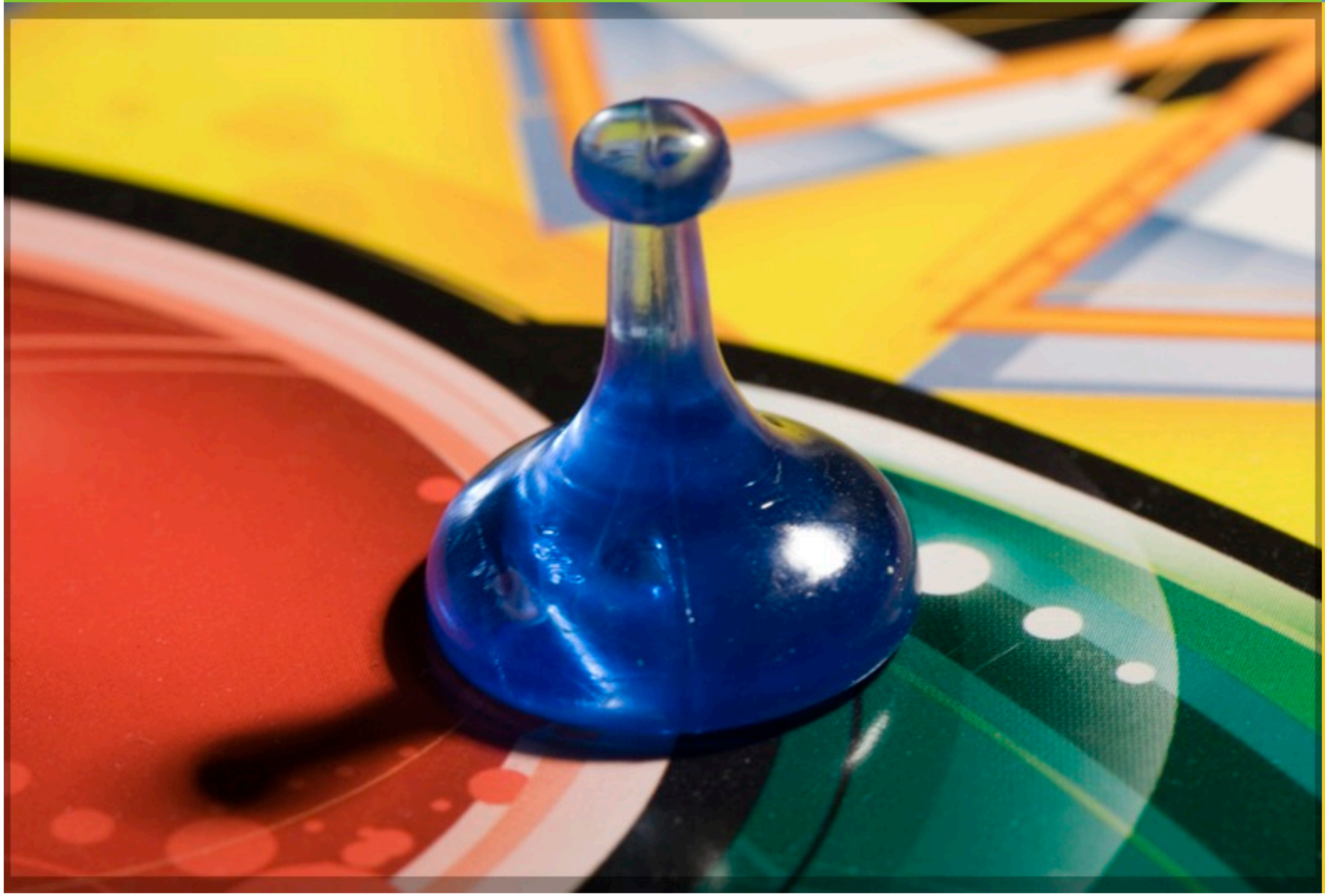
Developer Testing 101



Goals

- Learn what **is** Developer Testing
- Gain confidence in writing tests with an xUnit Testing Framework

Non-Goals

- When to test with external dependencies, ie. databases
- Not to necessarily learn how to write the best (code) tests

Resources

Code Samples

https://github.etsycorp.com/llincoln/DeveloperTesting101

PHPUnit Manual

http://www.phpunit.de/manual/current/en/

PHPUnit GitHub

https://github.com/sebastianbergmann/phpunit

Etsy PHPUnit Extensions

https://github.com/etsy/phpunit-extensions/wiki

Topics

- Methodology, Terminology, Kool-Aid
- xUnit Basics and Theory
- Assertions
- Data Driven Tests
- Test Classification
- Stubs, Fakes, and Mocks

- Methodology, Terminology, Kool-Aid
  - Agile
  - Test-Driven Development (TDD)
  - Pyramid of Testing

Agile

a·gil·i·ty [*uh*-**jil**-i-tee]
noun
1. the power of moving quickly and easily; nimbleness: *exercises demanding agility*.
2. the ability to think and draw conclusions quickly; intellectual acuity.

