Unit Testing With PHP

A far too quick survey of a few tools

What's a unit test?

Simply, code that tests code

A unit is the smallest unit of your program. Probably a function, method, or procedure.

Why unit tests?

Objectives:

Prevent regressions
Document expected behavior (prove that it works this way)
Find problems early. So many studies have shown the exponential cost of defect repair over time. (though some studies suggest TDD may make that cost curve logarithmic)
Takes the fear out of refactoring
Test Driven Design, you'll find that design and interfaces improve

Test Driven Development

Sometimes used interchangeably with Test First Development
To put it simply, it's about making unit tests an integral part of your process, not as a second phase or afterthought.

Write your test first. Verify it fails (red). Implement the method. Watch it go green.

Behavior Driven Development

An evolution of TDD. Tests focusing on user behavior, not as much on method behavior. Subtle differences all around. This mostly changes the language you use to describe the test and perhaps what to test more than the mechanics of how to test.

We'll be focusing on how to unit test more than the principles of TDD vs BDD here. Lots of reading material linked at the end

Why test first?

You will code faster (after you get used to it).

- Integrate into your IDE, you'll get red / green notifications as you type

Your code will be easier to change later

You start with the documents (the tests) of what you intended and the proof that it works

The earlier your find defects the less costly

PHP Unit Testing tools

PHPUnit, we'll use this in our examples. It's yet another implementation of the xUnit pattern. It started with Kent Beck and smalltalk. Popularized by JUnit.

There are other tools out there for PHP too like Simpletest and phpspec.
I'm intrigued by phpspec which follows the BDD specification style declarations. Similar to rspec, jasmine and others. I would probably look at using this if I were to start over today.

BDD tools. behat for php which is similar to Cucumber and JBehave

Lexicon

- Test Case
- Test Suite
- SUT System Under Test

Test Case. Your basic grouping
Test Suite. Collections of test cases.

System Under Test. What you are trying to test (you will accidentally test other things)

```php
<?php

namespace CentralDesktop\OCPHP;
use CentralDesktop\OCPHP;

/**
 * Class Math
 * @package CentralDesktop\OCPHP
 */
class Math {
    /**
     * @return integer
     */
    function add($a, $b) {
        return $a + $b;
    }

```

```php
<?php

namespace CentralDesktop\OCPHP\Test;

use CentralDesktop\OCPHP;

class MathTest extends \PHPUnit_Framework_TestCase {

    public
    function testAddSimple() {
        $math = new OCPHP\Math();

        $this->assertEquals(1, $math->add(0, 1));
        $this->assertEquals(2, $math->add(1, 1));
        //$this->assertEquals(2, $math->add(1, 2));
    }

```

Let's show some code

```
PHPUnit 3.7.27 by Sebastian Bergmann.

Configuration read from /Users/thyde/ocphp-examples/phpunit.xml

.............

Time: 49 ms, Memory: 7.75Mb

OK (13 tests, 18 assertions)
```

```
PHPUnit 3.7.27 by Sebastian Bergmann.

Configuration read from /Users/thyde/ocphp-examples/phpunit.xml

F............

Time: 42 ms, Memory: 8.00Mb

There was 1 failure:

1) CentralDesktop\OCPHP\Test\MathTest::testAddSimple
Failed asserting that 3 matches expected 2.

/Users/thyde/ocphp-examples/test/src/CentralDesktop/OCPHP/Test/MathTest.php:15

FAILURES!
Tests: 13, Assertions: 19, Failures: 1.
```


Warning: Some very basic and tired examples ahead. You'll find very similar things in lots of other examples.

Pro tip: Code and test side by side.

Unit testing's ideal, your methods are true functions. For a given input, the output will always be exactly the same.

