



The PHP Company

# Testing Your Zend Framework MVC Application

Matthew Weier O'Phinney  
Project Lead  
Zend Framework

# What we'll cover

---

- Unit Testing basics
- Basic functional testing in ZF
- Several “Advanced” ZF testing topics

# Why test?

# Simplify maintenance

---

- Testing defines expectations
- Testing describes behaviors identified by the application
- Testing tells us when new changes break existing code and behaviors

# Quantify code quality

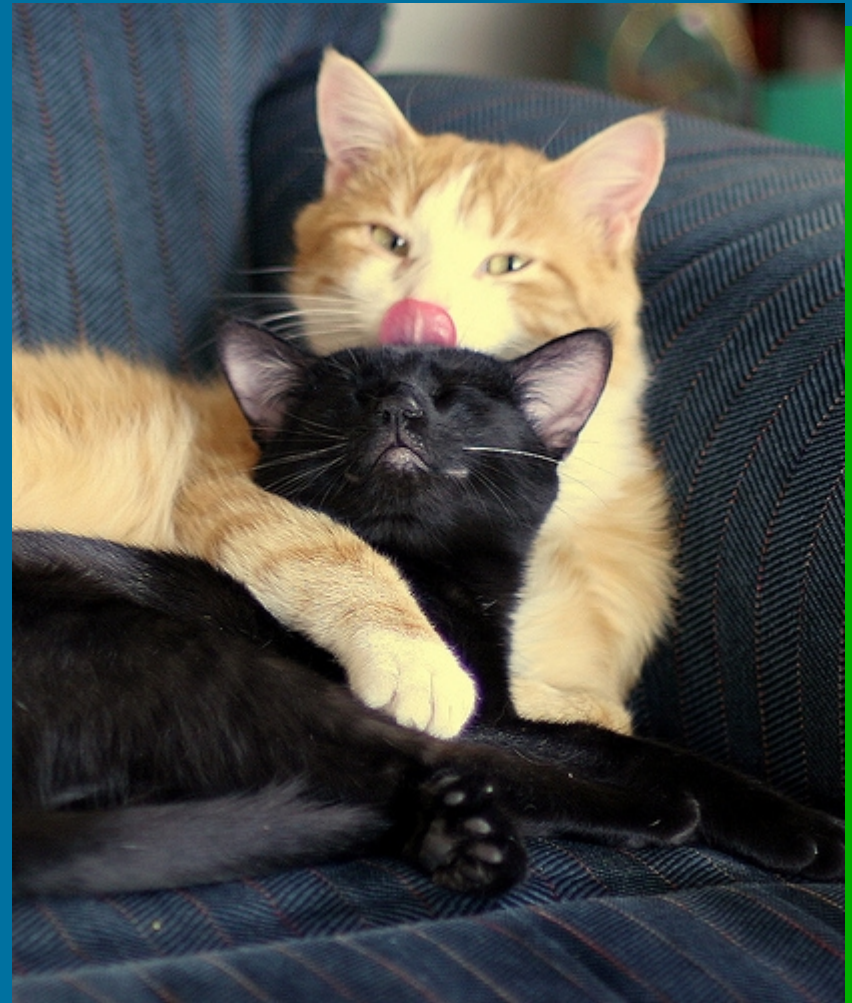
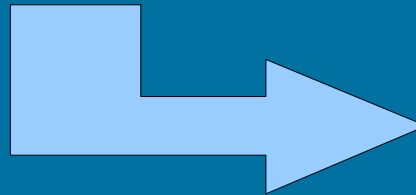
---

- Code coverage exercised by unit tests
- Test methods document behaviors code should define

# Psychological benefits

*Warm fuzzy feeling  
from seeing green*

```
matthew@zlaptop ~-framework/tests  
% phpunit --colors Zend/Dom  
PHPUnit 3.4.2 by Sebastian Bergmann.  
.....  
Time: 1 second  
OK (38 tests, 58 assertions)
```