Agile Manifesto

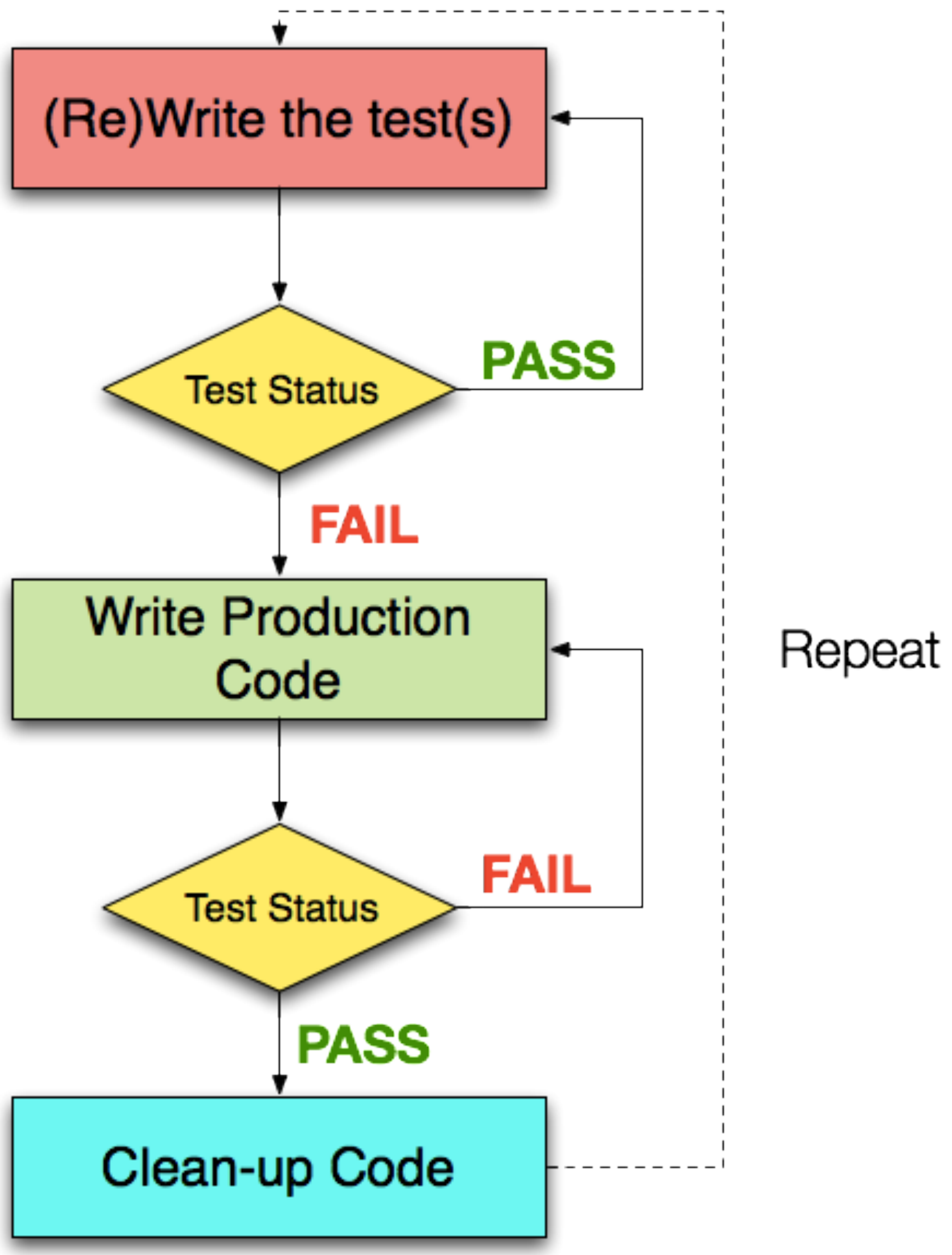
Individuals and interactions
over processes and tools

