

PHP DEVELOPER BEST PRACTICES

Mike Naberezny
Matthew Weier O'Phinney

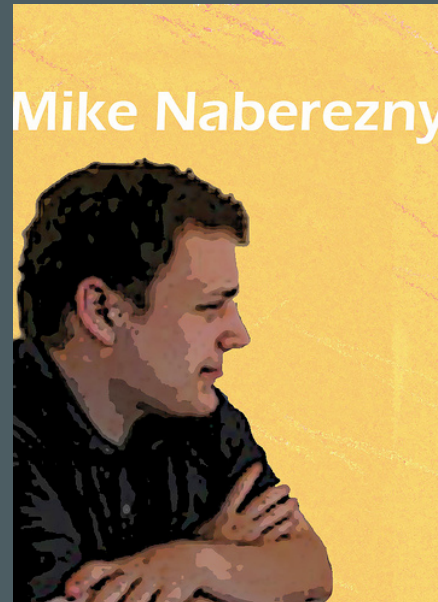


Mike Naberezny

- <http://mikenaberezny.com>
- <http://maintainable.com>
- <http://ohloh.net/accounts/mnaberez>



ZendCon '06

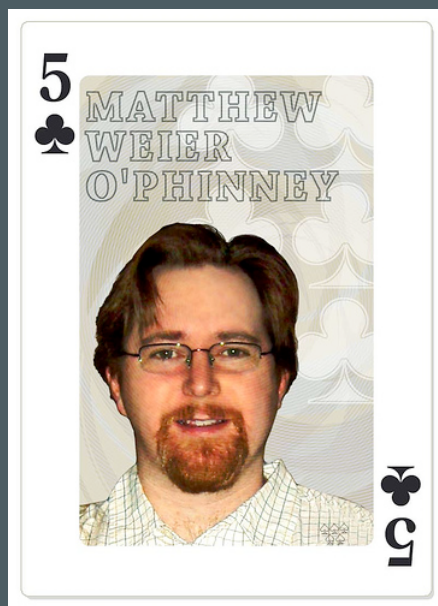


ZendCon '07

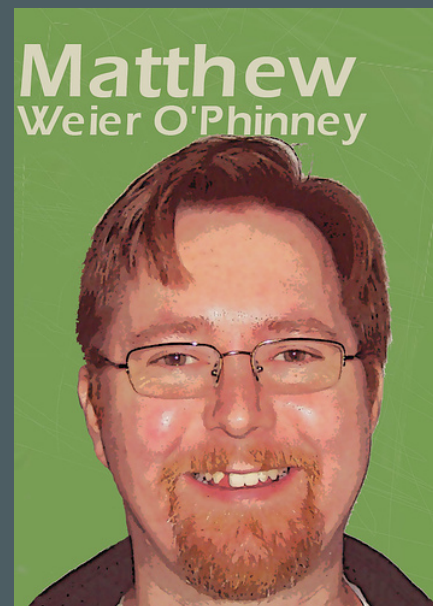


Matthew Weier O'Phinney

- <http://weierophinney.net/matthew>
- <http://framework.zend.com>
- <http://ohloh.net/accounts/weierophinney>



ZendCon '06



ZendCon '07

About You

- Web developer, using PHP
- Is your code well organized and maintainable?
- Are you using source control?
- Is your software documented?
- Do you have automated tests for your software?



Agenda

- Today we will present ideas and techniques that you can use to improve your development process.
 - Not every technique can apply to every project.
 - Start small; implement a few new practices at a time.
 - Find what works for your team and iterate on that.
 - Use this talk as a starting point to go off and learn more.



Agenda

- Source Control
- Coding Standards
- Testing
- Documentation
- Deployment
- Q & A



Source Control



Source Control

- Problems Source Control Solve
 - How do I know if somebody did something?
 - How do others know I did something?
 - How do I get my updates from others?
 - How do I push my updates out to others?
 - Do we have the old version?
 - What changed?



Source Control

- General types of source control:
 - Distributed
 - Non-Distributed



Distributed Source Control

- Methodology
 - Developers work directly on local copies or branches
 - Changes are shared between repositories and/or developers
- Benefits
 - No server necessary
 - Typically very space efficient
 - A git version of a Subversion repository may be 90% smaller
 - Fork a project locally while keeping it synced with the master



Distributed Source Control

- Issues
 - “Master” repository is by convention
 - Which is the canonical version?
 - Harder to automate process based on commits
- Examples
 - Git
 - Developed for, and used by, Linux kernel development
 - Gaining popularity with web developers (c.f. github.com)
 - GNU Arch
 - Developed for tracking kernel development
 - Darcs
 - “Theory of patches”
 - Considered more “pure” implementation, small adoption



