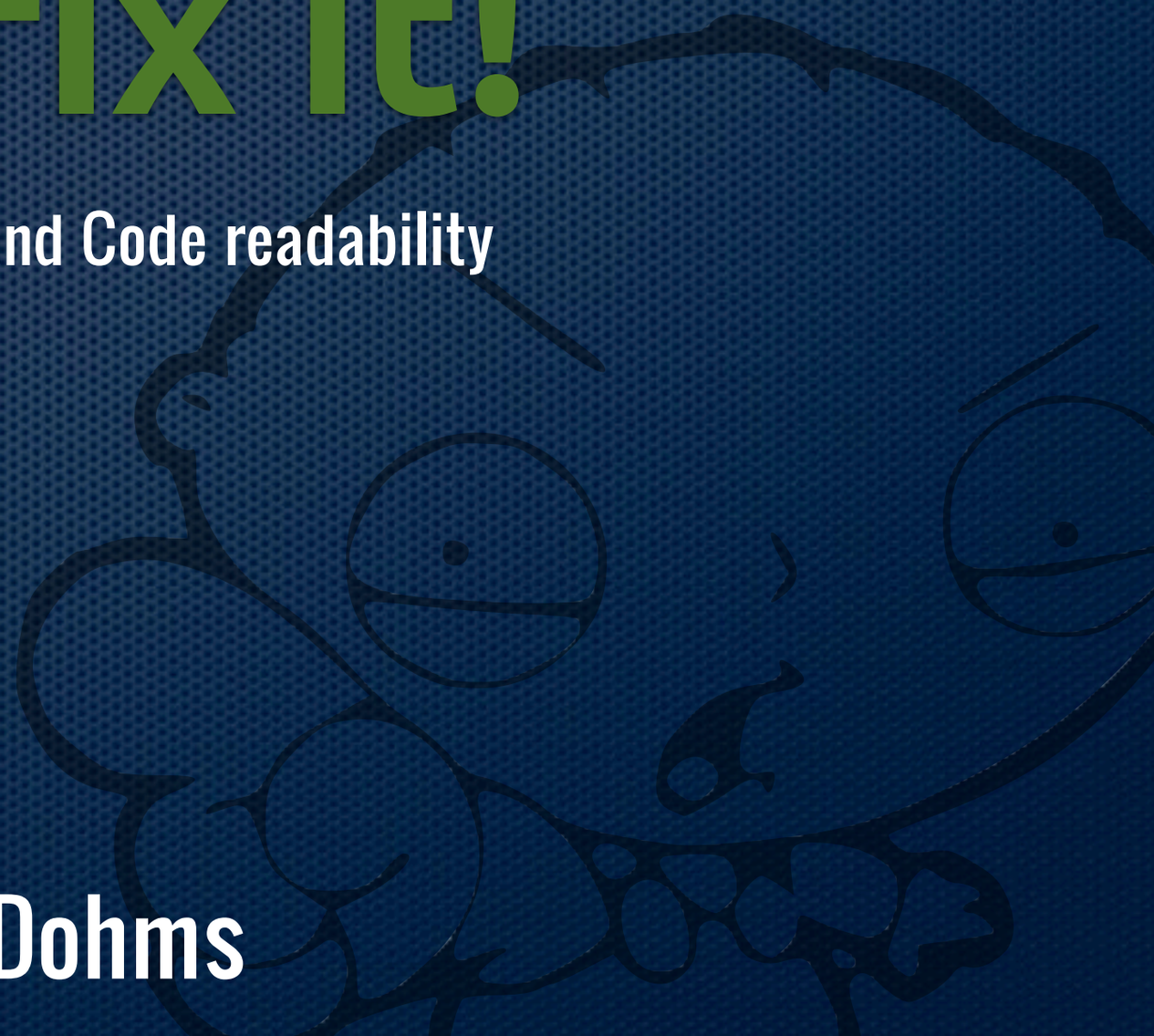
Your code sucks, let's fix it!

