

Worst PHP Practice

Marcus Börger Johannes Schlüter



PHP Quebec 09: http://talks.somabo.de/200903c.pdf .pps



Topics

- **b** Security
- **b** Overdesign
- Spagetthi code
- **b** DIY Do It Yourself
- **b** Utilize available Tools
- Micro Optimizations
- **b** References
- **b** Do everything with Objects
- b Include vs. Require vs _once
- **p** Provide a Style Guide
- **b** Use with Caution





Security

M

M

M

Adress security once the application is ready No hacker will ever care for my application I do not have security issues

Since hackers automaticaly scan, they will find you

J

Take care of security right from the beginning

- J Security should and will influence:
 - J Your overall design
 - J Your development and deployment process









Overdesign

- M Always plan for everything
- Limit yourself to what you and your customer want
- Do not fear restarting development
- J The more complex your design gets:
 - J The more complex your code gets
 - J The more bugs you have
 - J The more the development will cost
 - J The more likely you are to miserably fail
 - PHP is not: Java, C++, Python, Ruby on Rails





Spaghetti code

M This code just needs a little bit more tweaking

Modularize / Componentize your code

Every day code can put in base repository

Not everything you use twice belongs there





DIY - Do It Yourself

M

Implementing everything yourself

M Waste of time

MDevelopment

MTesting

MDocumenting

MMaintenance

M Creating unnecessary bugs

J

Prefer NIH

J Existing code should be

J Well developed

J Tested

J Documented

J Maintained



J Have very few bugs if at all



Utilize available Tools

M

Designing, Testing, Versioning, Documenting . . .

... That all takes far too much time!

- Software design lets you capture errors early
- I Testing obviously lets you find bugs
- Versioning helps you track down issues
- Documenting helps everyone understand the code

J

Familiarize yourself with available tools

- J Design: UML might be overkill, but . . .
- J Testing: Run-tests, SimpleTest, PHPUnit, . . .
- J Versioning: SVN, HG, GIT





Micro Optimizations

M Always write optimized code

- Optimized code usually is harder to maintain
- Harder to maintain code is often more error prone
- Writing optimized code takes longer

J Follow the 80 : 20 rule

- J 80% of the time is spent in 20% code
- J Optimizing the 80% by 20% gains: 4%
- J Optimizing the 20% by 10% gains: 8%
- J Use Profiling System Profiling





M Using references to optimize code

References don't do what you think they do

Do not use references (avoid them like holy water)





```
function ConfigFramework(ARRAY $config) {
   // . . .
$config = array(...);
ConfigFramework($config);
class Application {
   function __construct($config) {
       $this->config = $config;
```



\$app = new Application(\$config);

Börger, Schlüter Worst PHP Practice 17



```
function ConfigFramework(ARRAY $config) {
   // Expensive read function
config = array(...);
ConfigFramework($config);
// This configure stuff is somehow slow
class Application {
   function __construct($config) {
       $this->config = $config;
```



\$app = new Application(\$config);



```
function ConfigFramework(ARRAY &$config) {
   // Expensive read function
config = array(...);
ConfigFramework($config);
// Should be faster now, no?
class Application {
   function __construct($config) {
       $this->config = $config;
```



\$app = new Application(\$config);

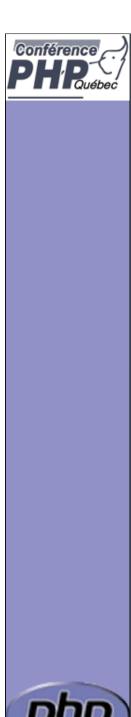
Börger, Schlüter Worst PHP Practice 13



```
function ConfigFramework(ARRAY &$config) {
   // Expensive read function
config = array(...);
ConfigFramework($config);
// Now $config is a reference
class Application {
   function __construct($config) {
       $this->config = $config;
// And now the following is slow
$app = new Application($config);
```