Distributed Source Control

- Useful Git commands
 - Create a branch on the fly and switch to it
 - `git branch branchname`
 - Switch to a branch
 - `git checkout branchname`
 - “Cherry-pick” commits to apply from the past hour
 - `git cherry-pick branchname@{1 hour ago}`
 - Create a source tarball of a given tag
 - `git archive --format=tar --prefix projname-TAGNAME/TAGNAME | gzip - > projectname.tar.gz`



Non-Distributed Source Control

- Methodology
 - Developers work on local checkouts or working directories
 - Changesets are checked in- and out- of a central repository
- Benefits
 - Canonical repository
 - Easy to automate processes based on commits



Non-Distributed Source Control

- Issues

- Server is necessary
 - Single point of failure
- Branching is more difficult
- Limited offline functionality

- Examples

- CVS: Concurrent Versions System
- Subversion (SVN): A compelling replacement for CVS



Non-Distributed Source Control

- Typical Workflow:
 - Get an initial checkout of the code
 - Make code changes
 - Commit changes to the repository
 - Update to latest change from repository
 - Repeat



Subversion

- Functions like a superset of CVS
 - Easily move files between directories while preserving history
 - Simplified process of tagging and branching
 - Transactions for when things go wrong
- Extensible and supported by excellent tools
- Popular with many open source projects
- Integrate with other projects with `svn:externals`
- Migrate existing CVS repositories with `cvs2svn`



Subversion Repository Layout

- project/
 - trunk/
 - tags/
 - release-1.0/
 - release-1.1/
 - branches/
 - production/
 - version-1.0/
 - version-1.1/



Subversion Repository Layout

- Use the trunk/ for ongoing development
- Use branches/ for maintained releases
 - Production branch: merge changes from development when stable enough for production
 - Release branches:
 - Merge in security or bug fixes from trunk
 - Create tags when releasing bug fixed versions
- Use tags/ for release/rollout snapshots



Subversion Externals

- Use `svn:externals` to connect remote repositories
 - Seamlessly merge your dependencies
 - Track against anything in the remote repository: trunk, tags, or even a specific revision
 - Pulls code from the remote repository each time you do a checkout or update



Subversion Externals

- `svn propedit svn:externals .`

- **In your editor:**

```
directory [-r##] http://remote-svn-repository/path/
```

- **Example (latest revision of trunk/):**

```
framework http://framework.maintinable.com/svn/  
framework/trunk/
```

- **Example (specific revision of trunk/):**

```
framework -r50 http://framework.maintinable.com/svn/  
framework/trunk/
```



Subversion Hooks

- Allow you to observe and interrupt the commit process
- Implemented as shell scripts on the repository server under the repository's hooks/ directory.
- Hook scripts can be any language (PHP, Ruby, Python, Tcl, shell...) as long as executable & named properly
- Example hooks:
start-commit, pre-commit, post-commit



Subversion Hooks

- Useful Subversion Commit Hooks
 - Pre-Commit:
 - Reject changes that do not pass lint (php -l)
 - Reject changes that violate coding standards (PHP_CodeSniffer)
 - Post-Commit:
 - Send email notification of the change to developers
 - Run unit tests and send email on failure
 - Rebuild DocBook documentation
 - Update tickets on Trac or other issue tracker



Source Control Summary

- Source control systems are necessary
 - As a history of changes to your project
 - To prevent developer change conflicts
- Subversion has many benefits
 - Wide adoption
 - Improved features over CVS
 - Integrate remote repositories with svn:externals
 - Hooks for extending its capabilities
- Distributed source control systems are rapidly gaining popularity and worth a look.



Coding Standards



Why use coding standards?

- Focus on code, not formatting
- Consistency
- Readability
- Collaboration
- Maintenance



What should coding standards provide?

- File, class, variable naming conventions
- Code formatting conventions
- Guidelines for consistency across the code
- Uniformity



Learn from Others

- Don't invent your own standard. All of the issues have already been debated to death by many others.
- Use an established standard
 - Minimize politics by choosing an external standard
 - Choose a standard compatible with the libraries you use
 - Use the standard as a requirement when outsourcing
- Stick to the standard you establish, don't mix



