Best Practices of PHP Development

Matthew Weier O’Phinney
PHP Developer
Zend Technologies

Mike Naberezny
Principal
Maintainable Software
About Us

Matthew Weier O’Phinney

- **PHP Developer, Zend Technologies**
  - Production site maintenance and deployment
  - Internal web services
  - Zend Framework contributor
- **Open Source Contributor**
  - PEAR
  - Cgiapp
  - Solar
About Us

Mike Naberezny

- Coauthored Zend PHP 5 Certification
- Professional software engineer for over ten years at some large companies
- Dynamic language advocate
  - Python Developer (6 years)
  - PHP Developer (about 4 years)
  - Ruby Developer (1.5 years)
- Open Source Contributor
- Principal, Maintainable Software LLC
- Can you read your own code? Can others?
- Is your software documented?
- More importantly, is your software tested?
- Are you using source control?
- Does your team work efficiently?
- Do you push buggy software into production?
Agenda

- **Programming**
  - Coding Standards
  - Documentation
  - Testing

- **Tools and Processes**
  - Collaboration
  - Source Control
  - Deployment

- **Q&A**
- Coding Standards
- Documentation
- Testing
Coding Standards
Coding Standards

- Focus on code, not formatting
- Consistency
- Readability
- Collaboration
Coding Standards

- Don’t invent your own standard. You are not special and your PHP source code is not unique.

- Use an established standard
  - Be objective
  - Minimize politics when choosing
  - Use as requirement when hiring or outsourcing
  - Encourage reuse
  - Compatible with many PHP projects
PEAR Coding Standard

- Popular Library
- Issues have already been debated
- Well known and accepted (more than any other)
- Basis for many open source PHP projects
  - Horde*
  - Solar
  - Zend Framework

* The PEAR coding standard was largely adopted from Horde.
  - Chuck Hagenbuch, Founder of the Horde project
Coding Standards

Naming Conventions

- Class names are CamelCased, with an initial cap, using underscores to separate logical package and code boundaries:
  - Spreadsheet_Excel_Writer
  - Services_Google_AdWords
Coding Standards

Naming Conventions

- **Files**
  - Class name used to name file
  - .php suffix
  - Class name underscores convert to directory separator:
    - Spreadsheet_Excel_Writer
    - Spreadsheet/Excel/Writer.php
  - One class per file, no loose PHP code
Coding Standards

Naming Conventions

- Variable names are camelCased, with the initial character lowercased.
- Constant names are ALL_CAPS with underscores for word separators.
- Private methods and properties are prefixed with an _underscore.
Coding Standards

- **One True Brace**
  - Functions and Classes have the opening brace on the line following the declaration, at the same indent 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
Coding Standards

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

- Reusable ideas, not code
- Proven solutions to common design problems
- Better communication through shared vocabulary
Documentation
Source Documentation
  • phpDocumentor

End User Documentation
  • DocBook
phpDocumentor tags are the most used standard for generating documentation from PHP source code.

Uses annotation tags in source comments very similar to those used by Javadoc.

Other documentation generators, such as Doxygen, support these same tags. Don’t invent your own tags.

Supported by a number of different IDEs. Zend Studio is perhaps the most prevalent.
Source Documentation

Completely Undocumented

```php
<?php

class Zend_Feed_EntryRss extends Zend_Feed_EntryAbstract {
    protected $_RootElement = 'item';
}
```

(is your’s like this?)
Document All Source Elements

- Files, Classes, Methods, Variables, and more
- Comments, Type Hints, other useful metadata
Write Meaningful Documentation

- Thoughtful Comments, Types, Throws, etc.
- Actually reflects source code (comments can lie)
Organize Your Code

• Learn to utilize @category, @package, @subpackage
• PEAR style is the de facto standard
• Always Prefix Your Classes (Foo_)
• http://paul-m-jones.com/organizing-php-projects.pdf
Source Documentation

```php
<?php

class Zend_Bar {
    public function sayHello()
    {
    }
}

class Zend_Foo {
    /**
     * @return Zend_Bar
     */
    public static function getBar()
    {
    }
}

$bar = Zend_Foo::getBar();
$bar->
```

Some IDEs introspect doc tags to infer information about the source.

Properly documenting return types can greatly enhance the experience for many IDE users.
Source Documentation

Automatically generate sophisticated documentation in many formats
DocBook

- Powers the php.net documentation and a large number of other open source projects
- Proven and used by publishers like O’Reilly
- XML-based
- Advanced editors available but not required
- Simple format is easy to learn and use
- Free toolchain runs on *nix or Cygwin
Testing
- Unit Testing
- Test Driven Development
If there is any single “best practice” that PHP developers should learn, testing is it.*

* Along with learning to write object oriented code that has some hope of being maintained.
Unit Testing

- Unfortunately, huge amounts of PHP code is procedural spaghetti, not object oriented, let alone tested.