xunit convention. Test methods start with "test". Test classes end with "Test"
Extending TestCase provides assertion methods. Usually tons of them

```php
    /**
     * @dataProvider addProvider
     */
    public
    function testAddDD($result, $a, $b) {
        $math = new OCPHP\Math();

        $this->assertEquals($result, $math->add($a, $b));
    }

    /**
     * Provides data for the test above
     *
     * @return array
     *
     */
    public
    function addProvider() {
        return array(
            array(1, 0, 1),
            array(2, 1, 1),
            array(3, 2, 1),
            //array(3, 2, 2),
        );
    }
```

```php
    /**
     * @param $num
     * @param $denom
     *
     * @return float
     * @throws DivideByZeroException
     */

    function divide($num, $denom) {
        if ($denom == 0) {
            throw new OCPHP\DivideByZeroException("You shouldn't do that.");
        }

        return $num / $denom;
    }
}
```

```
PHPUnit 3.7.27 by Sebastian Bergmann.

Configuration read from /Users/thyde/ocphp-examples/phpunit.xml

....F.........

Time: 40 ms, Memory: 8.00Mb

There was 1 failure:

1) CentralDesktop\OCPHP\Test\MathTest::testAddDD with data set #3 (3, 2, 2)
Failed asserting that 4 matches expected 3.

/Users/thyde/ocphp-examples/test/src/CentralDesktop/OCPHP/Test/MathTest.php:25

FAILURES!
Tests: 14, Assertions: 19, Failures: 1.
```

Data Driven Tests

Shown output is with commented condition uncommented

```php
    /**
     * @expectedException \CentralDesktop\OCPHP\DivideByZeroException
     *
     * I could have also called
     * $this->setExpectedException('\CentralDesktop\OCPHP\DivideByZeroException');
     *
     */
    public
    function testDivideByZero() {
        $math = new OCPHP\Math();

        $math->divide(10, 0);
    }
```

```php
    /**
     * @param $num
     * @param $denom
     *
     * @return float
     * @throws DivideByZeroException
     */

    function divide($num, $denom) {
        if ($denom == 0) {
            throw new OCPHP\DivideByZeroException("You shouldn't do that.");
        }

        return $num / $denom;
    }
}
```

```
PHPUnit 3.7.27 by Sebastian Bergmann.

Configuration read from /Users/thyde/ocphp-examples/phpunit.xml

.....F........

Time: 37 ms, Memory: 7.75Mb

There was 1 failure:

1) CentralDesktop\OCPHP\Test\MathTest::testDivideByZero
Failed asserting that exception of type "\CentralDesktop\OCPHP\DivideByZeroException" is thrown.


FAILURES!
Tests: 13, Assertions: 18, Failures: 1.
```

Test Exceptions

If I change
$math->divide(10,0) to
$math->divide(10,1)
See failure on this slide

Test Doubles

- Fakes
- Stubs
- Mocks

Fakes: A working implementation specifically for testing. In memory data instead a call to the database, a MTA that writes to a log instead of actually sending email, etc

Stubs: Provides canned answers to methods. Might remember test state (was method foo() called?)

Mocks: Builds the testing assertions into object. Example this method must be called once in order for the test to succeed.

Most people and frameworks combine mocks and stubs. Examples up next

```php
    /**
     * @param User $user
     *
     * @return Mailer|SMS
     * @throws UnknownContactPreferenceException
     */
    public
    function getNotifier(User $user) {
        $pref = $user->get_notification_preference();

        $notifier = null;

        switch ($pref) {
            case User::PREF_EMAIL:
                // harder to test
                // $notifier = new Mailer();
                $notifier = $this->create_mailer();

                break;
            case User::PREF_SMS:
                // harder to test
                // $notifier = new SMS();
                $notifier = $this->create_sms();
                break;

            default:
                throw new OCPHP\UnknownContactPreferenceException();

        }

        return $notifier;
    }
```

```php
interface Notifier {

    /**
     * Sends the user a message
     * @param User $user
     * @param      $message
     *
     * @return boolean
     */
    public
    function send(User $user, $message);
}
```

Slightly more complicated...

What should we be testing here?

create_mailer, create_sms return different classes of Notifier.

Given a notification preference, do we return the right object class?

To test this, we're dependent on the behavior of the User object

```php
    public
    function testGetNotifierTDD() {
        $factory = OCPHP\NotificationFactory::getInstance();

        $user     = new OCPHP\User("Joe", "joe@test.com", "+16194922222");
        $notifier = $factory->getNotifier($user);

        $this->assertInstanceOf('\CentralDesktop\OCPHP\SMS', $notifier,
                                "This was not the instance you were looking for");


    }
```