# Testing is not... reloading



# Testing is not... var\_dump()

```
}
["_POST"] =>
array(1)
  ["date"] =>
string(0) ""
  ["event_date"] =>
string(0) ""
  ["wksp_type"] =>
string(0) ""
  ["wksp_vers"] =>
string(0) ""
  ["wksp_num"] =>
string(7) "F096341"
  ["search"] =>
string(15) "Search (Alt-S)"
  ["/cgi-bin/auth_secure.cgi_L"] =>
string(10) "1188159806"
  ["PHPSESSID"] =>
string(32) "0a6938bflc9a941b5270d2...75600"
}
```

# Testing is... reproducible

```
Terminal
File Edit View Terminal Tabs Help
sb@ubuntu PHPUnit % ll Examples
total 20K
-rw-r--r-- 1 sb sb 1.4K 2008-03-30 17:03 BowlingGame.php
-rw-r--r-- 1 sb sb 1.4K 2008-03-30 17:03 BowlingGameTest.php
-rw-r--r-- 1 sb sb 225 2008-03-30 17:03 Schaltjahr.php
-rw-r--r-- 1 sb sb 386 2008-03-30 17:03 SchaltjahrTest.php
-rw-r--r-- 1 sb sb 1.5K 2008-04-09 09:57 WebTest.php
sb@ubuntu PHPUnit % phpunit Examples
PHPUnit @package_version@ by Sebastian Bergmann.

E.....

Time: 0 seconds

There was 1 error:

1) testTitle(WebTest)
RuntimeException: Could not connect to the Selenium RC server.

FAILURES
Tests: 7, Errors: 1.
sb@ubuntu PHPUnit %
```

# Testing is... automatable

The screenshot displays the phpUnderControl web interface. At the top, the project is identified as 'phpuc-merge-test' and the build as 'More builds'. The interface includes navigation tabs for Overview, Tests, Metrics, Coverage, Documentation, CodeSniffer, and PHPUnit PMD. The 'Tests' tab is active, showing a table of test results for 'Example::PhpUnderControl\_Example\_MathTest'.

Name	Status	Time(s)
Example::PhpUnderControl_Example_MathTest		0.101
testAddSuccess	Success	0.012
#1 - testAddSuccess - (build: <b>php-5.3.0</b> )	Success	0.006
#2 - testAddSuccess - (build: <b>php-5.2.6</b> )	Success	0.006
#3 - testAddSuccess - (build: <b>php-5.2.5</b> )	Success	0.001
testSubSuccess	Success	0.012
#1 - testSubSuccess - (build: <b>php-5.3.0</b> )	Success	0.006
#2 - testSubSuccess - (build: <b>php-5.2.6</b> )	Success	0.006
#3 - testSubSuccess - (build: <b>php-5.2.5</b> )	Success	0.000
testSubFail		0.015
#1 - testSubFail - (build: <b>php-5.3.0</b> )	Failure >	
#2 - testSubFail - (build: <b>php-5.2.6</b> )	Failure >	
#3 - testSubFail - (build: <b>php-5.2.5</b> )	Failure >	
testDataProviderOneWillFail	Success	0.051
testDataProviderOneWillFail - (build: <b>php-5.3.0</b> )		0.024
#1 - testDataProviderOneWillFail with data set #0	Success	0.006
#2 - testDataProviderOneWillFail with data set #1	Success	0.006
#3 - testDataProviderOneWillFail with data set #2	Failure >	
#4 - testDataProviderOneWillFail with data set #3	Success	0.006
testDataProviderOneWillFail - (build: <b>php-5.2.6</b> )		0.024
#1 - testDataProviderOneWillFail with data set #0	Success	0.006
#2 - testDataProviderOneWillFail with data set #1	Success	0.006
#3 - testDataProviderOneWillFail with data set #2	Failure >	
#4 - testDataProviderOneWillFail with data set #3	Success	0.006
testDataProviderOneWillFail - (build: <b>php-5.2.5</b> )		0.002
#1 - testDataProviderOneWillFail with data set #0	Success	0.001
#2 - testDataProviderOneWillFail with data set #1	Success	0.000
#3 - testDataProviderOneWillFail with data set #2	Failure >	
#4 - testDataProviderOneWillFail with data set #3	Success	0.001
testFail		0.011
#1 - testFail - (build: <b>php-5.3.0</b> )	Failure >	
#2 - testFail - (build: <b>php-5.2.6</b> )	Failure >	
#3 - testFail - (build: <b>php-5.2.5</b> )	Failure >	