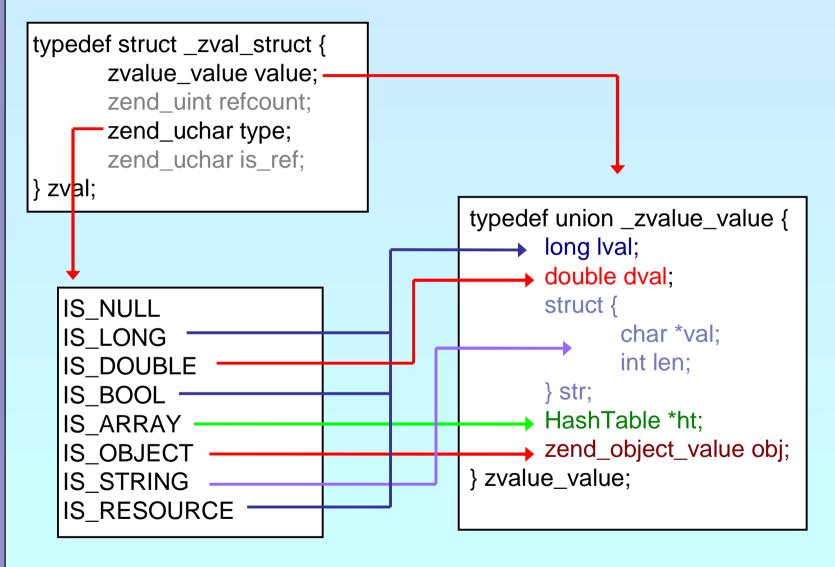
Börger, Schlüter Worst PHP Practice 14







In PHP all values are zval's







In PHP all values are zval's

```
typedef struct _zval_struct {
    zvalue_value value;
    zend_uint refcount;
    zend_uchar type;
    zend_uchar is_ref;
} zval;

Userspace notion of "Reference"

0 == Not a reference

1 == Is a reference
```

How many "labels" are associated with this zval?



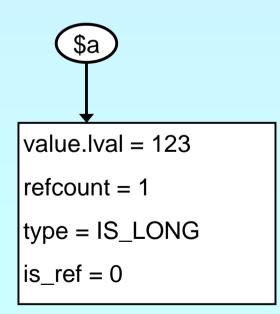


Copy On Write

```
typedef struct _zval_struct {
    zvalue_value value;
    zend_uint refcount;
    zend_uchar type;
    zend_uchar is_ref;
} zval;

• Has a value of 0 (zero)
• zval shared by 1 or more labels
• If one label wants to make a change, it must leave other labels with the original value.
```

$$a = 123$$
;







Copy On Write

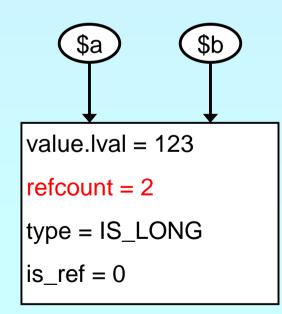
```
typedef struct _zval_struct {
    zvalue_value value;
    zend_uint refcount;
    zend_uchar type;
    zend_uchar is_ref;
} zval;

• Has a
• zval s
• zval s
• If one change
labels
```

- Has a value of 0 (zero)
- zval shared by 1 or more labels
- If one label wants to make a change, it must leave other labels with the original value.

```
a = 123;
```

$$b = a;$$







Copy On Write

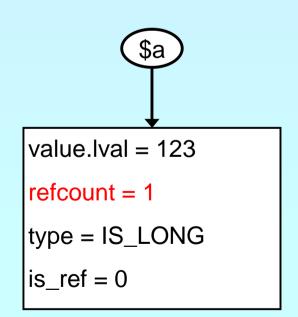
```
typedef struct _zval_struct {
    zvalue_value value;
    zend_uint refcount;
    zend_uchar type;
    zend_uchar is_ref;
} zval;

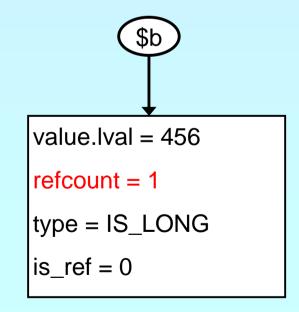
• Has a value of 0 (zero)
• zval shared by 1 or more labels
• If one label wants to make a change, it must leave other labels with the original value.
```

a = 123;

b = a;

b = 456;









Full Reference

```
typedef struct _zval_struct {
    zvalue_value value;
    zend_uint refcount;
    zend_uchar type;
    zend_uchar is_ref;
} zval;

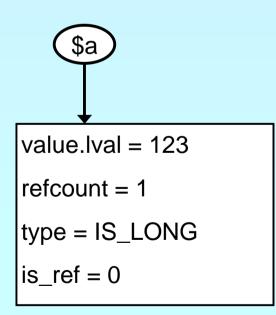
• Has a
• zval s
• zval s
• If one change
labels
```

Has a value of 1 (one)

zval shared by 1 or more labels

 If one label wants to make a change, it does so, causing other labels to see the new value.

```
a = 123;
```







Full Reference

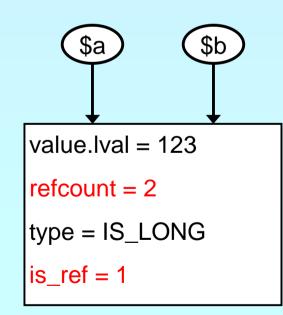
```
typedef struct _zval_struct {
    zvalue_value value;
    zend_uint refcount;
    zend_uchar type;
    zend_uchar is_ref;
} zval;

• Has
• zval
• zval
• If one chang
labels
```