Working software
over comprehensive documentation

Customer collaboration
over contract negotiation

Responding to change
over following a plan

TDD

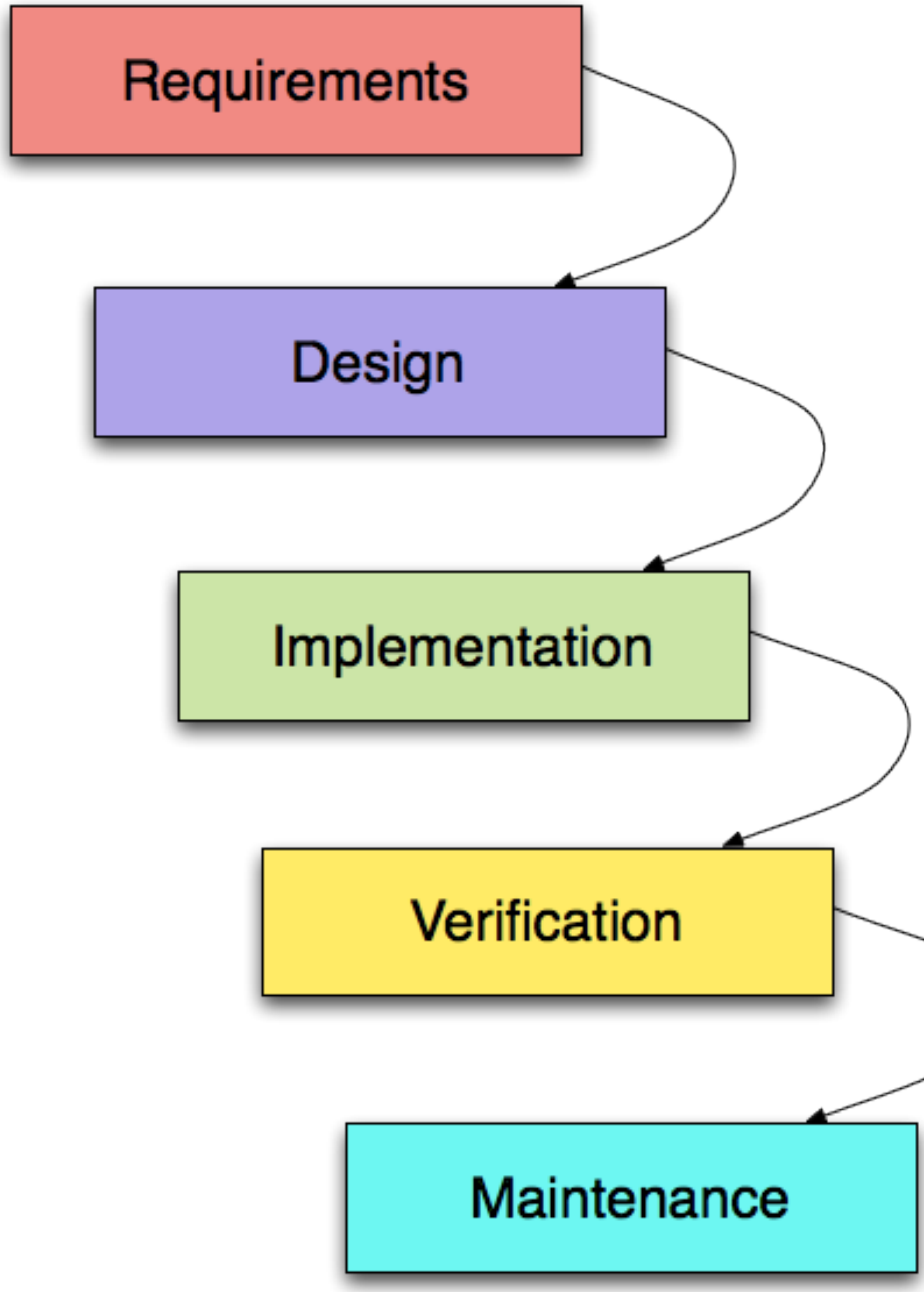


TDD

Test Driven Development

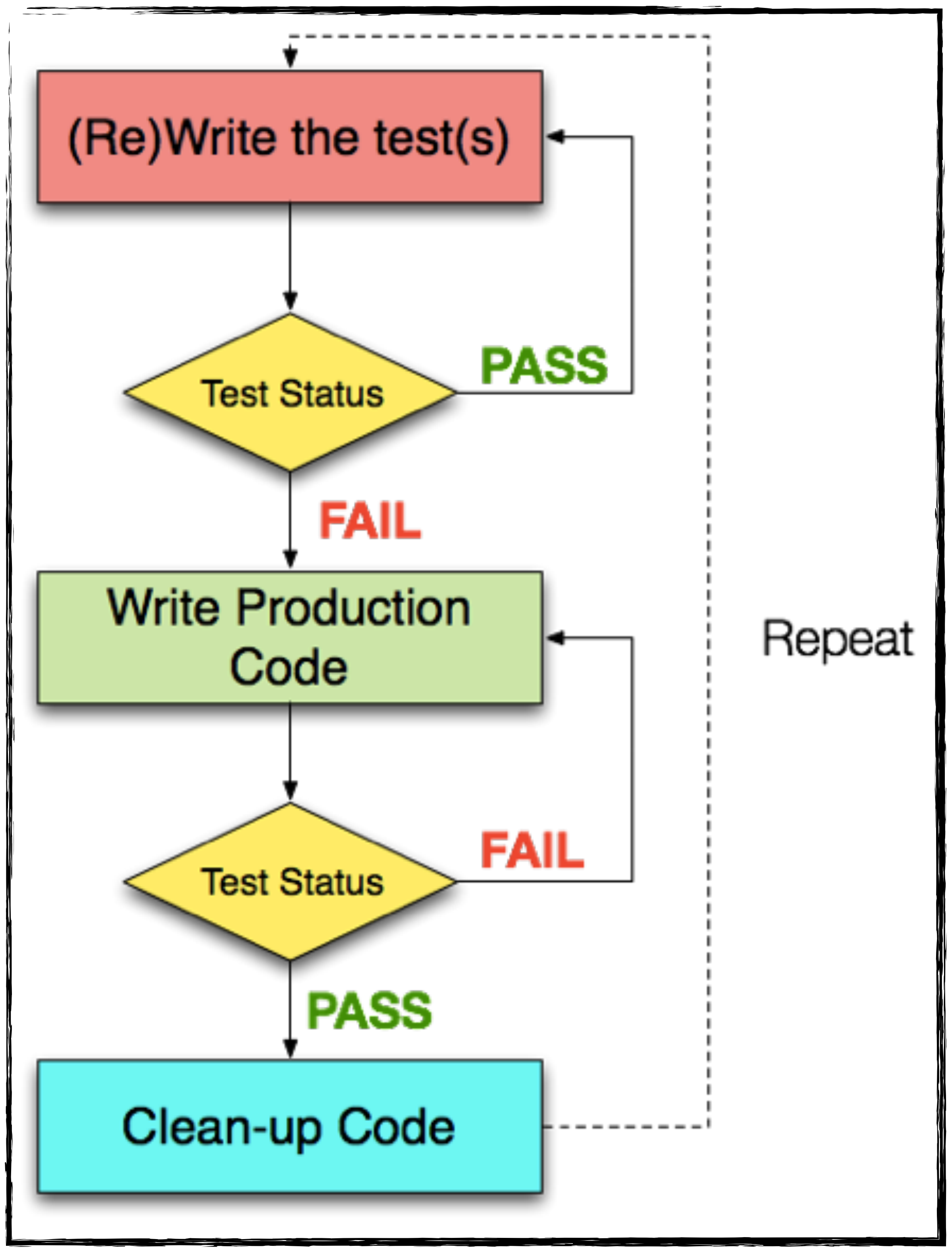
Waterfall

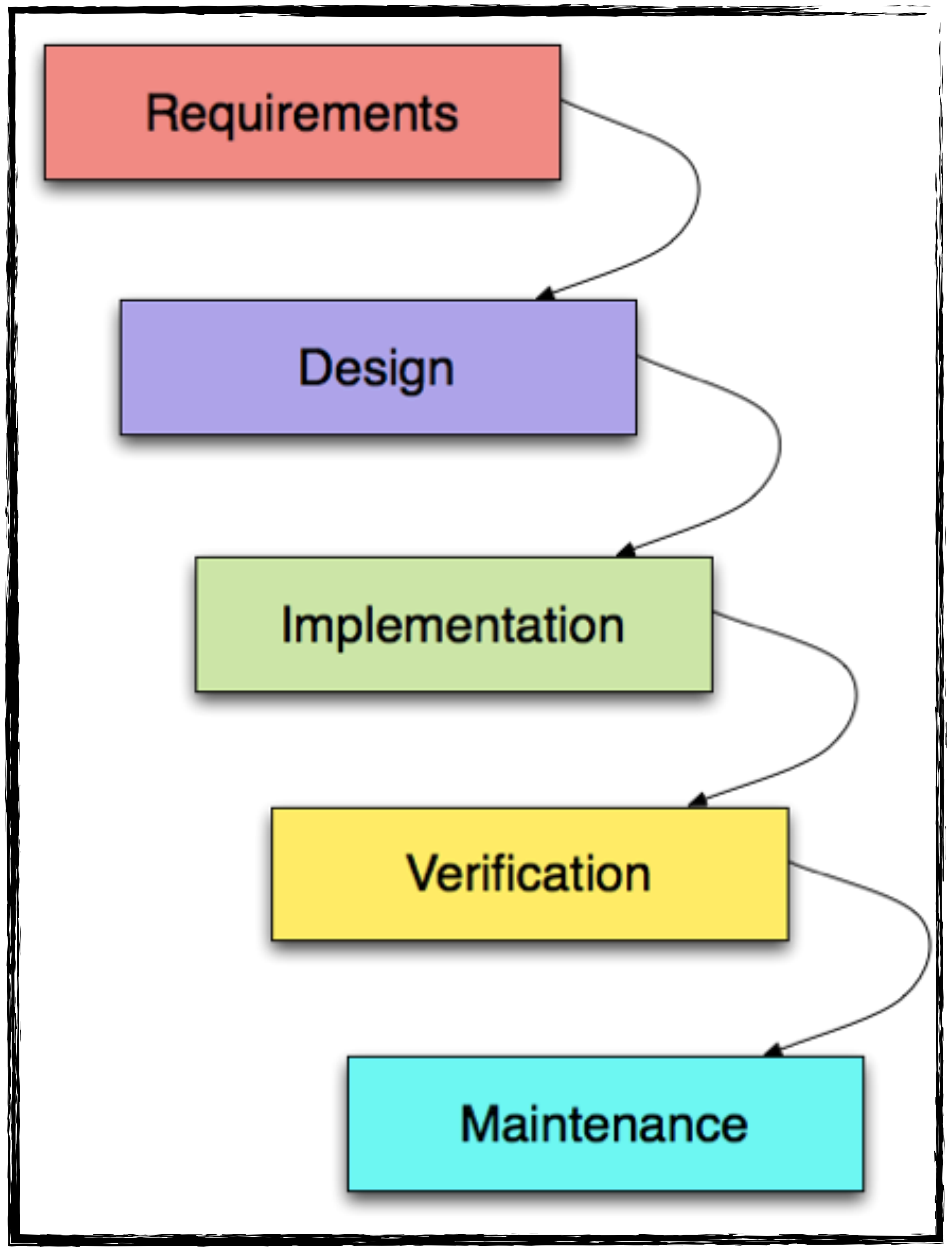
The Assembly Line



TDD

Waterfall





Developer Testing