Martin Fowler, classicist vs mockist

```php
    /**
     * Mockist example, make sure we are testing behaviors ONLY of getNotifier and not the User object
     * If the User object default behavior changed, this test doesn't fail (User tests should!)
     *
     * This also shows up the mock proxies, when instantiating an object is more complicated
     */
    public
    function testGetNotifierMockist() {
        $factory = OCPHP\NotificationFactory::getInstance();

        $user = \Mockery::mock('\CentralDesktop\OCPHP\User', array('mocked', 'mock@mock.com', '+16194922222'));
        $user->shouldReceive('get_notification_preference')->andReturn(OCPHP\User::PREF_EMAIL)->once();

        $notifier = $factory->getNotifier($user);

        $this->assertInstanceOf('\CentralDesktop\OCPHP\Mailer', $notifier,
                                "This was not the instance you were looking for");


    }
```


For more complex systems. Do you use real objects or mock objects? The debate will rage on but it might boil down to how complex your objects are.

Mockist example using mockery. PHPUnit has a mocking framework built in but we are using mockery because it's interface is easier to read and as a demonstration you have some choice.

Test Driven Design

In the previous example, what if my Mailer object had a built in MTA class with a has-a relationship? How would I prevent an email from being sent?

This is where your design changes because you are testing. Avoid hardcoding an MTA entirely, use an interface. Perhaps even load up the object using a Dependency Injection framework like Symphony's DI component. You can override an internal ->get_mta() method with a mock, but wouldn't it be easier to change the implementation under the hood with a ->set_mta() method?

Writing testable code

It's important to talk about. It can be difficult to retrofit tests on to old code.

So many best practices out there, read the resources section here for a start.

Testing anti-patterns:
Using global state
Methods too long
Large number of conditions
Too many parameters
Too many public methods (your testing goals are largely around your public interfaces, these are the most impactful to test)

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		93.94%	31 / 33		85.71%	12 / 14		71.43%	5 / 7
DivideByZeroException.php								100.00%	1 / 1
Mailer.php		0.00%	0 / 1		0.00%	0 / 1		0.00%	0 / 1
Math.php		100.00%	4 / 4		100.00%	2 / 2		100.00%	1 / 1
NotificationFactory.php		100.00%	19 / 19		100.00%	5 / 5		100.00%	1 / 1
Notifier.php									
SMS.php		0.00%	0 / 1		0.00%	0 / 1		0.00%	0 / 1
UnknownContactPreferenceException.php								100.00%	1 / 1
User.php		100.00%	8 / 8		100.00%	5 / 5		100.00%	1 / 1

Legend

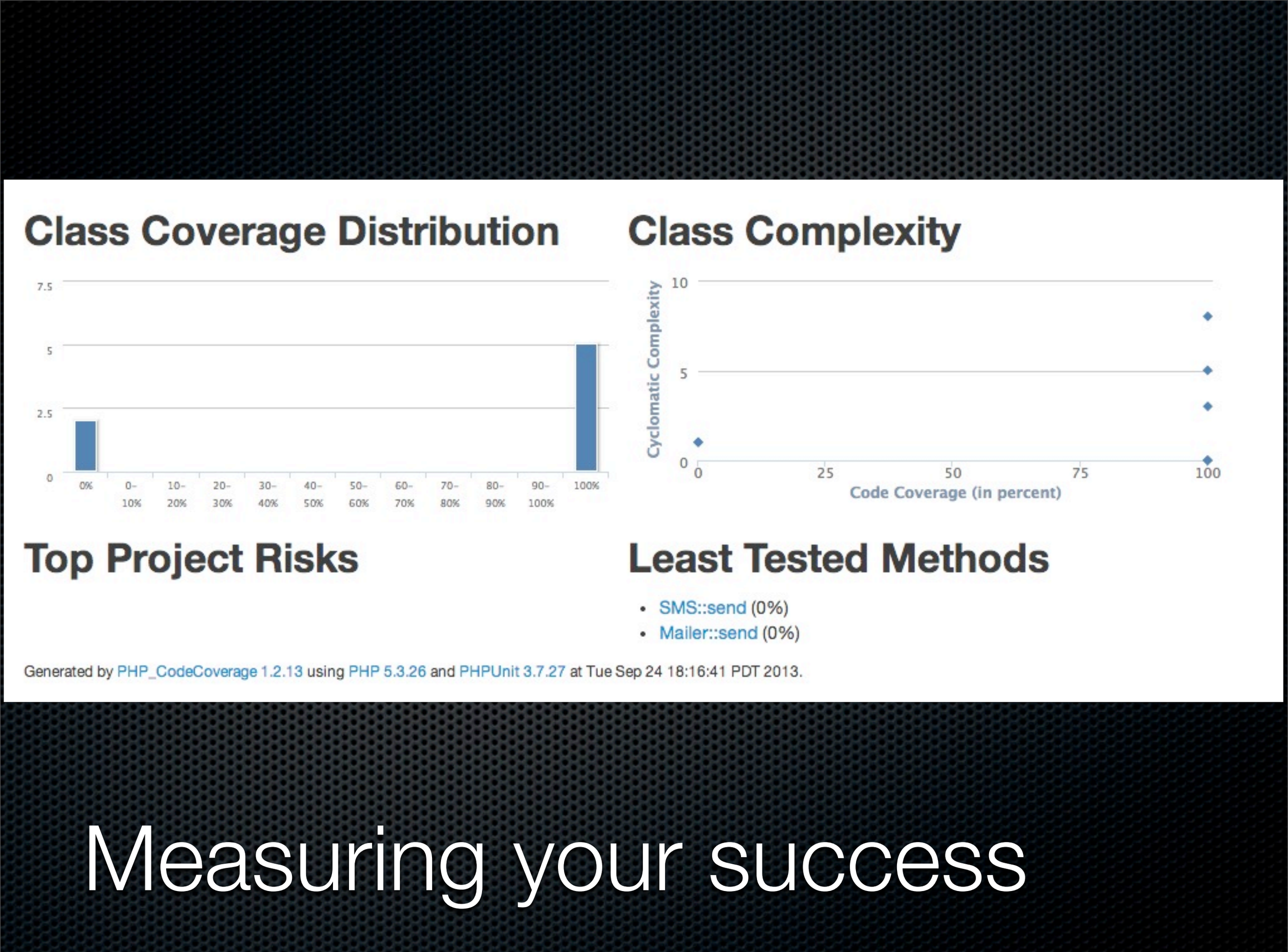
Low: 0% to 35% **Medium:** 35% to 70% **High:** 70% to 100%