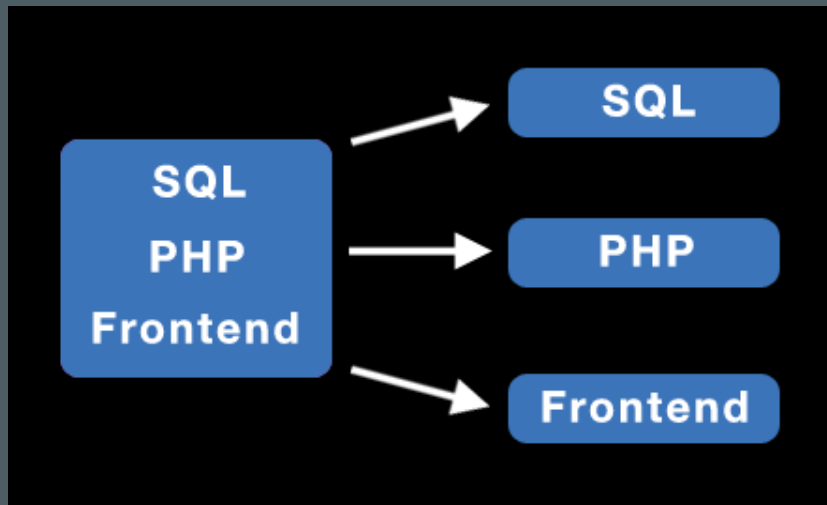
PEAR-like Coding Standards

- Originated with the Horde Project
- Well known, more accepted than any other
- Basis for many open source projects
 - Horde
 - Solar Framework
 - PEAR
 - Zend Framework

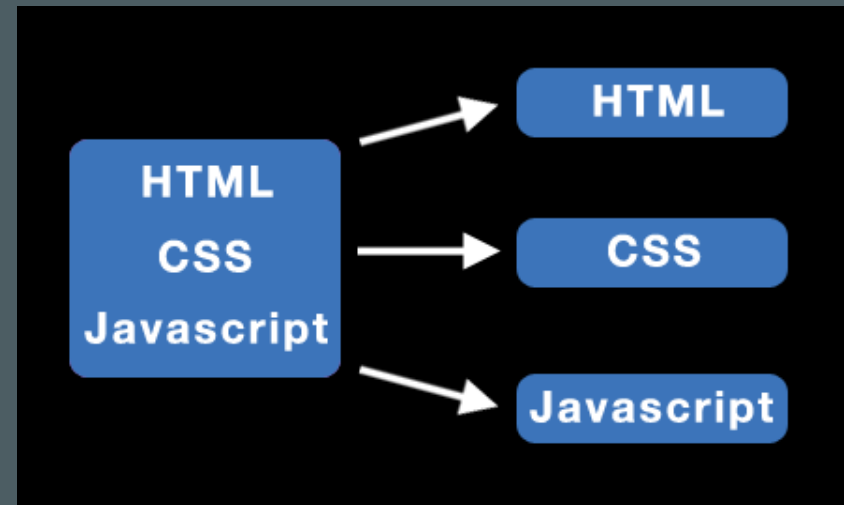


Files and Directories

Server-side Code



Client-side Code



Separate your code cleanly by type and responsibility.

Files and Directories

- Class name used to name file
- .php extension
- Class name underscores convert to directory separator:
 - Spreadsheet_Excel_Writer
 - Spreadsheet/Excel/Writer.php
- One class per file, no loose code



Naming Conventions

- Class names are `MixedCase`
- Method names are `camelCase`
- Constants are `ALL_CAPS`
- Properties and variables are `camelCase`
- Non-public class members are `_underscorePrefixed`



Source Formatting

- One True Brace
 - Functions and Classes have the opening brace on the line following the declaration, at the same indentation level
 - Control Structures keep the opening brace on the same line as the declaration
- Indentation
 - Spaces only; no tabs
 - Four (4) spaces per level of indentation
 - Purpose is consistency of viewing



Aside: Design Patterns

- What are design patterns? Why use them?
 - Reusable ideas, not code
 - Proven solutions to common design problems
 - Better communication through shared vocabulary



Aside: Design Patterns

- I need to notify other objects when an interesting event occurs in the system: Observer
- I need only a single instance of this object to be accessed during this HTTP request: Singleton
- I need to modify the output of an object or change its external interface: Decorator



Example

```
<?php
class Zend_Foo_Bar extends Zend_Foo
{
    const BAZ = 0;

    public $foo;

    private $_bar;

    public function sayHello($name)
    {
        if ($name == 'Matthew Weier O\Phinney') {
            $greeting = "Hello, MWOP!";
        } else {
            $greeting = "Hello, $name!";
        }

        return $greeting;
    }
}
```

- All control structures use braces; no one liners
- Keep lines 75-85 characters in length, maximum
- No shell-style comments (#)



Enforcing Coding Standards

```
% /usr/local/zend/bin/phpcs --standard=Zend --extensions=php .  
  
FILE: /home/matthew/git/paste/application/models/Paste.php  
-----  
FOUND 0 ERROR(S) AND 5 WARNING(S) AFFECTING 5 LINE(S)  
-----  
70 | WARNING | Line exceeds 80 characters; contains 94 characters  
98 | WARNING | Line exceeds 80 characters; contains 92 characters  
99 | WARNING | Line exceeds 80 characters; contains 94 characters  
121 | WARNING | Line exceeds 80 characters; contains 93 characters  
167 | WARNING | Line exceeds 80 characters; contains 94 characters  
-----
```