TDD is a design process

Why Developer Testing?

- Verify Correctness
- Communication
- Gain Confidence

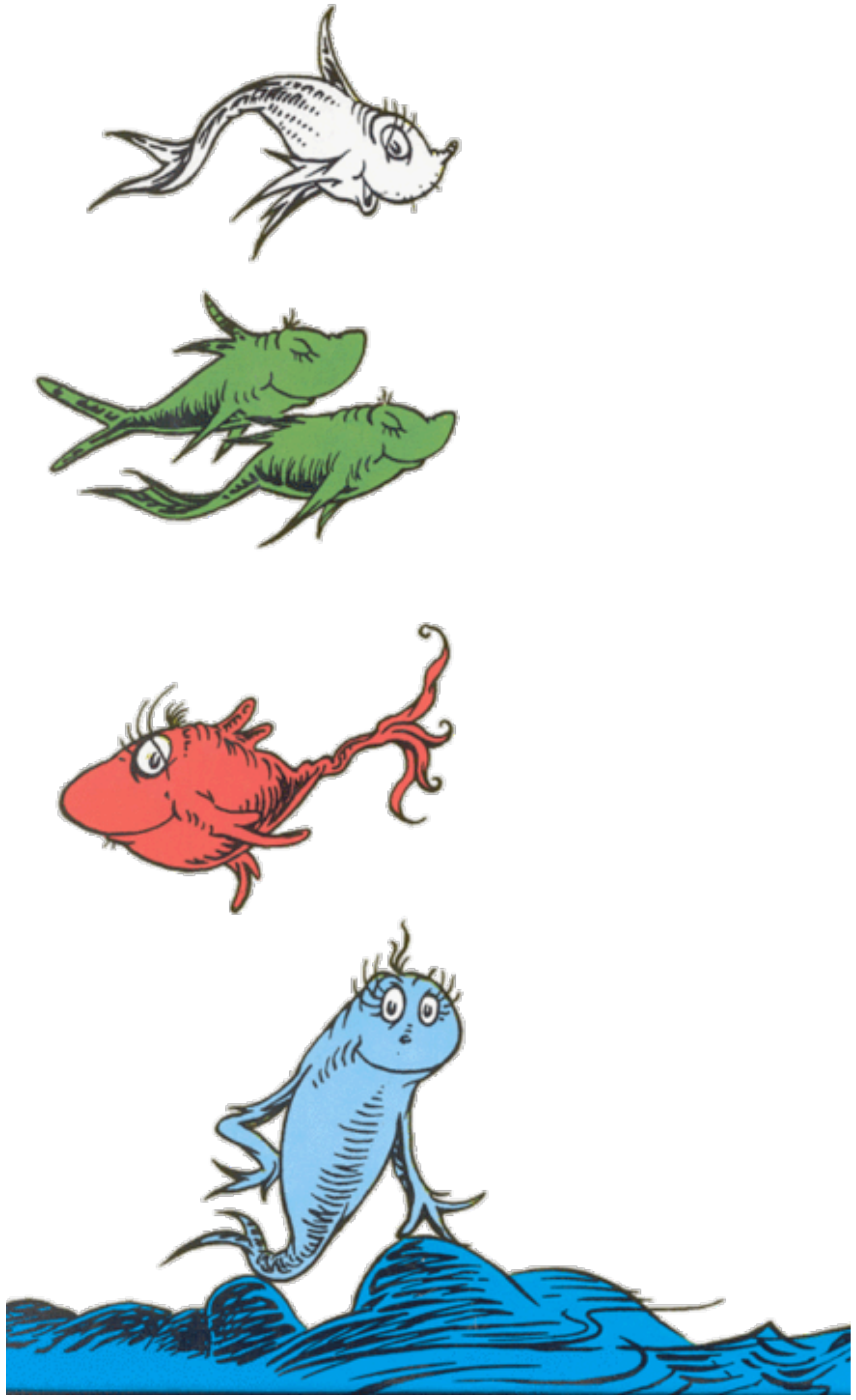
Inside Out v. Outside In

Inside Out v. Outside In

• Implement **first**	• Test **first**
• *Easy* to code	• *Easy* to test
• *Difficult* to test	• *Easy* to code

Tests should always be treated like every other consumer of the subject under test.