Object Calisthenics and Code readability



Rafael Dohms

Rafael Dohms


photo credit: Eli White

PHP Evangelist, Speaker and contributor. He is a very active member of the PHP Community, having helped create and manage two PHP User Groups in Brazil. He shared the **lead of PHPSP** for 3 wonderful years making a positive mark on the local market. Developer, gamer and lover of code he also hosts Brazil's first PHP Podcast: **PHPSPCast**, as well as contributing to well known projects.

He moved to the **Netherlands** in search of new challenges and is now part of the team at **WEBclusive**.

What's the talk about?

- Why does my code suck?
- How can we fix it?

Is it **Maintainable**?

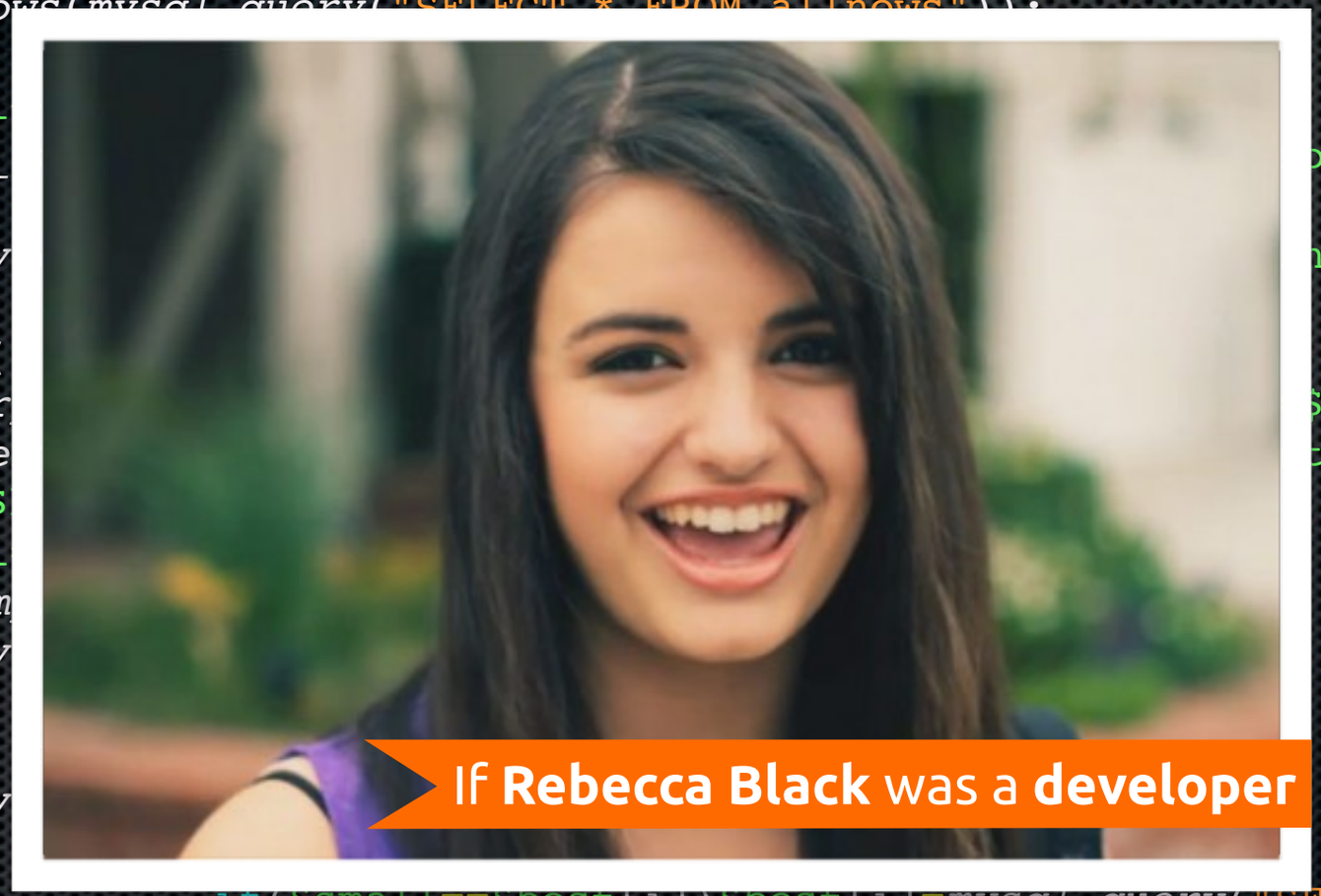
Is it **Readable**?