- Automatically check your code against several common standards or teach it your own standard
- http://pear.php.net/package/PHP_CodeSniffer



Coding Standards Summary

- Adopt a coding standard in your organization
 - Use an existing coding standard that plays well with the libraries that you use
 - Enforce usage of the standard
- Learn and use design patterns as part of your development team's vocabulary



Unit Testing



Unit Testing

- Untested code can be fragile and prone to regression.
- No time to write tests? Start writing tests instead of reloading your browser and doing senseless debugging. Increase your productivity and product quality.
- Start by testing the most critical aspects of your code, strive for testing all of your code. Be practical.
- PHPUnit (<http://phpunit.de>) is one of the most feature-rich and widely-used testing frameworks.



Unit Testing

```
<?php
class Person
{
    private $_name = 'John Doe';

    public function setName($name)
    {
        if (empty($name)) {
            throw new InvalidArgumentException();
        }
        $this->_name = $name;
    }

    public function getName()
    {
        return $this->_name;
    }
}
```

- Class representing a person
- Until named otherwise, the person has a default name.
- The name can be changed.
- The new name cannot be empty.



Unit Testing

```
<?php
class PersonTest extends PHPUnit_Framework_TestCase
{
    public function testNameIsInitiallyJohnDoe()
    {
        $p = new Person();
        $this->assertEquals('John Doe', $p->getName());
    }

    public function testNameCanBeChanged()
    {
        $p = new Person();
        $newName = "Matthew Weier O'Phinney";
        $this->assertNotEquals($newName, $p->getName());
        $p->setName($newName);
        $this->assertEquals($newName, $p->getName());
    }

    public function testNameCannotBeEmpty()
    {
        $p = new Person();
        try {
            $p->setName('');
        } catch (InvalidArgumentException $e) {
            return;
        }
        $this->fail();
    }
}
```

- Each test examines a discrete behavior or “unit” of functionality of the Person object.
- Each test asserts that the behavior of the object meets our expectations.
- If a code change breaks the behavior, the tests will fail and show the regression.



Unit Testing

What else could go wrong here?

```
public function setName($name)
{
    if (empty($name)) {
        throw new InvalidArgumentException();
    }
    $this->_name = $name;
}
```

- Change the method to make it work properly by only accepting valid strings.
- Write a test to assert that its new behavior meets your expectations.



Unit Testing

```
<?php
class PersonTest extends PHPUnit_Framework_TestCase
{
    public function testNameIsInitiallyJohnDoe()
    {
        $p = new Person();
        $this->assertEquals('John Doe', $p->getName());
    }

    public function testNameCanBeChanged()
    {
        $p = new Person();
        $newName = "Matthew Weier O'Phinney";
        $this->assertNotEquals($newName, $p->getName());
        $p->setName($newName);
        $this->assertEquals($newName, $p->getName());
    }

    public function testNameCannotBeEmpty()
    {
        $p = new Person();
        try {
            $p->setName('');
        } catch (InvalidArgumentException $e) {
            return;
        }
        $this->fail();
    }
}
```

```
$ phpunit --testdox PersonTest
```

PHPUnit 3.2.10 by Sebastian Bergmann.

Person

- name is initially john doe
- name can be changed
- name cannot be empty

- Concise documentation that can be understood by a non-technical person
- Only as good as the names of your tests



Test Driven Development

- Write the tests first.
- First make a test that fails because a new behavior does not yet exist. (go red)
- Write the code to make the test pass. (get to green)
- Refactor and repeat.
- Avoid dogma. Find what finds your brain the best. Try to test first or test during. Try not to test too late.



PHPUnit Configuration

```
<phpunit>
  <testsuite name="Pastebin Test Suite">
    <directory>./</directory>
  </testsuite>




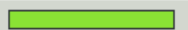
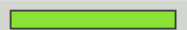













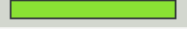



  <php>
    <!-- <ini name="include_path" value="../library"/> -->
  </php>

  <filter>
    <whitelist>
      <directory suffix=".php">../library/</directory>
      <directory suffix=".php">../application/</directory>
      <exclude>
        <directory suffix=".phtml">../application/</directory>
      </exclude>
    </whitelist>
  </filter>

  <logging>
    <log type="coverage-html" target="./log/report" charset="UTF-8"
      yui="true" highlight="true"
      lowUpperBound="50" highLowerBound="80"/>
    <log type="testdox-html" target="./log/testdox.html" />
  </logging>
</phpunit>
```



