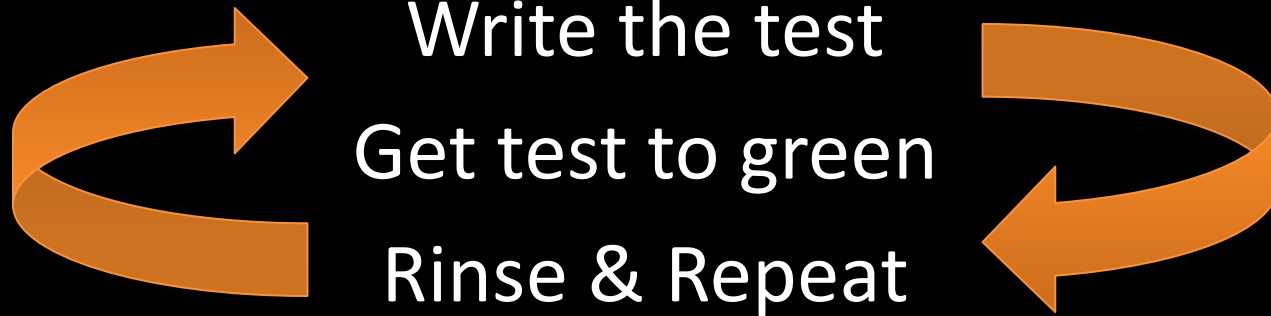
Find someone to pair with

Decide on an approach

Write the test

Get test to green

Rinse & Repeat



Remember: Failure *IS* an option!

Post-Exercise Discussion

Impression

Viewpoints

Experiences

Values

Ways to think about Unit Testing

Unit Testing is...

An alternative to spending your life in the debugger

A way to validate that your code is aligned with your intent

‘it compiles’ is a validation of the syntax of your code

‘it passes the unit tests’ is a validation of the behavior of your code

An hedge against future bugs

Prevents regression bugs (oops, I broke it!)

A safety-net that allows you to experiment with your design

‘What-if’ can be safely explored

No code is ‘hands-off for fear of something breaking’

Ways to think about Unit Testing

Unit Testing is not...

Going to save you any time in the project

...until someone changes the project requirements



Session Wrap-Up

Thoughts and Impressions



Agile
Firestarter

I bet you can smell the pizza...

LUNCH TIME!!!!

Twitter: AlexHung
alex.y.hung@gmail.com

Types of Unit Tests

State-Based Tests

Pattern is...

- Set a bunch of input values

- Do some work (call a method on the class, whatever)

- Test one or more output values

Usually validated via one or more 'Assert' statements from the unit test framework

```
Assert.AreEqual(expectedValue, actualValue);
```

Types of Unit Tests

Interaction-Based Tests

Pattern is...

- Create a bunch of mock (or stub) objects

- Do some work (pass the objects around, call some methods)

- Test one or more aspects of the interaction between the objects

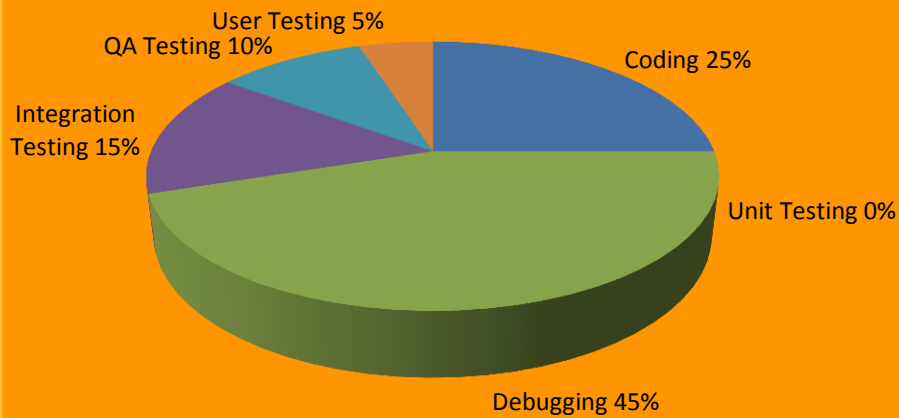
Usually validated by asking the mock framework if the expected interactions were observed between the objects

- Often no 'Assert' statements are needed; the mock framework will fail the test if the expected interactions were not observed during the test-run

Can be very effective when combined with State-based 'Assert' statements

Allocation of Developer Effort

Effort Allocation without Unit Tests



Effort Allocation with Unit Tests