Why does my code suck?

Is it **Reusable**?

Is it **Testable**?

Does it **look** like this?




```
<?php
$list=mysql_connect("******","*******","*****");
if(!$list)echo 'Cannot login.';
else{
    mysql_select_db("******", $list);
    $best=array("0","0","0","0","0","0");
    $id=mysql_num_rows(mysql_query("SELECT * FROM allnews"));
    $count=0;
    for($i=0;$i<6;$i
        while(mysql_
+;
        $best[$i]=my
    $id=$id-$count;
    $maxdate=mktime(
    while(mysql_quer
        if(mysql_que
            $small=$
            while($i
                if(m
$small)"<mysql_query

$small")<mysql_query

                if($small==$best[i])$best[i]=mysql_query("SELECT id FROM
allnews WHERE id=$id-$count");}}}}
    while($i=0;$i<6;$i++)
    echo '<a href="news-page.php?id='.$best[i].'"><div class="box '.mysql_query("SELECT
type FROM allnews WHERE id=$best[i]").'">'.mysql_query("SELECT title FROM allnews WHERE id=
$best[i]").'<div class="img" style="background-image:url(images/'.mysql_query("SELECT
image1 FROM allnews WHERE id=$best[i]").');"></div></div></a>';
    mysql_close($list);
}
?>
```

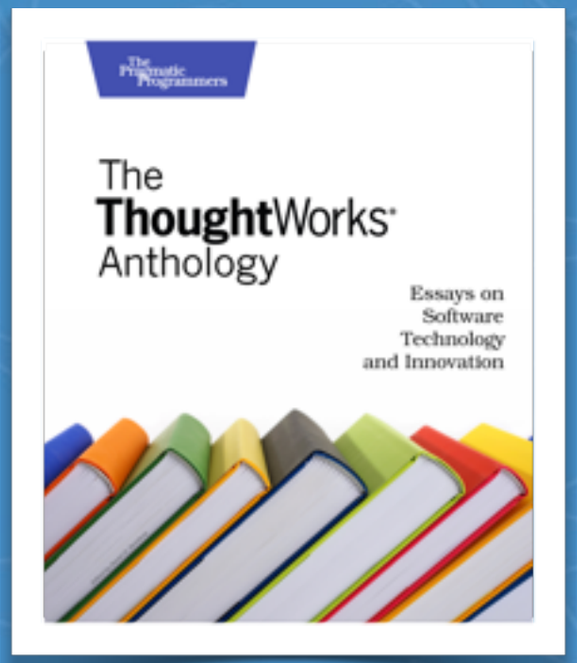
How do we fix it?

cal·is·then·ics - noun - /ˌkaləsˈTHeniks/

Calisthenics are a form of dynamic exercise consisting of a variety of simple, often rhythmical, movements, generally using minimal equipment or apparatus.

Object Calisthenics

A variety of **simple**, often **rhythmical**, exercises to achieve **better** OO and **code quality**



"So here's an exercise that can help you to internalize principles of good object-oriented design and actually use them in real life."

