Unit Test Your Code
With PHPUnit

Matthew Radcliffe
mradcliffe
@mattkineme

kosada incorporated

vuo
Software Quality Assurance

• Prevent defects from making it to the customer:
  • Adopt standards and specifications
  • Review code
  • Manage releases
  • Test code
Value of Testing

- Increase reliability and quality of software.
  - “I didn’t think of that!”
- Discover regressions in software.
  - “Why did it break?”
- Improve confidence in our code.
  - “I think this will work.”
## Common Complaints Regarding Tests

<table>
<thead>
<tr>
<th>Issue</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writing tests takes too long.</td>
<td>Start small. 100% coverage isn’t going to come in a day.</td>
</tr>
<tr>
<td>I don’t have source control / version control.</td>
<td>Do not pass go. Do not write any more code. Go directly to a Git repository.</td>
</tr>
<tr>
<td>I don’t have any testing infrastructure</td>
<td>Run locally, enforce social contract. Or setup TravisCI publicly or privately.</td>
</tr>
<tr>
<td>I don’t know what I’m going to write until I write it.</td>
<td>Not everyone needs to adopt Test-Driven Development, but it is “best practice”.</td>
</tr>
<tr>
<td>My code is heavily integrated with state (database or web services)</td>
<td>That’s where test doubles come into play, but you may be stuck in similar to Drupal 6-7.</td>
</tr>
</tbody>
</table>
Some Types of Testing

- User Acceptance Test: Test according to specification or requirement.

- Functional Test: Test one function (or feature) expected output.

- Unit Test: Test the smallest “unit” of testable code (in isolation).

- Integration Test: Test the interaction of “units” in 1 or more systems.

- Behavioral Test: Automated UAT or black box testing.

- Stress Test: Test the product under heavy load/stress.
Unit Tests

• A unit is the smallest testable piece of code, which is often a function, class or method.

• Plug a set of inputs into that code, and confirm the expected output (like behavioral and integration tests).

• Units should act in memory and not depend on other systems.

• Should be fast.

• Run all unit tests after code change.
Testing: The Basic Idea

- Fibonacci Sequence \([0, 1, 1, 2, 3, 5, \ldots N]\)

- Given a number assert the correct Fibonacci number in the sequence.

- Inputs: 1, 5, 10, 100

- Output: 0, 8, 55, 354224848179261915075
<?php
// fibonacci.php

// Well I only need to know this up to 5, so why bother?

function fibonacci($number) {
    $sequence = [0 => 0, 1 => 1, 2 => 1, 3 => 2, 4 => 5, 5 => 8];
    return $sequence[$number];
}

<?php
// fibonacciTest.php

require_once "fibonacci.php";

function testFibonacci() {
    // Assert our expected output.
    var_dump((fibonacci(0) === 0) === TRUE);
    var_dump((fibonacci(5) === 8) === TRUE);
    var_dump((fibonacci(10) === 55) === TRUE);
    var_dump((fibonacci(100) === 354224848179261915075) === TRUE);
    var_dump((fibonacci(-5) === 5) === TRUE);
    var_dump((fibonacci(-6) === -8) === TRUE);
}

testFibonacci();
PHP Test Frameworks

• **SimpleTest**
  
  • Also allows for unit and integration tests in the same framework.

• **PHPUnit**
  
  • De facto standard for PHP applications and frameworks.
  
  • Annotation-based test discovery and handles testing Exceptions.
  
  • Coverage report with static code analysis.
  
  • Supports introspection for test doubles.
  