# Good testing includes...

---

- Defined behaviors
- Code examples of use cases
- Expectations

# PHP testing frameworks

---

- PHPT
  - ▶ Used by PHP, and some PEAR and independent libraries
- SimpleTest
  - ▶ JUnit-style testing framework
- PHPUnit
  - ▶ JUnit-style testing framework
  - ▶ *De facto* industry standard

# Testing Basics

# Writing unit tests

---

- Create a test class
- Create one or more methods describing behaviors
  - ▶ State the behaviors in natural language
- Write code that creates the behavior(s)
  - ▶ Write code exercising the API
- Write assertions indicating expectations

# Create a test class

---

- Usually, named after the Unit Under Test

```
class EntryTest
    extends PHPUnit_Framework_TestCase
{
}
```

# Write a method describing a behavior

---

- Prefix with “test”

```
class EntryTest
    extends PHPUnit_Framework_TestCase
{
    public function testMaySetTimestampWithString()
    {
    }
}
```

# Write code creating the behavior

---

```
class EntryTest
    extends PHPUnit_Framework_TestCase
{
    public function testMaySetTimestampWithString()
    {
        $string = 'Fri, 7 May 2010 09:26:03 -0700';
        $ts     = strtotime($string);
        $this->entry->setTimestamp($string);
        $setValue = $this->entry->getTimestamp();
    }
}
```

# Write assertions of expectations

```
class EntryTest
    extends PHPUnit_Framework_TestCase
{
    public function testMaySetTimestampWithString()
    {
        $string = 'Fri, 7 May 2010 09:26:03 -0700';
        $ts     = strtotime($string);
        $this->entry->setTimestamp($string);
        $setValue = $this->entry->getTimestamp();
        $this->assertSame($ts, $setValue);
    }
}
```

# Run the tests

---

- Failure?
  - ▶ Check your tests and assertions for potential typos or usage errors
  - ▶ Check the Unit Under Test for errors
  - ▶ Make corrections and re-run the tests
- Success?
  - ▶ Move on to the next behavior or feature!

# Some testing terminology

# Test scaffolding

---

- Make sure your environment is free of assumptions
- Initialize any dependencies necessary prior to testing
- Usually done in the “setUp()” method

# Test doubles

---

- **Stubs**

*Replacing an object with another so that the system under test can continue down a path.*

- **Mock Objects**

*Replacing an object with another and defining expectations for it.*

# Additional testing types

---

- **Conditional testing**

*Testing only when certain environmental conditions are met.*

- **Functional and Integration tests**

*Testing that the systems as a whole behaves as expected; testing that the units interact as expected.*

# Quasi-Functional testing in Zend Framework

# Overview

---

- Setup the phpunit environment
- Create a TestCase based on ControllerTestCase
- Bootstrap the application
- Create and dispatch a request
- Perform assertions on the response

# The PHPUnit environment

---

- Directory structure

```
tests
|-- application
|   `-- controllers
|-- Bootstrap.php
|-- library
|   `-- Custom
`-- phpunit.xml

4 directories, 2 files
```

# The PHPUnit environment

- phpunit.xml

```
<phpunit bootstrap="./Bootstrap.php">
  <testsuite name="Test Suite">
    <directory>./</directory>
  </testsuite>
  <filter>
    <whitelist>
      <directory
        suffix=".php">../library/</directory>
      <directory
        suffix=".php">../application/</directory>
      <exclude>
        <directory
          suffix=".phtml">../application/</directory>
        </exclude>
      </whitelist>
    </filter>
  </phpunit>
```