Pyramid of Testing



Test Types

Vocabulary Break!

Functional Tests

Does the overall product satisfy the the requirements?

Integration Tests

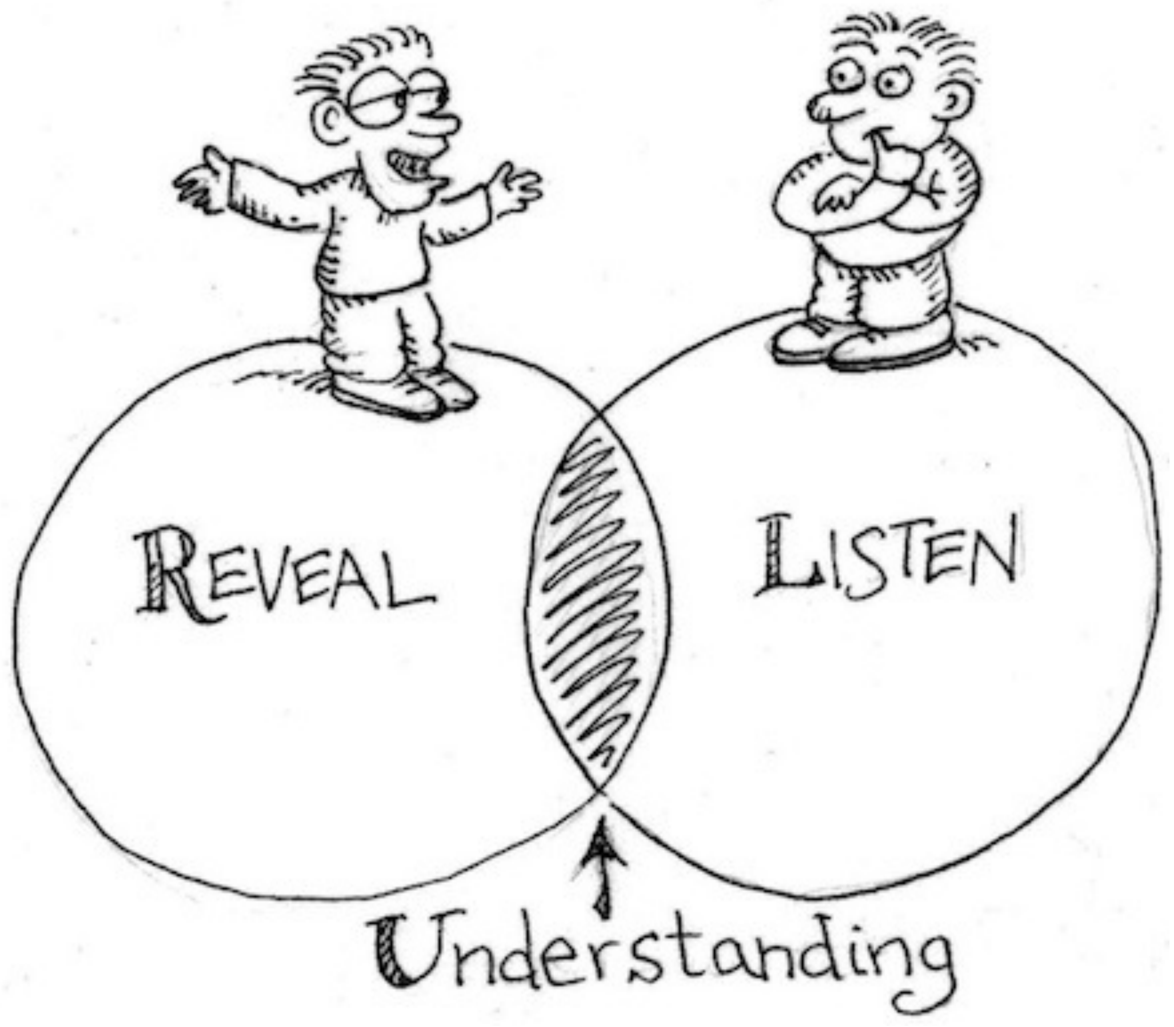
Do the pieces fit together?

Unit Tests

Is the logic correct in that function?

Functional

Understand the product.



The Tools

The Tools

- BeHat (Cucumber)