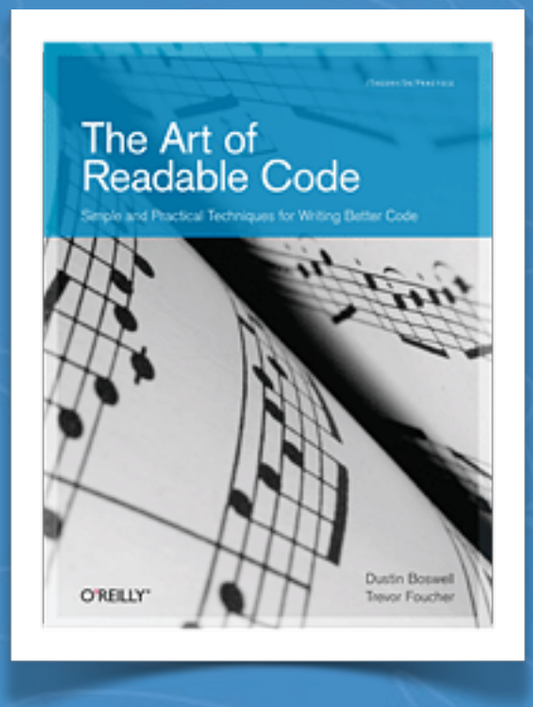
-- Jeff Bay

Object Calisthenics

Important:

PHP != JAVA

Adaptations will be done



"You need to write code that minimizes the time it would take someone else to understand it—even if that someone else is you."

-- Dustin Boswell and Trevor Foucher

Readability Tips

I'm a **tip**

Disclaimer:
“These are guidelines, not rules”

OC #1

“Only one indentation level per method”

```php
function validateProducts($products) {

    // Check to make sure that our valid fields are in there
    $requiredFields = array(
        'price',
        'name',
        'description',
        'type',
    );

    $valid = true;

    foreach ($products as $rawProduct) {
        $fields = array_keys($rawProduct);

        foreach ($requiredFields as $requiredField) {
            if (!in_array($requiredField, $fields)) {
                $valid = false;
            }
        }
    }

    return $valid;
}
```

0 1 2 3

```php
function validateProducts($products) {

    // Check to make sure that our valid fields are in there
    $requiredFields = array(
        'price',
        'name',
        'description',
        'type',
    );

    $valid = true;

    0 whitespace

    awProduct) {
        validateSingleProduct($rawProduct, $requiredFields);

    1 if ( ! $validationResult){
        2 $valid = false;
        }
    }

    return $valid;
}

function validateSingleProduct($product, $requiredFields)
{
    $valid = true;

    duplicated logic

    $fields = array_keys($rawProduct);

    0 foreach ($requiredFields as $requiredField) {
        1 if (!in_array($requiredField, $fields)) {
            2 $valid = false;
        }
    }
    return $valid;
}
```

```php
function validateProducts($storeData) {

    $requiredFields = array('price','name','descriptio

    foreach ($storeData['products'] as $rawProduct) {
        if ( ! validateSingleProduct($rawProduct, $requiredFields)) return false;
    }

    return true;
}
```

I see cheating!

Single line IF, simple operations

return early

```php
function validateSingleProduct($product, $requiredFields)
{
    $fields = array_keys($rawProduct);
    $missingFields = array_diff($requiredFields, $fields);

    return (count($miss
}
```

C (native) functions are faster then PHP

List is more readable the plural

faster iteration

```
function validateProductList($p
{
    $invalidProducts = array_filter($products, 'isInvalidProduct');

    return (count($invalidProducts) === 0);
}
```

readable return: zero invalid products

reusable method

```
function isInvalidProduct($rawProduct)
{
    $requiredFields = array('price', 'name', 'description', 'type');

    $fields         = array_keys($rawProduct);
    $missingFields  = array_diff($requiredFields, $fields);

    return (count($missingFields) > 0);
}
```

