

php | tek - Chicago, May 16-18, 2007

Writing Maintainable PHP Code

Organizing for Change

Jeff Moore

<http://www.procata.com/>

Jeff Moore

- * First programming job in 1987
- * 9 years working on the same ERP system
- * PHP Since 2000

**What is
Maintainable Code?**

Maintainable Code is...

- * Quick to change
- * Safe to change

Maintainability is About Change

- * “The system doesn’t handle this case”
- * “This is too hard to do”
- * A new law goes into effect
- * “Good news, we’ve been bought out”

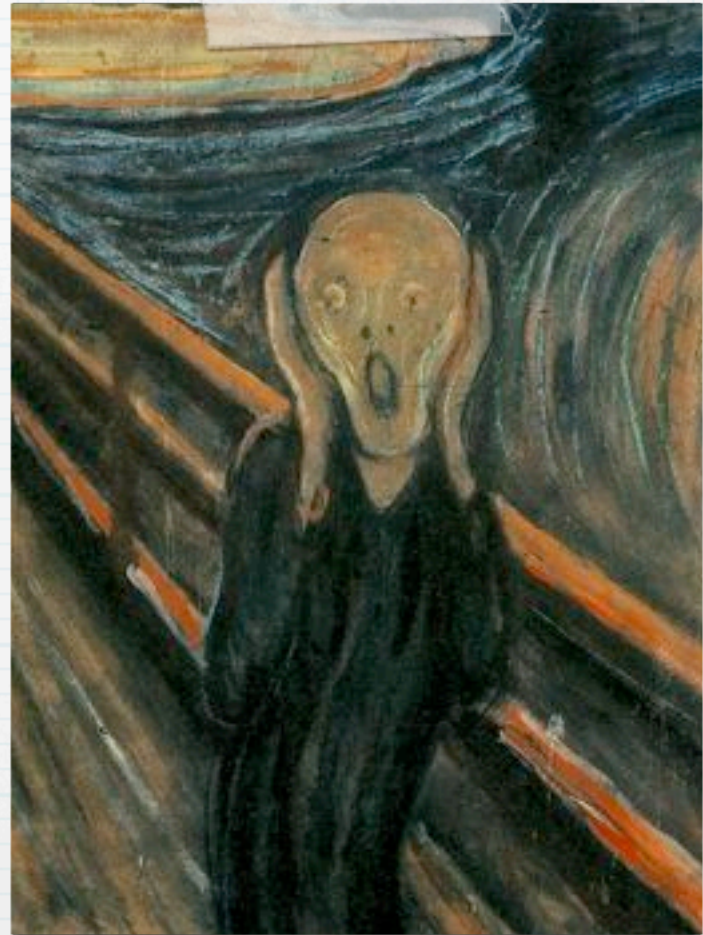
How Do You React To Change?

- * Excitement!
- * Great idea...
- * We can do that...



How Do You React To Change?

- * Fear
 - * That won't work because...
 - * We'll do that someday...



Much of Programming is Maintenance

- * Successful small systems grow into large systems
- * Systems last longer than you expect

Organizing For Change

How Do You Organize Your Stuff?

- * Put tools and materials for a specific task in one place
- * Eliminate distractions
- * Have different places for different tasks

Mise en Place

- * Everything in place
- * Clean up as you go
- * Don't leave a mess

Refactoring

- * Improving the design of existing software without changing its behavior
- * Cleaning up code as we go
- * Book by Martin Fowler

Code Smells

- * Signs that you need to refactor

Smell: Duplicate Code

- * When the same code exists in more than one place changes have to be made in more than one place
- * Its easy to miss a change
- * It takes time to analyze variation

Smell: Shotgun Surgery

- * When a single change requires visiting many places in the code

The Rule Of Threes

- * First time just do it
- * Second time, wince and ignore it
- * Third time, eliminate the duplication

Put Things that are
Alike Together

Grouping PHP Code

- * Application

- * Package

- * File

- * Namespace

- * Class

- * Function

- * Braces

- * Whitespace

Cohesion

- * How closely is the code in a group related?