Generated by PHP_CodeCoverage 1.2.13 using PHP 5.3.26 and PHPUnit 3.7.27 at Tue Sep 24 18:16:41 PDT 2013.

Measuring your success

How much of your code is being tested? PHPUnit can tell you.

Track at your project level.



Measuring your success

PHPUnit will also give you some pretty graphs and some top risks.

```
45      /**
46       * @param User $user
47       *
48       * @return Mailer|SMS
49       * @throws UnknownContactPreferenceException
50       */
51      public
52      function getNotifier(User $user) {
53          $pref = $user->get_notification_preference();
54
55          $notifier = null;
56
57          switch ($pref) {
58              case User::PREF_EMAIL:
59                  // harder to test
60                  // $notifier = new Mailer();
61                  $notifier = $this->create_mailer();
62
63                  break;
64              case User::PREF_SMS:
65                  // harder to test
66                  // $notifier = new SMS();
67                  $notifier = $this->create_sms();
68                  break;
69
70              default:
71                  throw new OCPHP\UnknownContactPreferenceException();
72
73          }
74
75          return $notifier;
76      }
77
78      /**
79       * @return Mailer
80       */
81      private
82      function create_mailer() {
83          return new Mailer();
84      }
85
```

Measuring your success

Which code do we know works. Which code are we less confidant in?

Green is covered by a test
Red is not

Set a target for code coverage

Parting thoughts...

