Welcome

The Seven Steps To Better PHP Code

(Part One)

Stefan Priebsch, e-novative GmbH

Who I Am

- PHP enthusiast since 2000
- IT and PHP Consultant
- From Munich, Germany
- University Degree in Computer Science
- Writer (Books and Articles)
- Blog: *http://inside.e-novative.de*
- Email: *stefan.priebsch@e-novative.de*

What is Refactoring?

- Changing code
  - to improve readability
  - to simplify the structure
- *Without* changing the results
- *Not* adding new functionality

- 'Cleaning Up' the code

What is Refactoring?

- important part of agile methods
- not only usable for object-oriented code
- an interesting experience
- **and fun to do!**

Resources

- *The* definitive book on Refactoring:
  Martin Fowler: *Refactoring. Improving the Design of Existing Code*
- www.refactoring.com
- http://c2.com/cgi/wiki?WhatIsRefactoring
- How to write unmaintainable code
  http://mindprod.com/jgloss/unmain.html

Why Refactor?

- The world constantly changes
  - environment changes
  - terminology changes
  - requirements change
- Software design decays over time
- We *never* get things right the first time

When should you Refactor?

- Before you add new functionality
  - Refactor until it becomes obvious how to add a feature
- When you don't understand the code
  - Make code more readable
- When you fix a bug
- After a code review

Refactoring Benefits

- Improves software design
- Less bugs, less debugging
- Speed up development
- More code reuse

How to Refactor

- Run tests to make sure things work
- Refactor, one step at a time
- Rerun tests to make sure things still work

- Automated Refactoring support starts showing up in PHP IDEs
  - Automation does not replace thinking

Testing

- Prove that software works to specification
- Ensure code still works after a change
- Test frequently
- Use self-testing, automated tests

Testing

- xUnit Framework for PHP
  - PHPUnit (www.phpunit.de)
  - SimpleTest (www.simpletest.org)
- Not only for object-oriented code
- If you don't have tests, create them as you go

- At least, run a Lint check:
  php -l <filename>

Step 1: Format Source Code

- Code is read more often than written
- Code should be human-readable
- It's easy to code for computers,
  but hard to code for humans

Unreadable C Code

```
#include <stdio.h>
#define y
for(c=0,i=54;i>=0;c=(s[i]+=a[i]+c)/11,s[i]=
s[i]%11,i--);
#define z for(i=0
main(){char s[55],a[55];int
c,i,d;for(;;){z;i<55;s[i++]=0);for(;(d=
getchar())!='\n';){d=d>58?(d>77?d-87:d-
55):d-48;z;i<55;a[i]=s[i],i++);y
z;i<54;s[i]=s[i+1],i++);s[54]=0;y
z;i<53;a[i++]=0);a[53]=d/11;a[54]=d%11;y}z;i
<54&&s[i]==0;i++);for(;i<55;i++)printf("%c",
s[i]<10?s[i]+48:32);printf("\n");}}
```

Unreadable PHP Code

```
$bool = ($i<7&&$j>8||$k==4);


      if ($a == $b)
    {
  if ($b > 4)
  {
    do_something(); }
  } else {
    do_something_else();
    }
```

Step 1: Format Source Code

- Format source code consistently
- Establish a coding standard
  - How to indent
  - Where to put whitespace
  - Capitalization
- Reuse existing coding standards
- Use a code beautifer

PHP_Beautifier

- Automatically formats PHP source code
- PEAR package
  http://pear.php.net/package/PHP_Beautifier
- Installation:
  pear install --alldeps --force PHP_Beautifier
- Usage:
  php_beautifier ugly.php pretty.php

PHP_Beautifier

```
function test ($aParameter)
{
   $a = 4;
     $b=5;

     if ($a == $aParameter)
   {
      var_dump('Hello world'); }

    }
```

PHP_Beautifier

```
function test($aParameter) {
    $a = 4;
    $b = 5;
    if ($a == $aParameter) {
        var_dump('Hello world');
    }
}
```

PHP_CodeSniffer

- Detects coding style violations
- PEAR package
  http://pear.php.net/package/PHP_CodeSniffer
- Installation:
  pear install PHP_CodeSniffer
- Usage:
  phpcs <directory>

PHP_CodeSniffer

```
1   | ERROR   | End of line character is invalid; expected "\n"
                but found "\r\n"
2   | ERROR   | You must use "/**" style comments for a file
                comment
20  | ERROR   | Expected "if (...) {\n"; found "if (...)\n{\n"
22  | ERROR   | Line indented incorrectly; expected at least 4
                spaces, found 3
27  | WARNING | Line exceeds 85 characters; contains 147
                characters
38  | ERROR   | Missing class doc comment
77  | ERROR   | Expected "if (...) {\n"; found
                "if (...)\n      {\n"
91  | WARNING | Equals sign not aligned with surrounding
                assignments; expected 4 spaces but found 1 space
93  | ERROR   | Space after opening parenthesis of function call
                prohibited
95  | ERROR   | Expected "} elseif (...) {\n";
                found "}\n    elseif (...) {\n"
97  | ERROR   | Space found before comma in function call
```