method name matches "**true**" result

Key Benefits

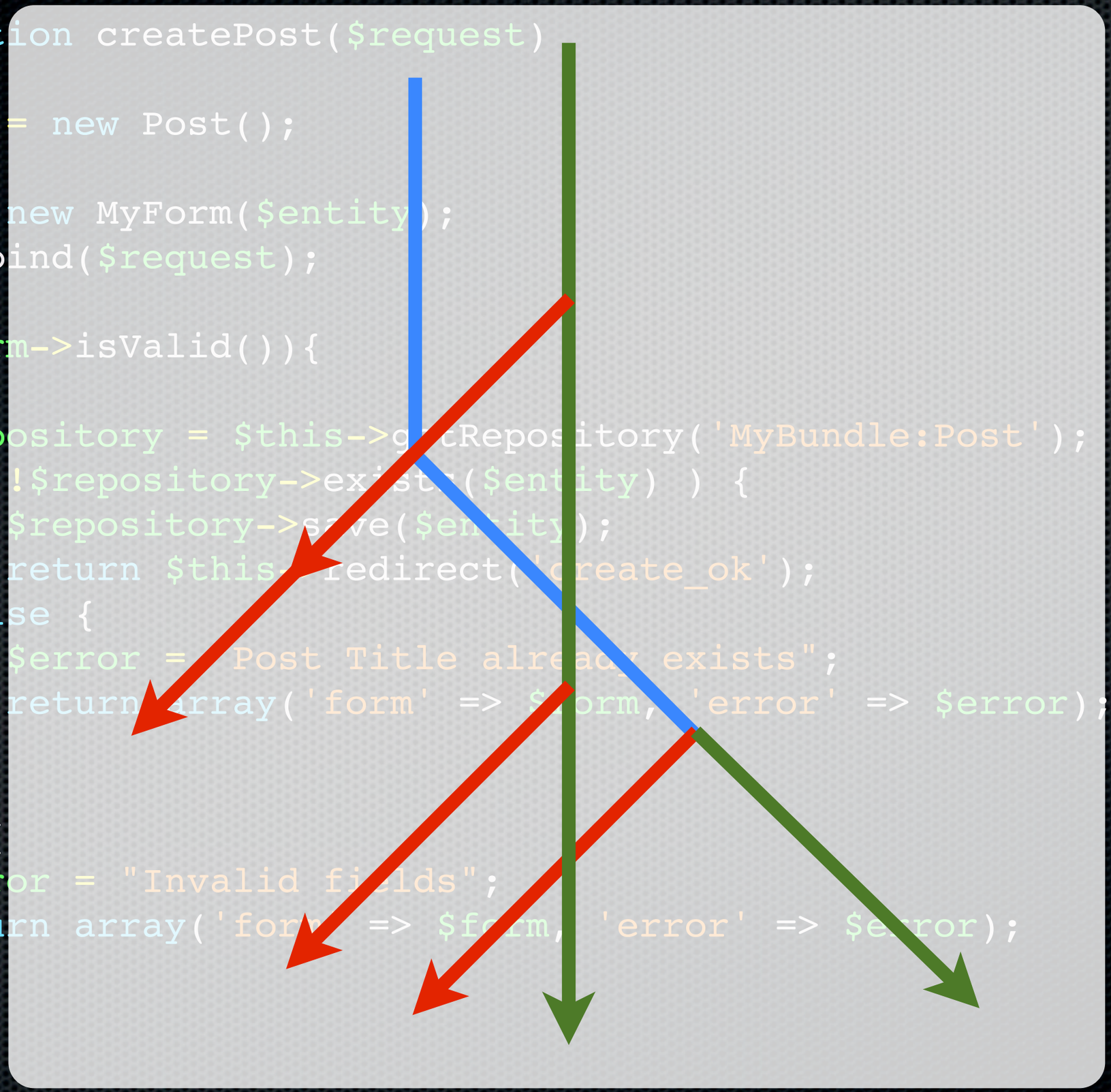
- Cohesiveness
- Methods does only one thing
- Increases re-use

