PHPUnit and Drupal 8

No Unit Left Behind

Paul Mitchum / @PaulMitchum

And You Are?

Paul Mitchum
Mile23 on Drupal.org
From Seattle

Please Keep In Mind

The ability to give an hour-long presentation is not the same as being an expert.

The Structure Of This Talk

- What's A Unit Test?
- Use The Tools
- Who Eats Tests?
- Writing A Test
- Patterns For Isolation
- Questions?
- ...until I run out of time

What's A Unit Test?

A test of a unit. Duh.

So What's A Unit?

A unit is:

- Function / Method
  - Code path within a method
  - Line of code within a method

```
$this->assertTrue(
  SomeClass::thisMethodIsAUnit($data)
);
```

Why do we need unit tests?

Some concrete benefits:

1. They quickly tell us whether the code works.
2. They allow us to analyze the maintainability of our code.

Whither SimpleTest

SimpleTest has web-based functional tests. These test system interactions rather than methods in code.

It's a different testing realm.

SimpleTest can stay. :-)

Tools

- How to run PHPUnit tests.
- How to generate PHPUnit coverage report.
- How to read the coverage report.

How to run PHPUnit

(the right way)

1. At the command line...
2. `cd core/`
3. `./vendor/bin/phpunit`
4. Wait 8 seconds.
5. Know the truth.

(demo)

How to run PHPUnit inside Drupal 8

(the wrong way)

1. Have XDebug.
2. Enable Testing module.
3. At the Drupal 8 testing page.
4. Find 'PHPUnit' group.
5. Click 'Run tests.'

How to run PHPUnit like the testbot

(the less wrong way)

1. With a working Drupal installation.
2. Enable the Testing module.
3. At the command line...
4. `php core/scripts/run-tests.sh PHPunit`
5. Wait a while.
6. Know the truth.

(demo)

The problem of coverage

`Unicode::convertToUtf8()`

(demo)

How to generate a coverage report

Make sure you have XDebug

Same as running the test...

```
./vendor/bin/phpunit --coverage-html [path]
```

Will generate static HTML site.

(crap-free demo)

What About SimpleTest?

SimpleTest coverage is not reflected in PHPUnit reports.

SimpleTests are functional, so they don't have the same kind of coverage as PHPUnit.

The problem of coverage

Missing files in `Database`

(demo)

Walk Through The Dashboard

(demo)

CRAP?

Yes, C.R.A.P.: Change Risk Anti-Pattern.

- Measures maintainability of code.
- *Not* a value judgement. :-)
- Low number is better. 1 is minimum.
- 30 is generally considered high.

How does CRAP work?

- Measures cyclomatic complexity of the method.
- Measures line coverage.
- Given coverage, importance of complexity is reduced.
- Basically: High coverage kills CRAP.

The Lesson Of CRAP

Coverage means not having to worry.

The problem of maintainability

`DisplayPluginBase::buildOptionsForm()`

(demo)

Who Eats Tests?

or

I wrote a test, what can I do with it?

You have some knowledge of the tools, let's put them in context.

Continuous Integration

A process for testing all changes as they happen.

Make a change, quickly find out if it's bad.

As automatic as possible.

1. git commit
2. Push to repo
3. CI process triggered
4. Automatically deploy and run all the tests
5. Pass/fail report
6. Know

https://github.com/paul-m/drupal-travis-ci

Drupal.org Testbot Process

Almost a CI process, but not quite. Still very useful.

1. Submit patch
2. Testbot applies patch
3. Testbot runs tests
4. Testbot shows status
5. Know (once you look at the issue again)

Not CI: You have to make the patch (and the interdiff), and it doesn't automatically tell you the results.

Test-Along Methodology

Obsessively run tests while developing, because you can.

- Keep a terminal window open.
- Command-tab to terminal, hit up arrow and return.
- Know.

A Modest Proposal For A Development Process

You're a developer. Your assignment: Fix a bug.

1. Generate a coverage report.
2. If coverage is low in your area of concern:
3. Write unit tests for high coverage baseline.
4. Begin fixing bug.
5. Run tests.
6. More bug fixing.
7. Run tests.
8. Etc.

PHPUnit/CI enables this process.

Essentially:

Your test will run every time anyone submits a patch.

The test you write will be a metric used to measure the quality of other code.

Your code and your test will be eaten by everyone else. Yum.

For-Reals PHPUnit Code Stuff

PHPUnit Definitions

Lots of overlap with SimpleTest

- **Assertion**: Testing logic, pass/fail.
- **Test**: A method that uses assertions.
- **Test case**: A class with test methods.
- **Test suite**: Which test cases to run. Config.