PHPUnit Code Coverage

	Coverage								
	Classes			Methods			Lines		
Total		100.00%	1 / 1		100.00%	8 / 8		100.00%	88 / 88
Paste		100.00%	1 / 1		100.00%	8 / 8		100.00%	88 / 88
<code>public function add(array \$data)</code>		100.00%	1 / 1		100.00%	12 / 12			
<code>public function get(\$id)</code>		100.00%	1 / 1		100.00%	16 / 16			
<code>public function fetchActive(array \$criteria = NULL)</code>		100.00%	1 / 1		100.00%	10 / 10			
<code>public function fetchActiveCount()</code>		100.00%	1 / 1		100.00%	7 / 7			
<code>public function getForm()</code>		100.00%	1 / 1		100.00%	5 / 5			
<code>public function getTable()</code>		100.00%	1 / 1		100.00%	5 / 5			
<code>protected function _getChildren(\$id)</code>		100.00%	1 / 1		100.00%	6 / 6			
<code>protected function _refineSelection(Zend_Db_Select \$select, array \$criteria)</code>		100.00%	1 / 1		100.00%	27 / 27			

Class-level Analysis



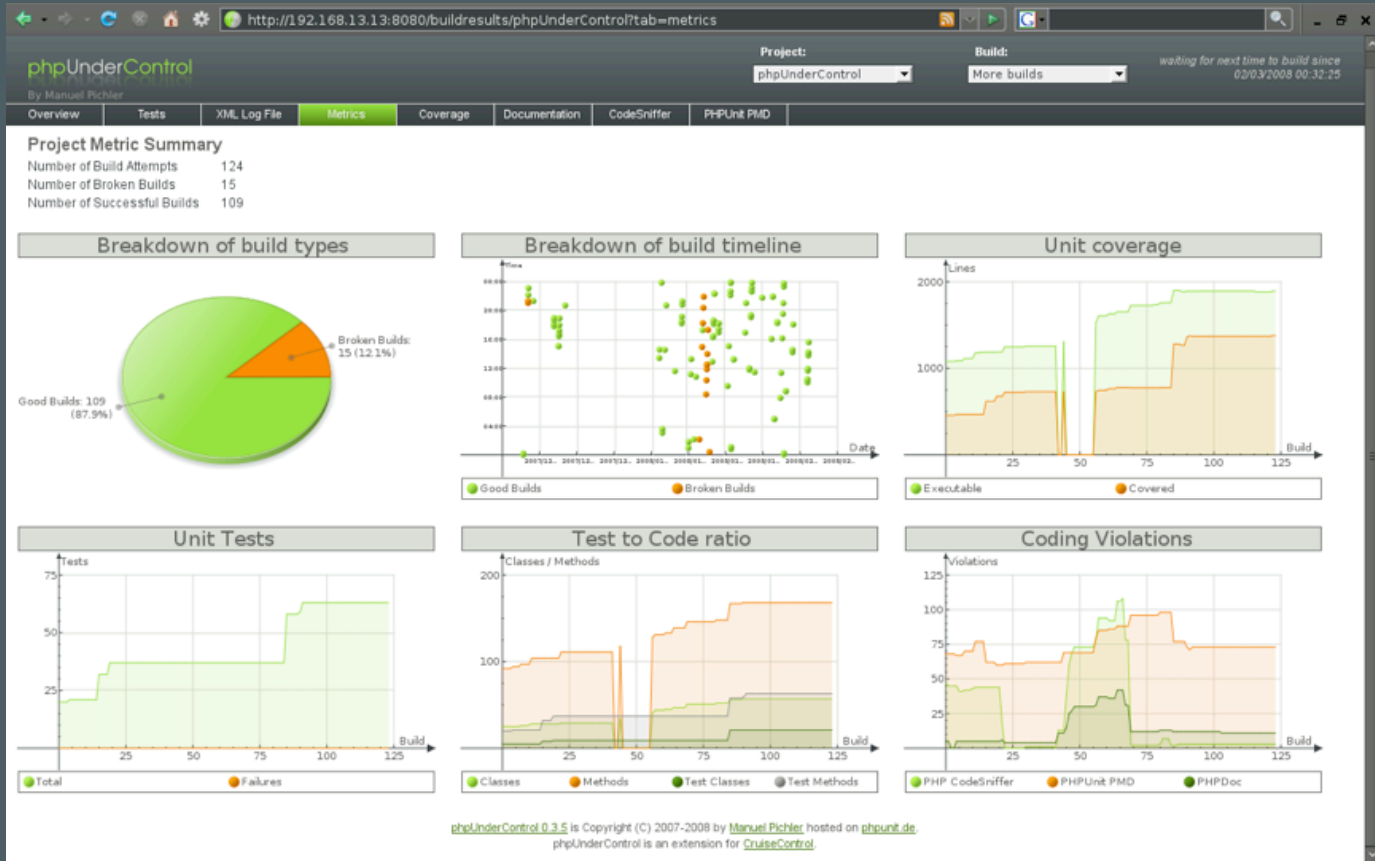
PHPUnit Code Coverage

```
 :      /**
 :      * Add a paste
 :      *
 :      * @param array $data
 :      * @return string
 :      */
 :      public function add(array $data)
 :      {
23 :          $form      = $this->getForm();
 :
23 :          $belongsTo = $form->getElementsBelongTo();
23 :          if (!empty($belongsTo) && array_key_exists($belongsTo, $data)) {
6 :              $data = $data[$belongsTo];
6 :          }
 :
23 :          if (!$form->isValid($data)) {
2 :              return false;
 :          }
 :
21 :          $values = $form->getValues();
21 :          if (!empty($belongsTo)) {
21 :              $values = $values[$belongsTo];
21 :          }
 :
21 :          return $this->getTable()->insert($values);
 :      }
```