- Has a value of 1 (one)
- zval shared by 1 or more labels
- If one label wants to make a change, it does so, causing other labels to see the new value.

```
a = 123;
```

$$b = \$a;$$







Full Reference

```
typedef struct _zval_struct {
    zvalue_value value;
    zend_uint refcount;
    zend_uchar type;
    zend_uchar is_ref;
} zval;
```

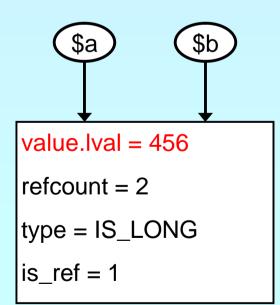
- Has a value of 1 (one)
- zval shared by 1 or more labels
- If one label wants to make a change, it does so, causing other labels to see the new value.

23

```
a = 123;
```

b = \$a;

b = 456;







Do everything with Objects

- M Everything must be an object
- PHP supports procedural code
- J When you use a singleton factory
 - J You could have used globals
- J An object that simply stores values
 - J Could simply be an array





Include vs. Require vs _once

marked require_once is the safe and correct way - always

There are four verisons for a reason

J include

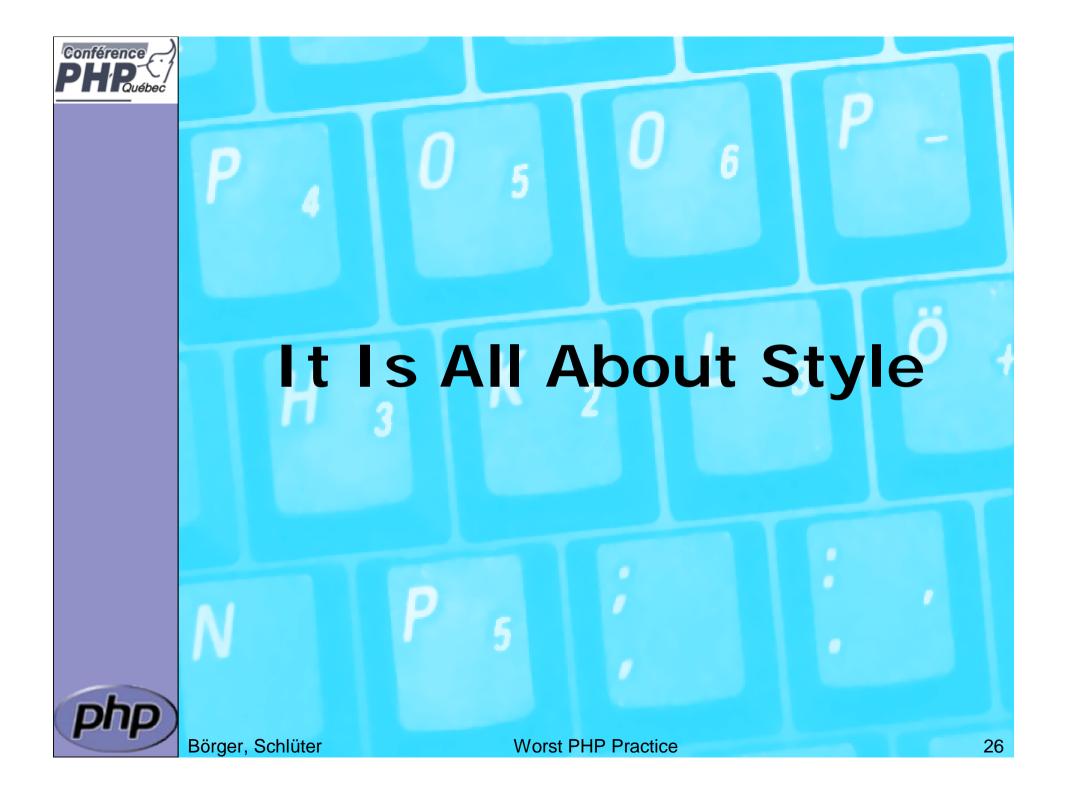
J require

J include_once / require_once

J fpassthru()

M eval







Provide a Style Guide

J Provide actual coding rules (coding style)

Provide useful error handling

Always develop with E_STRICT + E_NOTICE on

Use your logs

Use .inc for includes + care for server config

Use 'instead of "

Do not constantly switch between HTML and PHP

Do not use auto_prepend_file, auto_append_file

Do not leave debugging in production

Do we really need to mention register_globals?

No Magic quotes - But Filter input & escape output





Use with Caution

J

J

J

J

J

J

\$_REQUEST

__get, __set, __isset, __unset

__call, __callStatic

__autoload

@

<?=





b Everythining about PHP

http://php.net

b These slides

http://talks.somabo.de

b George Schlossnagle

Advanced PHP Programming

Andi Gutmans, Stig Bakken, Derick Rethans

PHP 5 Power Programming