The Tools

- BeHat (Cucumber)
- PHPSpec

The Tools

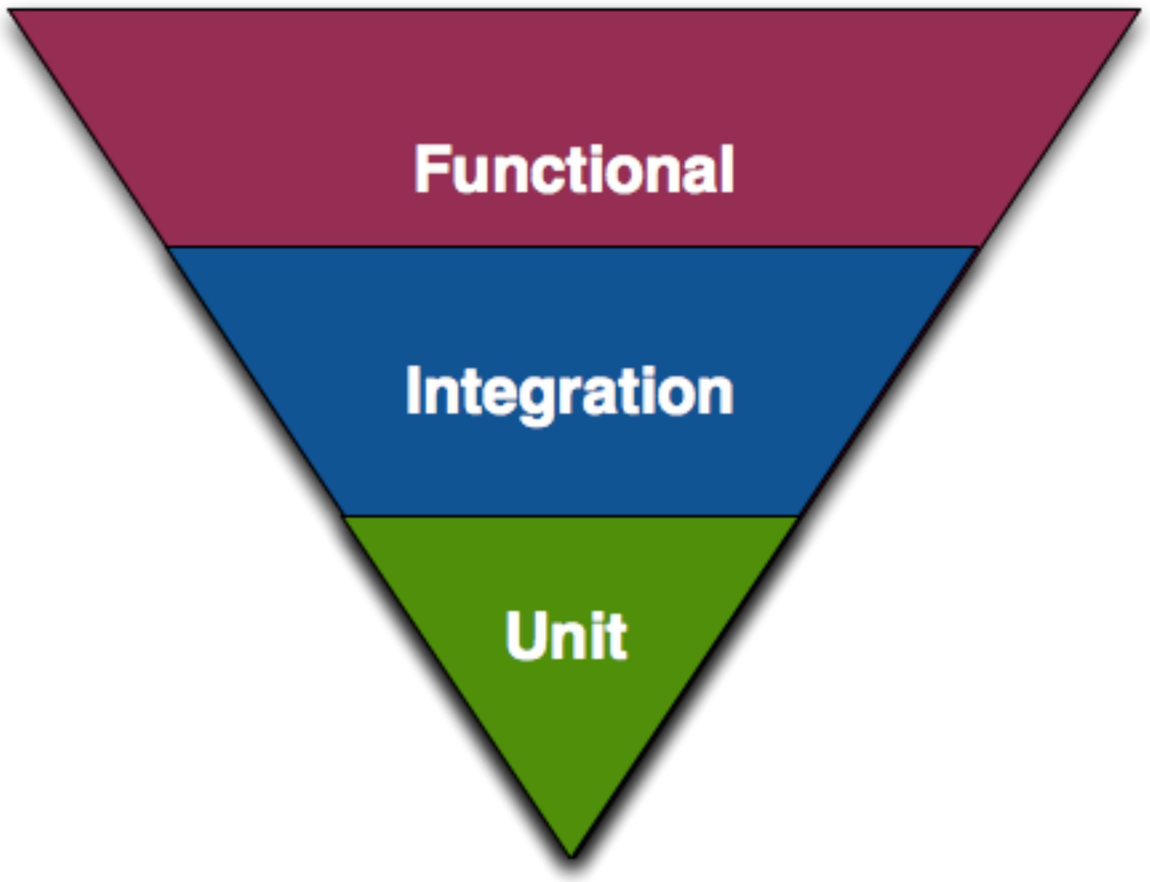
- BeHat (Cucumber)
- PHPSpec
- Keyboard, Mouse, and You

When Do They Work?

- During prototyping
- Focused on the product requirements
- Refactoring
- Regression of key features
- Better for smaller teams

...Stop Helping?

- Focused on the implementation
- Rapidly changing functionality
- Large organizations

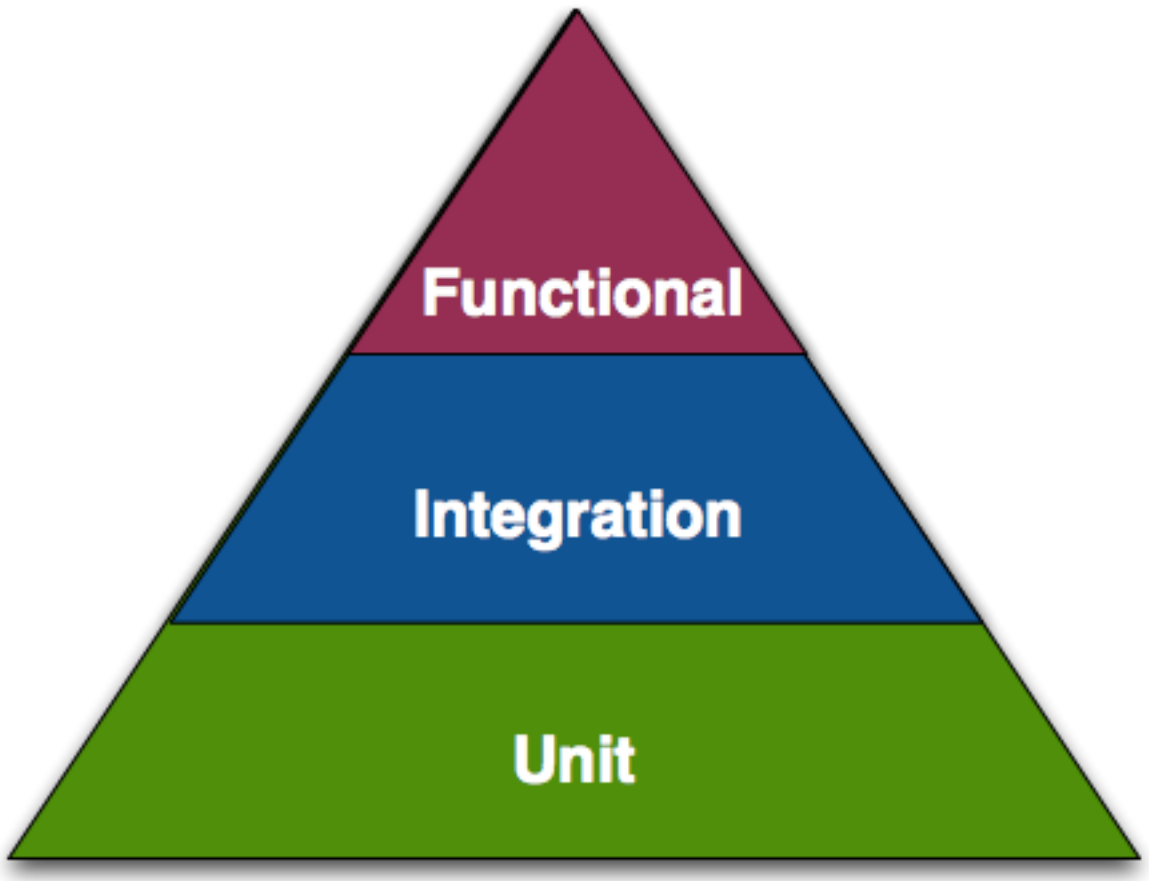


After Greenfield

Inverted Test Pyramid

Test Pyramid

Food Pyramid of the 90s



Unit Tests

Simple and to the point.