Method-level Analysis



Continuous Integration



<http://phpundercontrol.org>



Unit Testing

- Learning to write good object oriented code that is easily testable takes practice and discipline.
- Wrapping your functions in classes is not the same as object oriented design.
- A great deal of PHP code is extremely difficult to test due to poor design. Learn to design for testability.
- Increase your confidence in changes. Your tests will fail if you break something.



Integration and Acceptance Testing



Selenium RC

- Unit testing is often not enough
- Selenium RC is a Browser-based testing tool
 - Launches a web browser
 - Retrieves URL
 - Inspects Results
- PHPUnit integration is simple to use



Selenium RC

- Download and install Selenium server
- Launch Selenium server on command line
- Run PHPUnit tests utilizing Selenium
- Shut down Selenium server



Example Selenium Test

```
class WebTest extends PHPUnit_Extensions_SeleniumTestCase
{
    protected function setUp()
    {
        $this->setBrowser('*firefox');
        $this->setBrowserUrl('http://www.example.com/');
    }

    public function testTitle()
    {
        $this->open('http://www.example.com/');
        $this->assertTitleEquals('Example Web Page');
    }
}
```

- Retrieve a web page and test its contents
- Notice the page can be hosted anywhere
- You can test any web application, PHP or not.



Selenium Assertions

- Fairly rich assertion vocabulary with specific assertions like `assertTitleEquals()`
- General purpose element assertions like `assertElementPresent()` take `$locator`
- Element locators can be a many formats, e.g. XPath.



Selenium Assertions

```
public function testTitle()
{
    $this->open('http://www.example.com/');
    $this->assertElementValueEquals('css=title', 'Example Web Page');
}
```

- Locators can be CSS selectors
- You can use `$locator` with CSS selectors to keep your tests similar to your CSS and JavaScript.



Documentation



Documentation

- Common types of technical documentation:
 - Agile Documentation
 - Test Cases
 - TestDox
 - Source Documentation
 - Doxygen
 - phpDocumentor
 - End User Documentation
 - DocBook



Source Documentation

- phpDocumentor: <http://phpdoc.org>
- Uses annotation tags in source comments very similar to JavaDoc
- phpDocumentor tags are the most used standard for generating documentation from PHP source. They even have their own token assigned to them in the PHP parser itself.



Source Documentation

- Other documentation generators like Doxygen already support phpdoc tags. Don't invent your own tags!
- Supported by a number of different IDEs. Zend Studio is perhaps the most prevalent.



Source Documentation

```
<?php  
  
class Zend_Feed_EntryRss extends Zend_Feed_EntryAbstract  
{  
    protected $_rootElement = 'item';  
}
```

Completely Undocumented



Source Documentation

- Document all source elements
- Files, classes, methods, variables
- Annotate with comments, type hints, and other useful data

```
<?php

/**
 * RSS Feed Entry
 *
 * @uses Zend_Feed_EntryAbstract
 */
class Zend_Feed_EntryRss extends Zend_Feed_EntryAbstract
{
    /**
     * Root XML Element for RSS items
     * @var string
     */
    protected $_rootElement = 'item';
}
```



Source Documentation

```
/**
 * Concrete class for working with Atom entries
 *
 * @category    Zend
 * @package    Zend_Feed
 * @copyright   Copyright (c) 2006-2008 Zend Technologies USA Inc.
 * @license    New BSD License
 */
class Zend_Feed_EntryAtom extends Zend_Feed_EntryAbstract
{
```

- Utilize @category, @package, @subpackage. Documentation systems use these tags to organize the generated documentation.
- Prefix your classes. Easier to browse, prevent top-level name collisions, easier to mix other libraries.