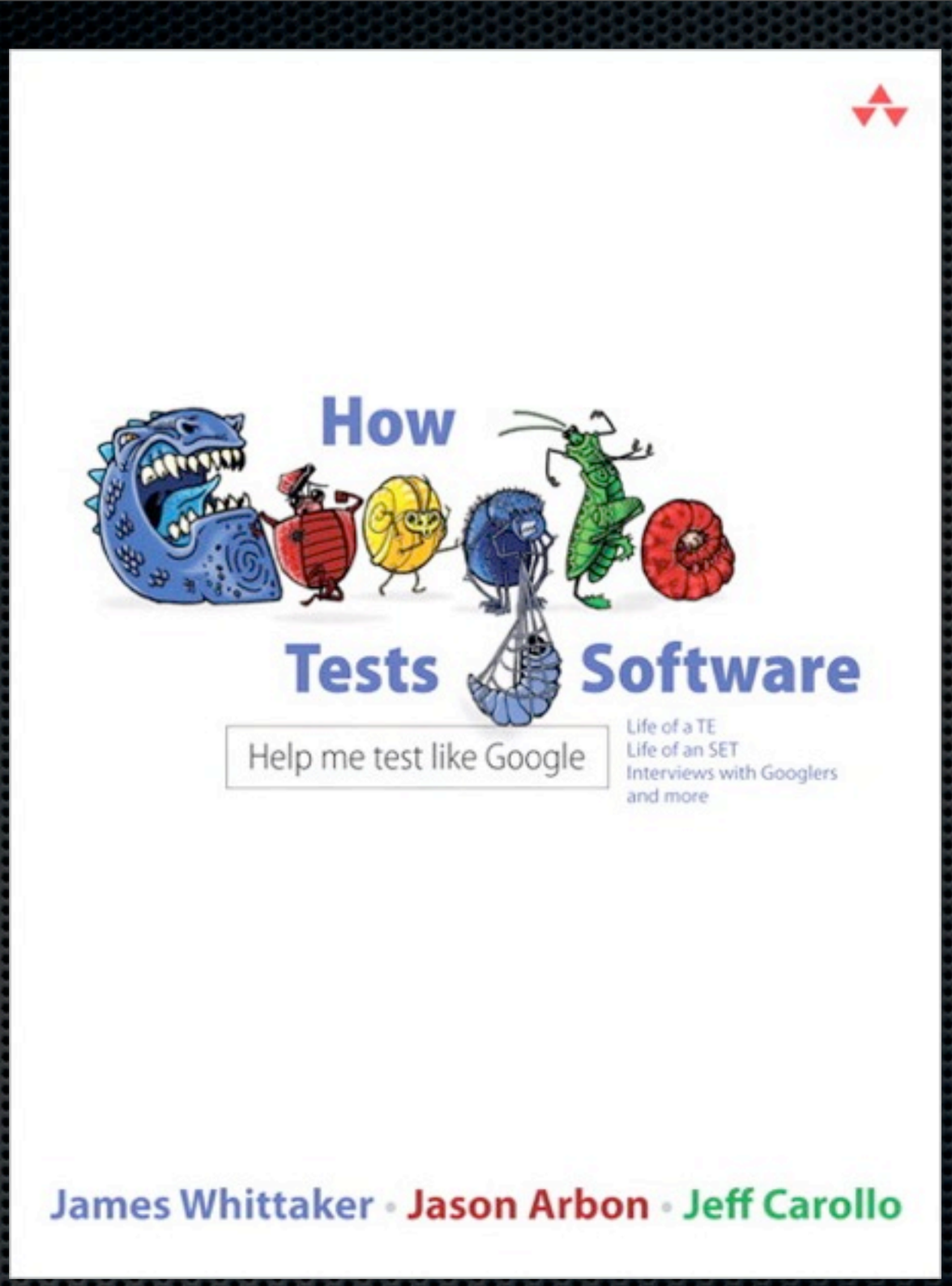
- Are you testing all your code? (JS)
- What's the point if there is anyone that doesn't run the tests? Continuous Integration!
- Integration tests
- Static Analysis

Plenty of JS unit testing frameworks. All languages have testing frameworks.

What's the point if they aren't ran religiously? Difficult to track down the source of the problem if they are not run for every commit. Please please integration with CI.

Write integration tests too (browser automation, Selenium, etc). While all your units might work perfectly, they might not work together.