Verify Correctness

- Line coverage
- Branch coverage
- Icky Bits-o-Logic

Gain Confidence

- Individual functions
- Variety of parameters
- Works for expected interactions with collaborators

Communication

- Show how to use the function
- Show expected interactions with other collaborators
- Increase discoverability of possible reuse

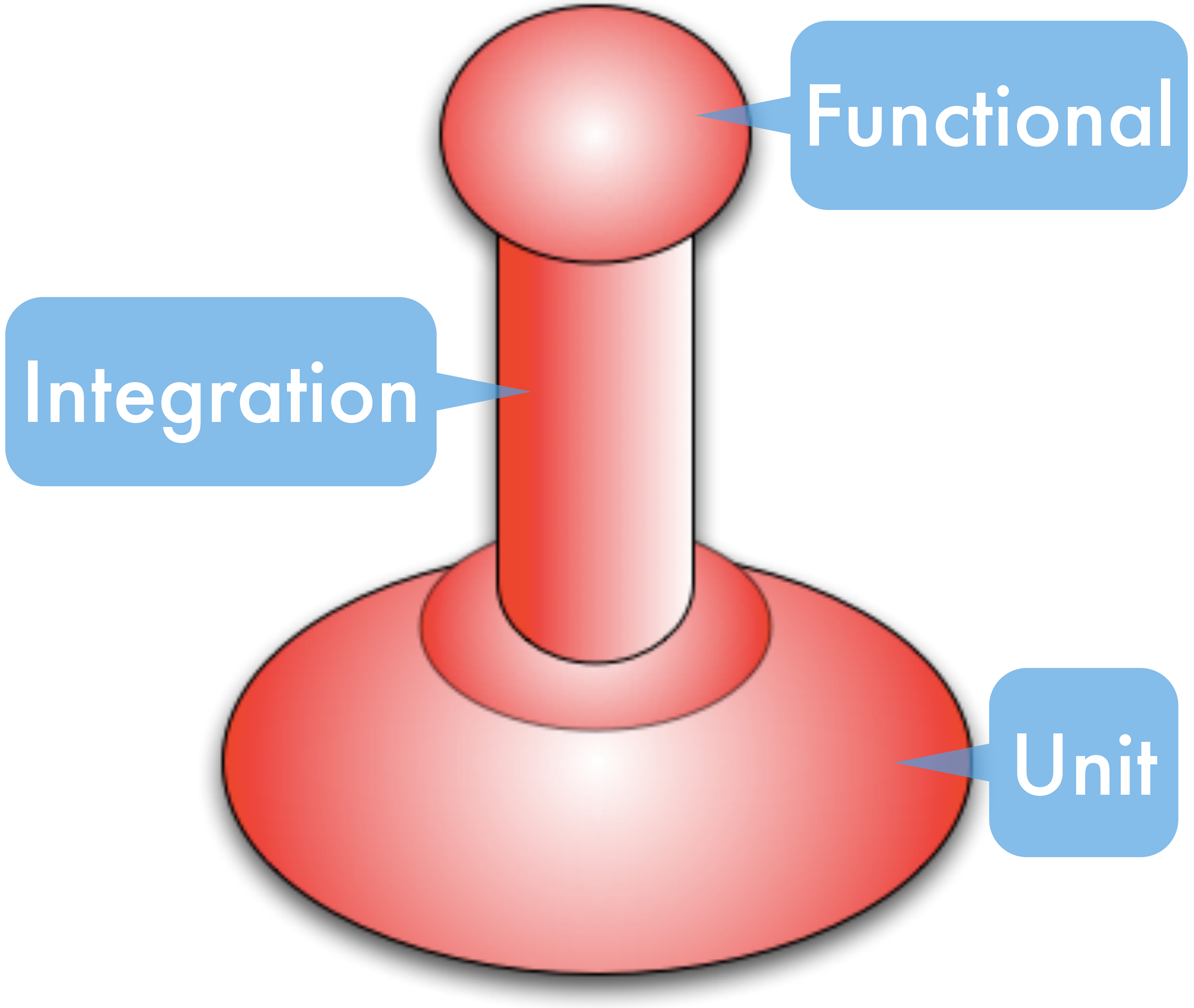
Integration

For everything in-between.

You're Paranoid

- Experimenting with third-party code or service
- You do not trust that your collaborators work as specified

Sorry! Piece of Testing



- Methodology, Terminology, Kool-Aid
  - Agile for realz!
  - ~~Test Driven Development (TDD)~~ Developer Testing
  - ~~Pyramid~~ Sorry! piece of Testing

xUnit Basics and Theory

TestCase

```
class My_Class { ... }


class My_ClassTest
extends PHPUnit_Framework_TestCase
{ ... }
```

testFunction

```
class My_Class {
    function foo(){ ... }
}


class My_ClassTest
extends PHPUnit_Framework_TestCase {
    function testFoo() { ... }
}
```

@test

```
class My_ClassTest
extends PHPUnit_Framework_TestCase {
    /**
     * @test
     */
    function foo() { ... }
}
```

Several Tests Per Function

```
class My_Class {
    function foo(/*bool*/ $param){ ... }
}
class My_ClassTest
extends PHPUnit_Framework_TestCase {
    function testFoo_true() { ... }
    function testFoo_false() { ... }
}
```

xUnit Basics

- `HappySet`
- `01-xUnitBasics`
  - `FlowTest.php`
  - `ChildTest.php -> ParentTest.php`
  - `UniqueInstanceTest.php`
  - `StaticTest.php`
  - `GlobalStateTest.php`
  - `TestListener.php` and `listener-`

Honorable Mentions

- `public static function setUpBeforeClass()`
- `public static function tearDownAfterClass()`
- `/** @depends */`

Equivalence Class Partitioning

A Little Math Lesson

Equivalence Relation

- Let S be any set and let $\sim$ be a relation on S. Then $\sim$ is called an **equivalence relation** provided it satisfies the following laws:
  - **Reflexive** $a \sim a$
  - **Symmetrical** $a \sim b$ implies $b \sim a$
  - **Transitive** $a \sim b$ and $b \sim c$ implies $a \sim c$

Partition

- A **partition** of a nonempty set S is a collection of nonempty subsets that are disjoint and whose union is S.

Equivalence Class

Consider again an equivalence relation ~ on a set S. For each s in S we define

$$[s] = \{t \in S : s \sim t\}$$

The set $[s]$ is called the **equivalence class** containing s.