Source Documentation

```
/**
 * Easy access to <link> tags keyed by "rel" attributes
 *
 * If link() is called with no arguments, returns an array containing
 * the values of all <link> tags. If $rel is passed, returns the "href"
 * value of the first <link> tag that has a "rel" attribute matching $rel.
 *
 * @param string The "rel" attribute to match (optional)
 * @return array|string
 */
public function link($rel = null)
{
}
```

- Thoughtful comments, types, throws, etc.
- Actually reflect the source code (comments can lie)



Source Documentation

```
<?php  
  
class Zend_Bar  
{  
    public function sayHello()  
    {}  
}  
  
class Zend_Foo  
{  
    /**  
     * @return Zend_Bar  
     */  
    public static function getBar()  
    {}  
}  
  
$bar = Zend_Foo::getBar();  
$bar->
```



- Some IDEs will parse phpdoc tags to infer information about the source
- Properly document parameters and return values
- Experience for IDE users can be greatly enhanced
- Documentation for other users is also improved



Source Documentation

- Some libraries and frameworks reflect on phpdoc tags for various kinds of automation.
- `Zend_XmlRpc_Server`
 - `@param` to provide and enforce parameter type hints
 - `@return` to provide method signatures
 - Text in the comment for method help



Source Documentation

The screenshot shows a web browser window titled "Generated Documentation" with the "Zend" logo in the top right. A dropdown menu shows "Packages" set to "Zend_Feed". The left sidebar displays a tree view of the "Zend_Feed" package, including sub-packages like "To-do List", "Class trees", and "Interface(s)", and a list of classes such as "Zend_Feed", "Zend_Feed_Abstract", "Zend_Feed_Atom", "Zend_Feed_Element", "Zend_Feed_EntryAbstract", "Zend_Feed_EntryAtom", "Zend_Feed_EntryRss", "Zend_Feed_Exception", and "Zend_Feed_Rss". The main content area is titled "Zend_Feed_EntryRss" and contains the following sections:

- Description:** "Concrete class for working with RSS items." It lists the license as "New BSD License" and the copyright as "Copyright (c) 2006 Zend Technologies USA Inc. (http://www.zend.com)". It notes the class is located in "/Zend/Feed/EntryRss.php (line 34)". A class diagram shows "Zend_Feed_Element" as a base class for "Zend_Feed_EntryAbstract", which in turn is a base class for "Zend_Feed_EntryRss".
- Variable Summary:** Lists a variable "string \$_rootElement".
- Variables:** Lists a variable "string \$_rootElement = 'item' (line 41)" with the description "Root XML element for RSS items." and an access level of "protected".

At the bottom left of the sidebar, it says "Generated by phpDocumentor 1.3.0RC6".

Automatically generated documentation (phpDocumentor)



DocBook: End User Documentation

- DocBook is an XML format that you can use to write end user documentation for your libraries or products
- Powers the php.net manual and a large number of other open source projects
- Used by publishers like O'Reilly and Pragmatic
- Output to a variety of formats: HTML, PDF, CHM (Windows Help), and more.



DocBook: End User Documentation

- Advanced editors are available but not required
- Docbook is a simple format that is relatively easy to learn and use
- Free toolchain runs on *nix or Cygwin
- XML means it can be manipulated by anything that can parse XML, like PHP itself.



DocBook Example

```
<example id="zend.controller.actionhelper.redirector.basicusage.example-4">  
  <title>Using route assembly with gotoRoute()</title>
```

```
<para>
```