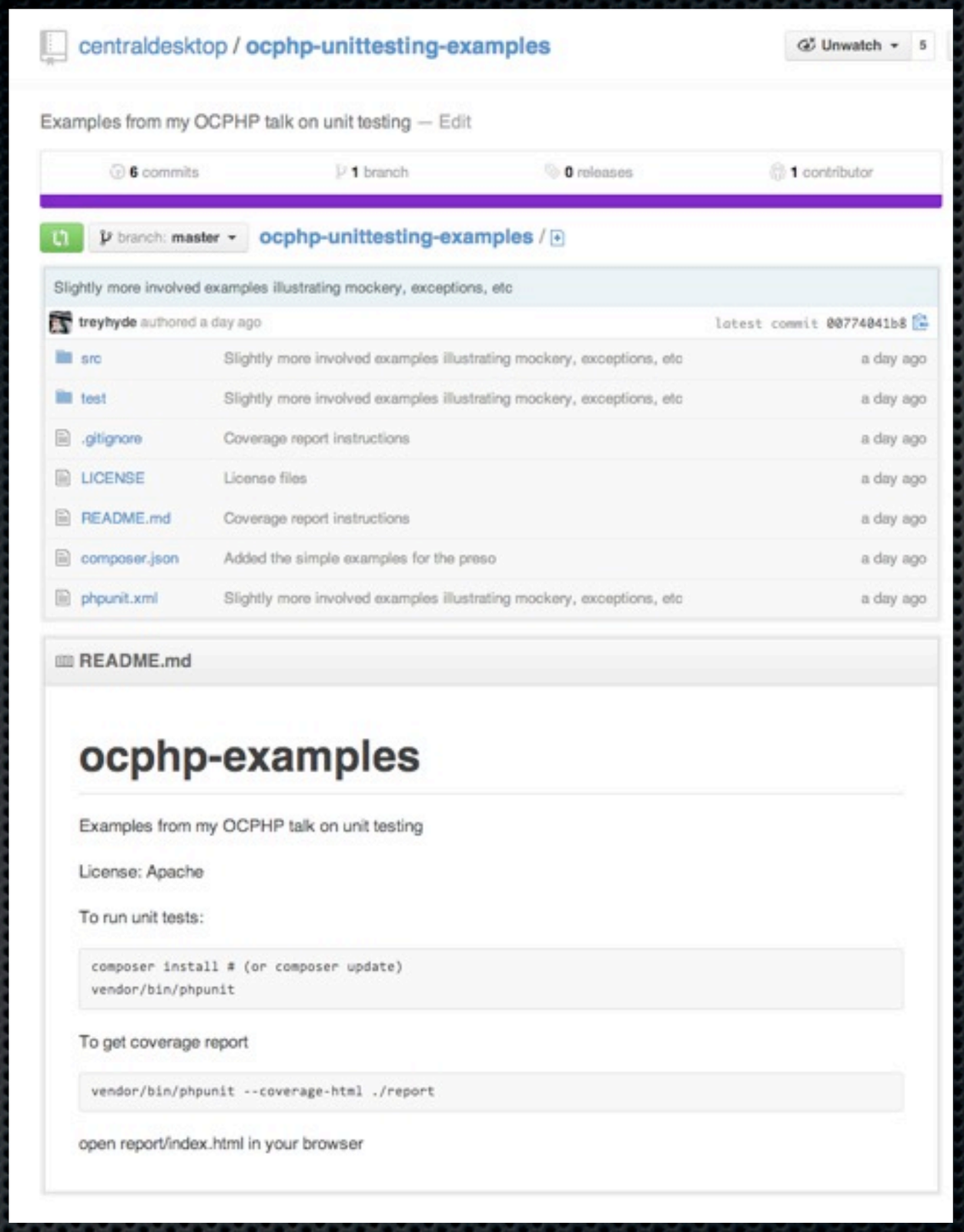
I've done whole talks on static analysis. Have tools analyze your code quality and make suggestions for improvement. THESE TOOLS FIND BUGS AUTOMATICALLY!



Go read this book right now

I can't believe I made it though a whole presentation without mentioning this book.

We've talked about tools, but we really need to talk about processes too.



All examples on github

https://github.com/centraldesktop/ocphp-unittesting-examples. Presentation will be uploaded too.

Resources

- http://en.wikipedia.org/wiki/Unit_Test
- http://toranbillups.com/blog/archive/2010/04/22/How-test-first-development-changed-my-life/
- http://phpunit.de/manual/current/en/index.html
- http://martinfowler.com/articles/mocksArentStubs.html
- http://www.jbrains.ca/permalink/the-worlds-best-intro-to-tdd-demo-video
- http://www.amazon.com/Google-Tests-Software-James-Whittaker/dp/0321803027
- http://misko.hevery.com/code-reviewers-guide/
- http://net.tutsplus.com/tutorials/php/mockery-a-better-way/
- http://www.agilemodeling.com/essays/costOfChange.htm
- http://symfony.com/doc/current/components/dependency_injection/introduction.html

Go read these, this talk breezed over all sorts of important details

I’m always looking for test infected engineers

thyde@centraldesktop.com

@treyhyde

https://plus.google.com/118381235166842329468/posts