Step 2: Naming Conventions

- Establish a naming scheme
  - Capitalization
  - Consistent names
  - Make the purpose clear
- Keep file names simple
  (alphanumeric chars, blanks, underscores)
- Prefix names in global namespace

Step 2: Naming Conventions

```
processData()

doIt()

innerLoop()

$del_pos

checkConnection() vs. isConnected()

$filename vs. $fileName

open_connection($a, $b, $c, $d);

ftp_connect($server, $port, ...);
```

Step 2: Naming Conventions

- Constants: capital letters, with prefix

```
define('MY_CONSTANT', 23);
var_dump(MY_CONSTANT);
```

Step 2: Naming Conventions

- Global Variables: CamelCase, with prefix (upper case first letter)

```
protected $_createTimestamp;
public $defaultFormat;
private function _listFiles() {}
$myGlobalVariable = 3;
$myOtherGlobal = 'password';
protected function _activateBar()
{}

public function setMember($aValue)
{}
```

Step 2: Naming Conventions

- Functions: Lower case, underscore-separated, with prefix
- Parameters: CamelCase, prefixed by 'a' or 'the' (upper case first letter)

```
function my_foo($aBar, $aBaz)
{
}
```

Step 2: Naming Conventions

- Class Names: Camel Case, with prefix (upper case first letter)

```
class myFoo
{
}
```

Step 2: Naming Conventions

- Class Constants don't need a prefix

```
class myFoo
{
  const CONSTANT = 42;
}
```

Step 2: Naming Conventions

- Member: CamelCase, lower case first letter
- non-public members prefixed by underscore

```
class myFoo
{
  private $_fooBar;
  protected $_createTimestamp;
  public $defaultFormat;
}
```

Step 2: Naming Conventions

- Methods: CamelCase, lower case first letter
- non-public methods prefixed by underscore

```
class myFoo
{
  private function _listFiles()
  protected function _activateBar()
  public function isActive()
}
```

Step 2: Naming Conventions

- Local Variables: underscore-separated, lower case, no prefix

```
function my_function($aBar, $aBaz)
{
  $local_variable = trim($aBar);
}
```

Step 2: Naming Conventions

```
public function setMember($aValue)
public function getMember()


public function isActive()
public function hasBar()
public function importBar()
public function createReport()
```

Step 3: API Documentation

- Document public interfaces
- Code assumptions instead of commenting
- Are inline comments necessary?
  - Code should communicate its purpose
  - The better the names, the fewer comments
- PHPDoc, derived from JavaDoc

PHPDocumentor

- Create API documentation from PHP source
- PEAR package
  http://www.phpdoc.org
- Installation:
  pear install --force --alldeps PHPDocumentor
- Usage:
  phpdoc -d <directory> -t apidoc

DocBlocks

- Special Comment Blocks

```
/**
 *  ...
 */
```

- Special tags

```
@author, @version, @param, @returns
```

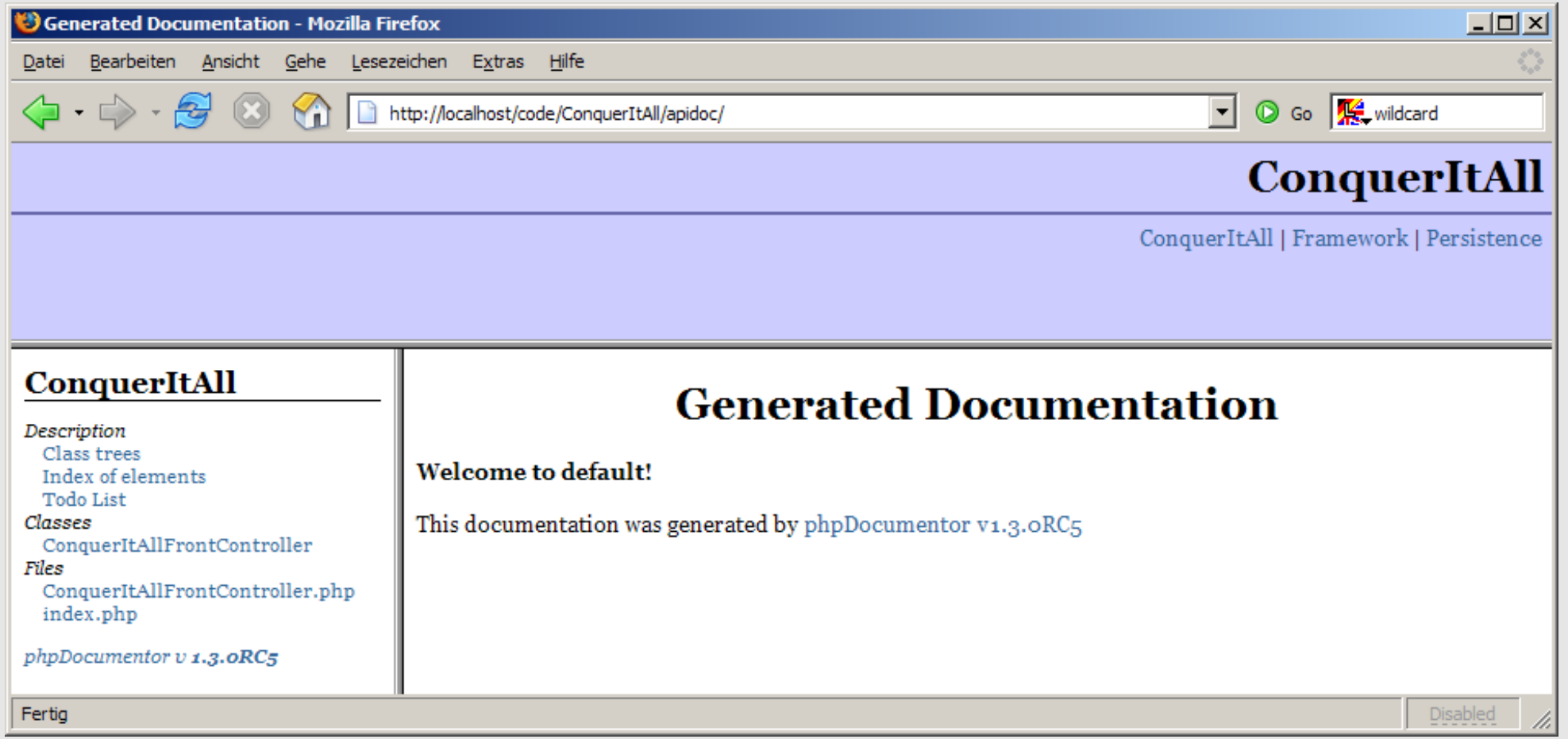
DocBlocks

```
/**
 * This is what the class does.
 *
 * @package     PackageName
 * @author      Your Name
 * @version     1.0.0RC2
 */
class myFooClass
```