  • Travis CI has built-in support.
PHPUnit: Getting Started

• Composer: composer require phpunit/phpunit:4.8.*

• phpunit.xml

• bootstrap.php

• Test class

• Run it: ./vendor/bin/phpunit
phpunit.xml

1  <?xml version="1.0" encoding="UTF-8"?>
2  <phpunit bootstrap="tests/bootstrap.php"></phpunit>
<?php
require_once __DIR__ . '/../fibonacci.php';
<?php

class FibonacciTest extends PHPUnit_Framework_TestCase {

    function testFibonacci() {
        $this->assertEquals(0, fibonacci(0));
        $this->assertEquals(8, fibonacci(5));
        $this->assertEquals(55, fibonacci(10));
        $this->assertEquals(354224848179261915075, fibonacci(100));
        $this->assertEquals(5, fibonacci(-5));
        $this->assertEquals(-8, fibonacci(-6));
    }
}

PHPUnit: Assertions

- assertEquals
- assertEmpty
- assertSame
- assertInstanceOf
- assertXmlStringEqualsXmlFile
- assertArrayHasKey
- assertArrayHasSubset
- assertCount
- assertFileExists
- assertNotNull

PHPUnit: Data Providers

- A data provider is a method that returns an array of parameters to pass into a test method.
  - A test method may test several inputs.
- Important to note that data provider methods are run before any setup so this cannot depend on any test doubles.
function sequenceProvider() {
    return [
        [0, 0],
        [5, 8],
        [10, 55],
        [100, 354224848179261915075],
        [-5, 5],
        [-6, 8]
    ];
}

/**
 * Test class with data provider.
 *
 * @dataProvider sequenceProvider
 */
function testFibonacciWithProvider($number, $expected) {
    $this->assertEquals($expected, fibonacci($number));
}
PHPUnit: setUp

• The setUp method is executed for every test method in a class.

• Used to configure fixtures or setting up a known state.

• Useful for setting up test doubles or mocks, which are dependencies of a unit that you do not need to test in that test class.

• A complex setUp helps to isolate design problems.
PHPUnit: Test Doubles

• Test doubles (or mock objects) allow to focus an unit test on the code that needs to be tested without bootstrapping dependencies.

• Example: I don’t want to load Drupal 8’s Entity system when I test my unrelated code. Instead PHPUnit allows to create a mock via Reflection so that I get an object that looks like an Entity.

• Reflection or introspection allows a program to know about and modify itself at runtime.
PHPUnit: Test Doubles

- `getMockBuilder()`
  - `disableOriginalConstructor()`, `getMock()`

- `method()`

- `with()`, `withConsecutive()`
  - `callback()`

- `will()`, `willReturn()`
  - `onConsecutive()`, `returnValueMap()`
<?php

class BlahTest extends PHPUnit_Framework_TestCase {

    protected function setUp() {

        $reader = $this->getMockBuilder('League\Csv\Reader')
            ->disableOriginalConstructor()
            ->getMock();

        $reader->expects($this->any())
            ->method('fetchAll')
            ->willReturn(
                [['john', 'doe', 'john.doe@example.com'],
                ['jane', 'doe', 'jane.doe@example.com']]);

        // Calling $reader->fetchAll() will return the array mocked above.
    }
}
Overview of Continuous Integration

• Every commit is tested.

• How is this achieved?
  • Automated test and build environment.

• What are some of my options?
  • Jenkins
  • Travis CI
  • Bamboo (Atlassian)
  • Some other proprietary solution
Travis CI for Open Source

- Anyone can setup Travis CI for their open source repository on GitHub.
  - Go to travis-ci.org, login with github, add your project.
  - Add a .travis.yml, validate it, commit, and push.
  - Test on PHP 5, 7, HHVM, etc…, add dependencies via composer, and setup databases into your build matrix.
language: php

php:
- 5.4
- 5.5
- 5.6

sudo: false

install:
- composer self-update

before_script:
- composer install --prefer-source -n

Putting it all Together

• mradcliffe/pokerhand
  • Write a test for Bill Condo’s poker hand feed.
  • Refactor code to support that test.
Code
Questions & Challenge

- Twitter: @mattkineme
- GitHub: @mradcliffe
- DrupalCamp Ohio 2015
  - 10/23/2015 - 10/24/2015

http://softpixel.com/~mradcliffe/#!/articles