Drupal 8 PHPUnit Test Case

```
// file: [PSRpath]/FooTest.php
// namespace Drupal\Tests\Core\Whatever;
namespace Drupal\[module]\Tests;

// Drupal\Tests\UnitTestCase extends \PHPUnit_Framework_TestCase
use Drupal\Tests\UnitTestCase;

class FooTest extends UnitTestCase { // <- test case
  public function testFoo() {        // <- test
    $this->assertTrue(TRUE);         // <- assertion
  }
}
```

How Does PHPUnit Find Tests?

1. Ignore class autoloader.
2. Read config for directories to search.
3. Recursively search through directories.
4. Look for files ending in `Test.php`
5. `include` those files.
6. Check if they're subclasses of `\PHPUnit_Framework_TestCase`
7. Check those classes have method names starting with `test`

File System

```
core/
  lib/
    [PSR Namespace Here]
  tests/
    [PSR Namespace Here]
modules/
  someModule/
    someModule.info.yml
    lib/
    tests/
      [PSR Namespace...]
      Drupal/
        someModule/
          Tests/
            SomeModuleTest.php
```

PSR-What?

BASICALLY...

- PHP might need help finding a class.
- You can use `spl_autoload_register()` to help it out.
- Various PSR systems are strategies for finding the file.

YEAH, BUT HOW?

In Drupal PSR-0, namespace maps to file path.

`Drupal\Core\Something\Else`

becomes

`[root dir]/Drupal/Core/Something/Else.php`

[isolation]

Isolation

Enables us to test only the thing we're trying to test.

Decouples the test from other concerns.

Types of isolation:

- System isolation: No database, no server
- Language isolation: Pick out extensions and libraries
- Code isolation: Let PHPUnit do the work

Patterns for Isolation

- Data Providers
- @expectedException
- Test Doubles (mock, stub)
- Dependency Injection
- Interfaces

Pattern: Data Provider

A data provider is a method that returns an array of data which PHPUnit then iterates into the test method's parameters.

```
public function providerTestSomething() {
  return array( array('expected', 'data'), );
}

/**
 * Special annotation:
 * @dataProvider providerTestSomething
 */
public function testSomething($expected, $data) {
  $something = new Something();
  $this->assertEquals($expected, $something->testMe($data));
}
```

More on Data Providers

Once a unit test is written, it becomes: The Test™

You want its regression value to stay the same.

When you change the test, you are now testing something else.

Data providers give us a way to change test data without changing test logic.

ALWAYS write a data provider, for any data-based test you write. The test method should not depend on specific data.

Anti-Pattern: Exceptions

Example without @expectedException annotation.

```
/**
 * @dataProvider providerTestException
 */
public function testException($boom)
{
  try {
    $item = new \Some\Class();
    $item->badDataMakesMeGo($boom);
  }
  catch (\InvalidArgumentException $e) {
    $this->assertTrue(TRUE); // PASS
    return;
  }
  catch ($e) {
  }
  $this->assertTrue(FALSE); // FAIL
}
```

Pattern: Expected Exception

Test passes if an exception is thrown.

Isolates test from test code.

```
/**
 * @expectedException \InvalidArgumentException
 * @dataProvider providerTestException
 */
public function testException($boom) {
  $item = new \Some\Class();
  $item->badDataMakesMeGo($boom);
}
```

Pattern: Test Double

PHPUnit can provide an imposter object which you can program to do stuff.

This is a ‘test double.’

Test doubles perform stubbing or mocking of items needed by the system under test.

Isolate behavior of SUT from other implementations.

Test Double

We want to test this:

```
Class_A::wtfMethod(\Interface_B $b);
```

Let's make a test double:

```
public function testWtfMethod() {
  // Make a mock object.
  $mock = $this->getMock('\Interface_b');

  // Tell it to handle the given method.
  $mock->expects($this->any())
    ->method('mockedMethod')
    ->will($this->returnValue('iAmAnExpectedValue'));

  // And finally run the test.
  $this->assertTrue(Class_A::wtfMethod($mock));
}
```

Drupal 8 Built-In Test Doubles

`Drupal\Tests\UnitTestCase` has some test double convenience methods.

Give them a look for clues about how to make test doubles.

- `getConfigFactoryStub()`
- `getConfigStorageStub()`
- `getBlockMockWithMachineName()`
- `getStringTranslationStub()`

Winding Up: Recap

- PHPUnit: `./vendor/bin/phpunit`
- Coverage Report: `--coverage-html [path]`
- CRAP: Reflects maintainability
- Functional vs. Unit Testing: Systems vs. Code
- Patterns: Isolation, Data providers, Test Doubles

Thank You!

Paul Mitchum

Mile23 on Drupal.org

@PaulMitchum on Twitter

paul-m on GitHub

http://mile23.com/phpunit-talk

Questions?

1. Good: I know the answer.
2. Excellent: I know the answer and have a slide.
3. Interesting: I have no clue what you are talking about.