# The PHPUnit environment

- Bootstrap.php

```
$rootPath = realpath(dirname(__DIR__));  
if (!defined('APPLICATION_PATH')) {  
    define('APPLICATION_PATH',  
          $rootPath . '/application');  
}  
if (!defined('APPLICATION_ENV')) {  
    define('APPLICATION_ENV', 'testing');  
}  
set_include_path(implode(PATH_SEPARATOR, array(  
    '.',  
    $rootPath . '/library',  
    get_include_path(),  
)));  
require_once 'Zend/Loader/Autoloader.php';  
$loader = Zend_Loader_Autoloader::getInstance();  
$loader->registerNamespace('Custom_');
```

# Create a test case class

---

- Extend Zend\_Test\_PHPUnit\_ControllerTestCase

```
class ExampleControllerTest
    extends Zend_Test_PHPUnit_ControllerTestCase
{
}
```

# Bootstrap the application

- Create a Zend\_Application instance, and reference it in your setUp()

```
class ExampleControllerTest
    extends Zend_Test_PHPUnit_ControllerTestCase
{
    public function setUp()
    {
        $this->bootstrap = new Zend_Application(
            APPLICATION_ENV,
            APPLICATION_PATH
            . '/configs/application.ini'
        );
        parent::setUp();
    }
}
```

# Create and dispatch a request

- Simple method: dispatch a “url”

```
class ExampleControllerTest
    extends Zend_Test_PHPUnit_ControllerTestCase
{
    // ...
    public function testStaticPageHasGoodStructure ()
    {
        $this->dispatch ('/example/page' );
        // ...
    }
}
```

# Create and dispatch a request

- More advanced: customize the request object prior to dispatching

```
class ExampleControllerTest
    extends Zend_Test_PHPUnit_ControllerTestCase
{
    // ...
    public function testXHRRequestReturnsJson()
    {
        $this->getRequest()
            ->setHeader('X-Requested-With',
                'XMLHttpRequest')
            ->setQuery('format', 'json');
        $this->dispatch('/example/xhr-endpoint');
        // ...
    }
}
```

# Create assertions

---

- Typical assertions are for:
  - ▶ Structure of response markup  
*Using either CSS selectors or XPath assertions.*
  - ▶ HTTP response headers and status code
  - ▶ Request and/or Response object artifacts

# CSS Selector assertions

---

- `assertQuery($path, $message = "")`
- `assertQueryContentContains($path, $match, $message = "")`
- `assertQueryContentRegex($path, $pattern, $message = "")`
- `assertQueryCount($path, $count, $message = "")`
- `assertQueryCountMin($path, $count, $message = "")`
- `assertQueryCountMax($path, $count, $message = "")`
- **each has a "Not" variant**

# XPath Selector assertions

---

- `assertXPath($path, $message = "")`
- `assertXPathContentContains($path, $match, $message = "")`
- `assertXPathContentRegex($path, $pattern, $message = "")`
- `assertXPathCount($path, $count, $message = "")`
- `assertXPathCountMin($path, $count, $message = "")`
- `assertXPathCountMax($path, $count, $message = "")`
- **each has a "Not" variant**

# Redirect assertions

---

- `assertRedirect($message = "")`
- `assertRedirectTo($url, $message = "")`
- `assertRedirectRegex($pattern, $message = "")`
- **each has a "Not" variant**

# Response assertions

---

- `assertResponseCode($code, $message = "")`
- `assertHeader($header, $message = "")`
- `assertHeaderContains($header, $match, $message = "")`
- `assertHeaderRegex($header, $pattern, $message = "")`
- **each has a "Not" variant**

# Request assertions

---

- `assertModule($module, $message = "")`
- `assertController($controller, $message = "")`
- `assertAction($action, $message = "")`
- `assertRoute($route, $message = "")`
- **each has a "Not" variant**

# Assertion examples

---

```
public function testSomeStaticPageHasGoodStructure ()
{
    $this->dispatch ('/example/page');
    $this->assertResponseCode (200);
    $this->assertQuery ('div#content p');
    $this->assertQueryCount ('div#sidebar ul li', 3);
}
```