- Code without tests is fragile and will regress.

- No time to write tests? Start writing tests instead of reloading your browser and doing senseless debugging. Increase your productivity and product quality.

- `print()` and `var_dump()` are not testing tools.
Unit Testing

Class representing a person

Until named otherwise, the person has a default name.

The name can be changed.

The new name cannot be empty.

```php
<?php

class Person
{
    private $_name = 'John Doe';

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

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

Testing the Person object

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.

```php
<?php

class PersonTest extends PHPUnit_Framework_TestCase {
    public function testNameIsInitiallyAnonymous() {
        $p = new Person();
        $this->assertEqual('John Doe', $p->getName());
    }

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

    public function testNameCannotBeEmpty() {
        $p = new Person();
        try {
            $p->setName('');
        } catch (InvalidArgumentException $e) {
            return;
        }
        $this->fail();
    }
}
```
What else could go wrong here?

```php
public function setName($name)
{
    if (empty($name)) {
        throw new IllegalArgumentException();
    }
    $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

- Learning to write good object oriented code that is testable takes practice and discipline.

- Using Classes != Object Oriented Design

- A great deal of PHP code is extremely difficult to test due to poor design. Learn how to design for testability.

- No longer fear changing code because your tests will fail if you break something.

- Stop reloading your browser.
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 to keep your code clean and DRY.

- Repeat.

- Please learn more about testing. Start here: http://www.phpunit.de/pocket_guide/
Tools & Processes

- Collaboration
- Source Control
- Deployment
Collaboration
Overview

Working with a geographically separated team is increasingly common and requires the same open communication channels as working in the same office.

- Messaging
- Web Collaboration
- Trac
Collaboration

Messaging
Collaboration: Messaging

Technologies

- Email
- Instant Messaging
- VOIP
- Face-to-Face (old technologies are best)
Email: When to use it

- Documenting and communicating decisions (be careful)
- Distribution lists
- Examples and use cases
- Review of code implementations
- Collaborating on specifications
Email: When not to use it

- Time critical tasks: “I need this now!”
- Quick questions: “Can you...?” “Where is...?”
- Keep in mind spam filters; messages get lost
IM: When to use it

- Quick questions: “Can you ...?” “Where is...?”
- Time critical tasks (e.g., deploying code or servers)
- Quick code snippet review: “Will this work?”
- Multi-way conversations in real-time
IM: When not to use it

- Decision making (drive by decisions)
- Anything important that should be documented
- Long conversations
VOIP: Why?

- Sometimes hearing something leaves a different impression than reading it

- Meetings

- Get to know people by spoken word (and possibly visual, if the VOIP solution has integrated video)
Collaboration: Messaging

VOIP: When to use it

- Meetings
- Decision making
- Time critical tasks (e.g., deploying code or servers)
VOIP: When not to use it

- Discussing code implementation details
  “Then take dollar-var and push it through fooAction; use the return value to append to dollar-underscore-bar.”

- Quick questions
Collaboration: Messaging

Face-to-Face

- Meet in person as often as time and budget allows
- Builds camaraderie
- Easier to understand written word when you can hear the voice behind it
Collaboration: Messaging

Summary

- Communicate often
- Communicate in a variety of media
- Be polite
- Provide context
- Messaging can be distracting; build ‘offline’ time into each day
Web Collaboration
Collaboration: Web Collaboration

Technologies

- Wikis
- Google Docs & Spreadsheets
- pastebin.com, paste2.org
- Thick-client technologies
Collaboration: Web Collaboration

Wikis

- Central documentation source; best place to record decisions and processes
- Easy markup
- Plugins often provide extra functionality
Collaboration: Web Collaboration

Google Docs & Spreadsheets

- Writely and Spreadsheets
- Invite-only for viewing and editing; control who sees what, and who can edit it
- Real-time updates
- Who owns the data? How long will it be available?
Source Control
Problems Solved

- How do I know if somebody did something? How do they know I did something?

- How do I get updates from others? How do I push my updates out to them?