Poor Cohesion

```
function import($valueList, $append = FALSE) {  
    if ($append) {  
        $this->vars += $valueList;  
    } else {  
        $this->vars = $valueList;  
    }  
}  
  
$obj->import($values, TRUE);
```

Smell: Boolean Parameters

- * Indicates lack of cohesion
- * Hard to understand in calling context

Better

```
define('REPLACE', 0);
define('APPEND', 1);

function import($valueList, $mode = REPLACE) {
    if ($mode == APPEND) {
        $this->vars += $valueList;
    } else {
        $this->vars = $valueList;
    }
}

$obj->import($values, APPEND);
```

Smell: Type Code Parameters

- * Can indicate lack of cohesion

Best

```
function append($valueList) {  
    $this->vars += $valueList;  
}
```

```
function replace($valueList) {  
    $this->vars = $valueList;  
}
```

```
$obj->append($values);
```

Good Cohesion

- * **Communicational**

- * Lines operate on the same data

- * **Sequential**

- * Output from one line is input to the next

- * **Functional**

- * All lines contribute to the same task

Smell: Divergent Change

- * When the same code changes for different reasons
- * Indicates poor cohesion

Keep Things That Are
Different Apart

Unwanted Interactions

- * A change in one place causes an unwanted effect in another
- * Requires time researching and doing due diligence before making a change
- * Risk may prevent change entirely

Dependencies

- * Defining dependencies in terms of change:
- * Software element A depends upon B if
“You can postulate some change to B that would require both A and B to be changed together in order to preserve overall correctness.” - Meiler Page-Jones

Coupling

- * The degree to which a software group is related to other parts of the program

Tools for Limiting Coupling in PHP

Variable Scopes

- * **Application (Global)**

- * `$_GLOBALS['var'];`

- * **Class (Object)**

- * `$this->var;`

- * `self::$var;`

- * **Function (Local)**

- * `$var;`

Visibility

- * Private
- * Protected
- * Public

Encapsulation

- * Limiting interactions
- * Information hiding
- * Strong encapsulation groups have names
- * May have a variable scope
- * May specify element visibility

What Happens When
We Don't
Encapsulate?

.ini Options

- * 146 .ini options in my PHP
- * Invisible parameters to every function and class
- * Equivalent to global variables

.ini Strategies

- * Set them up front
 - * Easy
 - * Documents your choices
- * Set and restore
 - * Portable
 - * Easy to miss something

Global Keyword is Evil

- * `global $var;`
- * Destroys the distinction between global and local variables
- * Global variables should be hard to use
- * `$_GLOBALS['var']`

Main Line Code

- * More opportunities for interaction
- * Avoid main line code

Keep Things That Are
Different Apart

Separation of Concerns

- * Put things that change for different reasons in different places
- * Templating
- * MVC

Anticipating Changes

- * Separation of concerns requires anticipating the likely changes
- * MVC allows the view to change independently of the other layers
- * MVC also requires some changes to be made in 3 places instead of 1
- * Balance tradeoffs

Testing and Maintainability

Testing and Maintainability

- * Unmaintainable code is hard to test
- * Maintainable code is easier to test
- * To write maintainable code, write testable code

**What Makes Code
Hard to Test?**

Cyclomatic Complexity

- * The number of decision points in the code plus one
- * Count each if, while, each case in a switch
- * A.k.a Conditional Complexity
- * Software metric for measuring maintainability?

Conditional Complexity

```
if ($condition) {  
    $x = getA();  
} else {  
    $x = getB();  
}  
foreach ($x => $message) {  
    echo $message;  
}
```



3

Smell: Long Method

- * Long methods are hard to maintain
- * How long is too long?
- * Conditional complexity > 10 is too long

Conditional Complexity and Testing

- * Estimate of the number of test cases you will need to write
- * The maximum number of paths through the code to achieve branch coverage

Coverage

- * Line Coverage

- * Has every line in the program been executed?

- * Branch Coverage

- * Has both the true and false branch of every decision in the program been taken?

**Why Are Fewer
Conditionals Easier
to Maintain?**

**Avoid Conditional
Code?**

Smell: Switch Statements



3

```
switch ($shape) {  
  'circle':  
    $area = PI * $radius * $radius;  
    break;  
  'rectangle':  
    $area = $width * $height;  
    break;  
}
```

With Polymorphism

```
* class Circle {  
    function area() {  
        return PI * $this->radius ** 2;  
    }  
}
```



```
* class Rectangle {  
    function area() {  
        return $this->width * $this->height;  
    }  
}
```



Testing Makes Code Safe to Change

- * Having a full test suite reduces fear of change
- * Tests increase confidence in the correctness of the change
- * Tests increase the speed of change

Consistency

Coding Standards

- * Standardize the stuff that doesn't matter
- * Reveals the patterns in the code that do matter
- * Use the PEAR coding standard

Refactored Normalize Form

- * Refactoring book becoming a standard
- * Refactoring code smells away leads to consistent code
- * Refactor to standard patterns
- * Reveals new patterns in code that matter

Go Write Good Code

Questions?