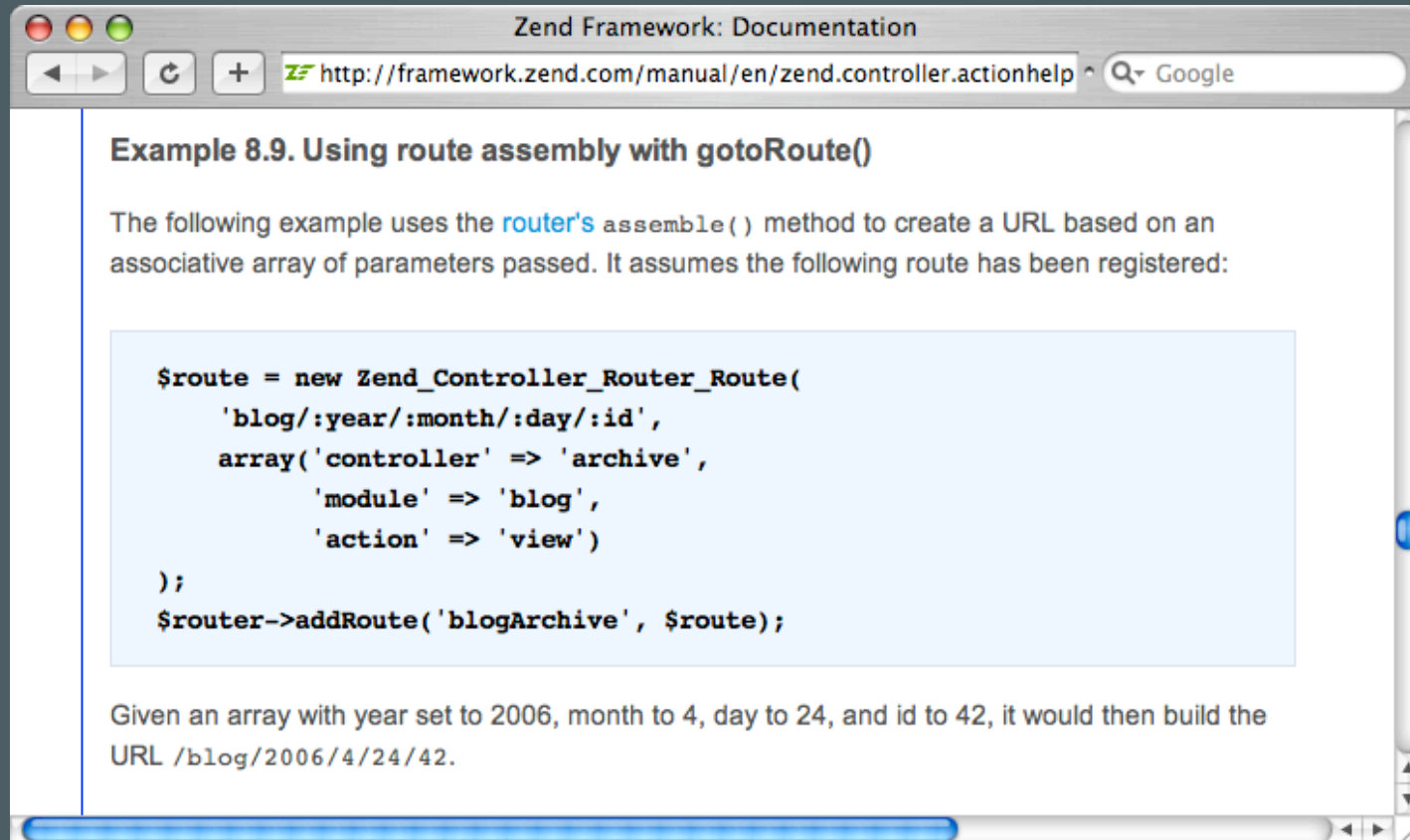
The following example uses the `<link linkend="zend.controller.router">router's</link>` `assemble()` method to create a URL based on an associative array of parameters passed. It assumes the following route has been registered:

```
</para>
```

```
  <programlisting role="php"><![CDATA[  
$route = new Zend_Controller_Router_Route(  
    'blog/:year/:month/:day/:id',  
    array('controller' => 'archive',  
          'module' => 'blog',  
          'action' => 'view')  
);  
$router->addRoute('blogArchive', $route);  
]]>  
  </programlisting>
```



DocBook Example



Example 8.9. Using route assembly with gotoRoute()

The following example uses the [router's assemble\(\)](#) method to create a URL based on an associative array of parameters passed. It assumes the following route has been registered:

```
$route = new Zend_Controller_Router_Route(
    'blog/:year/:month/:day/:id',
    array('controller' => 'archive',
          'module' => 'blog',
          'action' => 'view')
);
$router->addRoute('blogArchive', $route);
```

Given an array with year set to 2006, month to 4, day to 24, and id to 42, it would then build the URL `/blog/2006/4/24/42`.

Documentation Summary

- Write API Documentation
 - phpdoc
 - Document all source elements
 - Write meaningful inline documentation
 - Organize using @category, @package, @subpackage
- Write End User Documentation
 - DocBook
 - HTML output, experiment with others



Deployment



Deployment Tips

- Never edit files on a production server!
- Deploy from repository tags.
- Don't go from Development to Production. Use a Staging environment to mimic Production.
- Establish a formal release procedure.



Deployment Tips

- Instead of overwriting files on the web server, use a symlink. After the new deployment is installed, switch the symlink to point to it. If anything goes wrong, just switch the symlink back to the old version.
- Don't manually interact with the Production server in any way. Write scripts to build and deploy the application without human intervention after starting. Increase repeatability, decrease mistakes.



Deployment Tips

- Write acceptance and integration tests for your application that run on deployment.
- Investigate open source deployment tools to help further automate the process.
- Use server management tools like Monit and Supervisor to keep watch over your deployment.
- Continue to run your tests periodically on a scheduler to detect failures.



Deployment Process Example

- Update QA server from production branch, run tests, get client acceptance
- Tag production branch
- Export from the tag, roll code to the staging server
- Run tests on the staging server as a sanity check
- Deploy to the production server



Questions?



Thanks!



Mike Naberezny
mike@maintainable.com



Matthew Weier O'Phinney
matthew@zend.com