- Do we have the old version? What changed?
Source Control

Distributed

- **Methodology**
  - Developers work directly on local repositories
  - Changesets are shared between repositories

- **Examples**
  - GNU Arch: Developed for Tracking Kernel Development
  - Darcs: “Theory of Patches”
  - Git: Linux Kernel Development
Non-Distributed

- **Methodology**
  - Developers work on local checkouts
  - Changesets checked in/out of a central repository

- **Examples**
  - CVS, Concurrent Versions System
  - Subversion: A compelling replacement for CVS
Source Control

Workflow

- Create repository
- Perform local checkout
- Write code
- Record changes
- Check changes in to repository
- Check for repository updates
- Lather, rinse, repeat
Advantages

- Central repository makes administration and control easier

- Central repository lends itself to automated processes (e.g., commit notifications, documentation builds, etc.)
Disadvantages

- Harder to move between servers reliably

- Author verification left to OS; no signed revisions
  - Note: Subversion’s pre-commit hooks allow greater flexibility in this regard
Source Control: Subversion

Subversion

- A compelling replacement for CVS
- Functions like a superset of CVS
- Migrate existing CVS repositories to SVN
- Popular with many open source projects
- Easily move files between directories while preserving histories
- Simplified process of tagging and branching
- Transactions for when things go wrong
- Extensible and supported by excellent tools
Source Control: Trac

Trac

http://trac.edgewall.com/
Source Control: Trac

- Simple Installation
- Repository Browser
- Wiki
- Issue Tracker
- Roadmap / Milestones
- Plugins
- Great Collaboration Tool
Link Changesets and Tickets

Changeset linking to ticket

Changeset 3342

Timestamp: 10/25/06 15:31:37
Author: daniel.b

Message:

- Add mapping for Studio Beta

Related to ticket:144

Ticket comment linking to changeset

10/27/06 00:34:27: Modified by luke

- **status** changed from new to closed.
- **resolution** set to fixed.

Implemented in [3363] and merged into production with [3364].
Timeline

09:30:06:

13:02: Changeset [3116] by matthew
   * Added response fault codes ...

09:29:06:

23:44: Changeset [3115] by lucas
   * New utility that decorates the document with calls to a controller ...

23:42: Changeset [3114] by lucas
   * Test template

23:39: Changeset [3113] by lucas
   * Basic structure for ModuleController style module

20:00: Changeset [3112] by matthew
   Related to #62: ...

02:42: Changeset [3111] by matthew
   $mrio/server changes related to #62: ...

02:27: Changeset [3110] by matthew
   All changes related to #62: * Modified Zend_Server_Interface ...

09:28:06:

21:29: Changeset [3109] by lucas

20:00: Changeset [3108] by lucas
   Branching myzend development

19:41: Ticket #59 (defect) closed by luke
# Reports

## Available Reports

This is a list of reports available.

<table>
<thead>
<tr>
<th>Report</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>Active Tickets</td>
</tr>
<tr>
<td>{2}</td>
<td>Active Tickets by Version</td>
</tr>
<tr>
<td>{3}</td>
<td>All Tickets by Milestone</td>
</tr>
<tr>
<td>{4}</td>
<td>Assigned, Active Tickets by Owner</td>
</tr>
<tr>
<td>{5}</td>
<td>Assigned, Active Tickets by Owner (Full Description)</td>
</tr>
<tr>
<td>{6}</td>
<td>All Tickets By Milestone (Including closed)</td>
</tr>
<tr>
<td>{7}</td>
<td>My Tickets</td>
</tr>
<tr>
<td>{8}</td>
<td>Active Tickets, Mine first</td>
</tr>
<tr>
<td>{9}</td>
<td>Current Week Report</td>
</tr>
<tr>
<td>{10}</td>
<td>Last Week Report</td>
</tr>
</tbody>
</table>

[Create new report]
Roadmap / Milestones

- /trac/roadmap
- Create “projects” or goals
- Assign deadlines
- Attach tickets by milestone
- View progress as tickets are opened and closed against the milestone
Email2Trac

- [http://trac-hacks.org/wiki/EmailToTracScript](http://trac-hacks.org/wiki/EmailToTracScript)
- Integrates with local MTA and Trac install
- Send email to ticket address to create new tickets
- Reply to Trac-generated issue emails, and comments to the issue will be created
- Email attachments are attached to the issue
Source Control: Trac: Tips

Tags

- http://muness.textdriven.com/trac/wiki/tags
- Tag wiki entries, issues, changesets for easy searching and categorization
- Create tag clouds
- List items by tag
Deployment
Deployment

- Never edit files on a production server!
- Deploy from repository tags.
- Don’t go from Development to Production. Use a Staging server to mimic the Production environment.
- Establish a Deployment Release Procedure (DRP).
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 it back.

Don’t manually interact with the Production server in any way. Write scripts to build and deploy the application without any human intervention after starting.
Deployment

- Write acceptance and integration tests for your application that run on deployment.

- Investigate open source deployment tools like Capistrano to help further automate the process.

- Use server management tools like Supervisord to continuously monitor your deployment.

- Continue to run your tests periodically on a scheduler to detect failures.
Questions?
Thanks!

Matthew Weier O’Phinney
PHP Developer
Zend Technologies
matthew@zend.com
http://weierophinney.net/matthew

Mike Naberezny
Principal
Maintainable Software
mike@naberezny.com
http://mikenaberezny.com