DocBlocks

```
/**
 * Method description
 *
 * @param  mixed  $key
 * @param  array  $array
 * @return string The value of that key
 */
public function foo($key, $array)
```

Step 3: API Documentation



Step 3: API Documentation

Class ConquerItAll_Server

Description

Description | Vars (details) | Methods (details)

Der ConquerItAll-Server

Der ConquerItAll-Server nimmt die Anfragen über HTTP vom Client entgegen und verarbeitet diese. Die Antworten werden als XML-Datei zurückgegeben.

Located in ***/ConquerItAll/Server.php*** *(line* ***21****)*

Variable Summary

Description | Vars (details) | Methods (details)

string **$clientName**
string **$clientUrl**
integer **$protocolVersion**
string **$sharedSecret**

Method Summary

Description | Vars (details) | Methods (details)

ConquerItAll_Server **__construct** ([*string* **$aSecret** = ''])
string **getClientName** ()
string **getClientSecret** ()
string **processRequest** (*string* **$aRequest**)

Step 3: API Documentation

Methods

Description | Vars (details) Methods (*details*)

Constructor __construct (line 56)

Konstruktor

- **access:** public

ConquerItAll_Server **__construct** ([*string* **$aSecret** = ''])

- *string* ***$aSecret****: Shared Secret*

getClientName (line 76)

Gibt den Client-Namen zurück

- **return:** Client-Name
- **access:** public

string **getClientName** ()

Step 3: API Documentation

Class Framework_Exception_FieldRequired

Description

Description | Vars | Methods

Implements interfaces:

- Framework_Message

Located in ***/Framework/Exception/FieldRequired.php*** *(line* ***12****)*

```
Exception
    |
    --Framework_Exception_Validation
        |
        --Framework_Exception_FieldRequired
```

Step 3: API Documentation

$parameters

Framework_RequestHelper::$parameters in RequestHelper.php

Request parameters

r top

$request

Framework_Command::$request in Command.php

$requestHelper

Framework_FrontController::$requestHelper in FrontController.php

RequestHelper.php

RequestHelper.php in RequestHelper.php

s top

setDefault

Framework_CommandMap::setDefault() in CommandMap.php

setDefaultCommand

Framework_FrontController::setDefaultCommand() in FrontController.php

setForward

Framework_Command::setForward() in Command.php

setLanguage

Framework_FrontController::setLanguage() in FrontController.php

Step 3: API Documentation

Todo List

ConquerItAll

ConquerItAll_Command_Confirm::doExecute()

- Bestätigung abhängig von Konfigurationseinstellung nicht anfragen

ConquerItAll_Admin_View_Configuration::getPageContent()

- Layout verbessern

ConquerItAll_Admin_FrontController::initApplication()

- Konfigurationseinstellungen in Session zwischenspeichern

ConquerItAll_Command::initialize()

- Schlüssel aus Konfigurationsdatei holen

ConquerItAll_Command_SignOn::initialize()

- Effizienteren XML-Parser verwenden

ConquerItAll_Application::setName()

- Standardwert vergeben

Thank you. Questions?

- Presentation Slides will be made available at http://www.zend.com/resources/webinars
- Join the second part of this presentation on January 23rd, 2008
- Email me at stefan.priebsch@e-novative.de
- Have a Nice Day ...
  ... and try some Refactoring!