Input Partitioning

Write just the
right number of tests

ECP

- 02-ECP
  - Boolean.php
  - String.php
  - Integer.php
  - ControlStructures.php
  - Object.php

Assertions

Assertions

- `03-Assertions`
  - `StatusTest.php`
  - `AssertTest.php`
  - `ExceptionTest.php`

Built-In Assertions

- `assertArrayHasKey`
- `assertContains`
- `assertContainsOnly`
- `assertCount`
- `assertEmpty`
- `assertEquals`
- `assertFalse`
- `assertGreaterThan`
- `assertGreaterThanOrEqual`
- `assertInstanceOf`
- `assertInternalType`
- `assertLessThan`
- `assertLessThanOrEqual`
- `assertNull`
- `assertRegExp`
- `assertStringMatchesFormat`
- `assertSame`
- `assertStringStartsWith`
- `assertStringEndsWith`
- `assertTag`
- `assertThat`
- `assertTrue`
- `more...`

Custom Asserts

- Work-around for multiple asserts
- No appropriate assert or constraint exists
- One of the few times `statics` aren't so bad
- Consider writing a new `PHPUnit_Framework_Constraint`
- Use

```
assertThat(
    mixed $value,
    PHPUnit_Framework_Constraint $constraint
    [, $message = '']
)
```

Data Driven Tests

Data Driven Testing

- `04-DataDrivenTesting`
  - `Calculator.php`
  - `CalculatorTest.php`
  - `DataDrivenTest.php`

@dataProvider

```
/**
 * @dataProvider <methodName>
 */
```

- Provider method must be `public`
- Return must be a double `array`
- Cannot depend on instance variables
- Always use literal values

Test Classification

External Dependencies

- Memcache
- MySQL
- Postgres
- Services via Network

More Sources of Flake

- Sleep
- Date / Time
- Random Number Generators

`@group`

- `@group cache`
  for test that use memcache
- `@group dbunit`
  for test that use DBUnit and databases
- `@group network`
  for tests that talk to external services
- `@group flaky`
  for tests that fail without a code change

Unit Tests are DEFAULT

There is **NO** `@group` for unit tests.

Test Sizes

- New in PHPUnit 3.6
- `@small` run in less than one second
- `@medium` run in less than 10 seconds
- `@large` run in less than 60 seconds
- Note: All times are configurable

Stubs, Fakes, and Mocks

The Difference

- **Stub** *returns fixed values*
- **Fake** *returns modifiable values*
- **Mock** *mimics the behavior of the original*

Creating a Mock

```
   /**
  * Returns a mock object for the specified class.
  *
  * @param  string  $originalClassName
  * @param  array   $methods
  * @param  array   $arguments
  * @param  string  $mockClassName
  * @param  boolean $callOriginalConstructor
  * @param  boolean $callOriginalClone
  * @param  boolean $callAutoload
  * @return PHPUnit_Framework_MockObject_MockObject
  * @throws InvalidArgumentException
  * @since  Method available since Release 3.0.0
  */
 public function getMock($originalClassName, $methods = array(), array
$arguments = array(), $mockClassName = '', $callOriginalConstructor = TRUE,
$callOriginalClone = TRUE, $callAutoload = TRUE) { ... }
```

Mock Builder

```
$stub = $this->getMock(
    'SomeClass', null, null, '', false
);
```

or

```
$stub =
    $this->getMockBuilder('SomeClass')
        ->disableOriginalConstructor()
        ->getMock();
```

Mock Builder Options

- `setMethods(array|null $methods)`
- `setConstructorArgs(array|null $args)`
- `setMockClassName($name)`
- `disableOriginalConstructor()`
- `disableOriginalClone()`
- `disableAutoload()`

expects()

- Takes a `PHPUnit_Framework_MockObject_Matcher`
  - `$this->any()`
  - `$this->never()`
  - `$this->once()`
  - `$this->exactly(int $count)`
  - `$this->at(int $index)`

`method()`

- Simply takes the name of the method to stub/mock as a `String`

`with()`

- Takes a variable number of `Strings` or `PHPUnit_Framework_Constraints`

with()

- equalTo()
- **anything()**
- isTrue()
- isFalse()
- isNull()
- contains()
- containsOnly()
- arrayHasKey()
- isEmpty()
- greaterThan()
- greaterThanOrEqual()
- lessThan()
- lessThanOrEqual()
- identicalTo()
- isInstanceOf()
- isType()
- matchesRegularExpression
- matches()
- stringStartWith()
- stringEndWith()
- stringContains()
- **logicalAnd()**
- **logicalOr()**
- **logicalNot()**
- **logicalXor()**
- more...

`will()`

- Takes a `PHPUnit_Framework_MockObject_Stub`
  - `$this->returnValue()`
  - `$this->returnArgument()`
  - `$this->returnSelf()`
  - `$this->returnValueMap()`
  - `$this->returnCallback()`
  - `$this->onConsecutiveCalls()`
  - `$this->throwsException()`

Return Stub

- See an example of a `PHPUnit_Framework_MockObject_Stub`
  - `etsy/phpunit-extensions`

Stub (or Fake) in PHPUnit

```
$stub = $this->getMock('SomeClass');

$stub
    ->expects($this->any())
    ->method('someMethod')
    ->will($this->returnValue(2));
```

Mocks in PHPUnit

```
$mock = $this->getMock('SomeClass');

$mock
    ->expects($this->any())
    ->method('someMethod')
    ->with($this->greaterThan(2))
    ->will($this->returnValue(true));
```

Preview: Mockery

- Requires PHP 5.3
- More fluent PHPUnit Mocks
    - expects->method->with->will
- Example

```
use \Mockery as m;

 class SimpleTest extends PHPUnit_Framework_TestCase {
    public function testSimpleMock() {
        $mock = m::mock('simple mock');
        $mock->shouldReceive('foo')->with(5, m::any())->once()->andReturn(10);
        $this->assertEquals(10, $mock->foo(5));
    }

    public function teardown() {
        m::close();
    }
}
```