# Assertion examples

---

```
public function testXhrRequestReturnsJson()
{
    // ...
    $this->assertNotRedirect();
    $this->assertHeaderContains(
        'Content-Type', 'application/json');
}
```

# Advanced Topics or: real world use cases

# Testing models and resources

---

- **The Problem:**

*These classes cannot be autoloaded with the standard autoloader.*

- **The Solution:**

*Use Zend\_Application to bootstrap the “modules” resource during setUp()*

# Testing models and resources

```
class Blog_Model_EntryTest
    extends PHPUnit_Framework_TestCase
{
    public function setUp()
    {
        $this->bootstrap = new Zend_Application(
            APPLICATION_ENV,
            APPLICATION_PATH
                . '/configs/application.ini'
        );
        $this->bootstrap->bootstrap('modules');

        $this->model = new Blog_Model_Entry();
    }
}
```

# Testing actions requiring authentication

---

- **The Problem:**

*Some actions may require an authenticated user; how can you emulate this?*

- **The Solution:**

*Manually authenticate against Zend\_Auth prior to calling dispatch().*

# Authenticating a test user

```
class ExampleControllerTest
    extends Zend_Test_PHPUnit_ControllerTestCase
{
    // ...
    public function loginUser($user)
    {
        $params = array('user' => $user);
        $adapter = new Custom_Auth_TestAdapter(
            $params);
        $auth = Zend_Auth::getInstance();
        $auth->authenticate($adapter);
        $this->assertTrue($auth->hasIdentity());
    }
}
```

# Authenticating and dispatching

---

```
class ExampleControllerTest
    extends Zend_Test_PHPUnit_ControllerTestCase
{
    // ...
    public function testAdminUserCanAccessAdmin()
    {
        $this->loginUser('admin');
        $this->dispatch('/example/admin');
        $this->assertQuery('div#content.admin');
    }
}
```

# Testing pages dependent on prior actions

---

- **The Problem:**

*Some actions are dependent on others; e.g., retrieving a page with content highlighted based on a search string.*

- **The Solution:**

*Dispatch twice, resetting the request and response between calls.*

# Testing pages dependent on prior actions

```
class ExampleControllerTest
    extends Zend_Test_PHPUnit_ControllerTestCase
{
    // ...
    public function testHighlightedTextAfterSearch()
    {
        $this->getRequest()->setQuery(
            'search', 'foobar');
        $this->dispatch('/search');

        $this->resetRequest();
        $this->resetResponse();

        $this->dispatch('/example/page');
        $this->assertQueryContains(
            'span.highlight', 'foobar');
    }
}
```

# Conclusions

# Test Always!

---

- Unit test your models, service layers, etc.
- Do functional/acceptance testing to test workflows, page structure, etc.

# Zend Framework Training & Certification

---

- **Zend Framework: Fundamentals**

This course combines teaching ZF with the introduction of the Model-View-Controller (MVC) design pattern, to ensure you learn current best practices in PHP development.

Next Class: August 16, 17, 18, 19 20, 23, 24, 25 & 26 from 9am-11am Pacific

# Zend Framework Training & Certification

---

- **Zend Framework: Advanced**

The Zend Framework: Advanced course is designed to teach PHP developers already working with Zend Framework how to apply best practices when building and configuring applications for scalability, interactivity, and high performance.

Next Class: Sept. 7, 8, 9, 10, 13, 14, 15, 16 & 17 from 8:30am-10:30am Pacific

# Zend Framework Training & Certification

---

- **Test Prep: Zend Framework Certification**  
The Test Prep: Zend Framework Certification course prepares experienced developers who design and build PHP applications using ZF for the challenge of passing the certification exam and achieving the status of Zend Certified Engineer in Zend Framework(ZCE-ZF).
- **Free Study Guide**  
<http://www.zend.com/en/download/173>

# ZendCon 2010!

---

- 1–4 November 2010
- <http://www.zendcon.com/>