OC #2

“Do not use the ‘else’ keyword”

```php
public function createPost($request)
{
    $entity = new Post();

    $form = new MyForm($entity);
    $form->bind($request);

    if ($form->isValid()){

        $repository = $this->getRepository('MyBundle:Post');
        if (!$repository->exists($entity) ) {
            $repository->save($entity);
            return $this->redirect('create_ok');
        } else {
            $error = "Post Title already exists";
            return array('form' => $form, 'error' => $error);
        }

    } else {
        $error = "Invalid fields";
        return array('form' => $form, 'error' => $error);
    }

}
```

```php
public function createPost($request)
{
    $entity = new Post();

    $form = new MyForm($entity);
    $form->bind($request);

    if ($form->isValid()){

        $repository = $this->getRepository('MyBundle:Post');
        if (!$repository->exists($entity) ) {
            $repository->save($entity);
            return $this->redirect('create_ok');
        } else {
            $error = "Post Title already exists";
            return array('form' => $form, 'error' => $error);
        }

    } else {
        $error = "Invalid fields";
        return array('form' => $form, 'error' => $error);
    }

}
```



```php
public function createPost($request)
{
    $entity = new Post();

    $form = new MyForm($entity);
    $form->bind($request);

    if ($form->isValid()){

        $repository = $this->getRepository('MyBundle:Post');
        if (!$repository->exists($entity) ) {
            ...entity);
            ...ect('create_ok');
        } else {
            $error = "Post Title already exists";
            return array('form' => $form, 'error' => $error);
        }
    } else
        $error = "Invalid fields";
        return array('form' => $form, 'error' => $error);
    }

}
```

intermediate variable

intermediate variable

```php
public function createPost($request)
{
    $entity     = new Post();
    $repository = $this->getRepository('MyBundle:Post');

    $form = new MyForm($entity);
    $form->bind($request);

    if ( ! $form->isValid()){
        return array('form' => $form, 'error' => 'Invalid fields');
    }

    if ($repository->exists($entity)){
        return array('form' => $form, 'error' => 'Duplicate post title');
    }

    $repository->save($entity);

    return $this->redirect('create_ok');

}
```

Separate code into **blocks**.

Its like using **Paragraphs**.

removed intermediates

early return

Key Benefits

- Helps avoid code duplication
- Makes code clearer (single path)

ADAPTED

OC #3

“Wrap ~~all~~ primitive types and string, if it has behaviour”

```php
class UIComponent
{
//...
    public function repaint($animate = true){
            //...
    }

//...
$component->repaint(false);
```

unclear operation

```
class UIComponent
{
    //...
    public function repaint( Animate $animate ){
            //...
    }
}

class Animate
{
    public $anim
```

This can now encapsulate all animation related operations

```
    public function __construct( $animate = true ) {
        $this->animate = $animate;}

}

//...
$component->repaint( new Animate(false) );
```

Key Benefits

- Type Hinting
- Encapsulation of operations

ADAPTED

OC #4

“Only one -> per line, if not getter or fluent”

Source: CodeIgniter

properties are harder to mock

```
$this->base_url = $this->CI->config->site_url().'/'.$this->CI->uri->segment(1).$this->CI->uri->slash_segment(2, 'both');
```

no whitespace

```
$this->base_uri = $this->CI->uri->segment(1).$this->CI->uri->slash_segment(2, 'leading');
```

move everything to **uri** object

```
$this->getCI()->getUriBuilder()->getBaseUri('leading');
```

- Underlying encapsulation problem
- Hard to debug and test
- Hard to read and understand

Source: Zend Framework App

fluent interface

```
$filterChain->addFilter(new Zend_Filter_Alpha())
            ->addFilter(new Zend_Filter_StringToLower());
```

operator alignment

only getters (no operations)

```
$user = $this->get('security.context')->getToken()->getUser();
```

where did my autocomplete go?

return **null**?

Source: Symfony 2 Docs.

Key Benefits

- Readability
- Easier Mocking
- Easier to Debug
- Demeter's Law

OC #5

“Do not Abbreviate”

```
if($sx >= $sy) {
    if ($sx > $strSysMatImgW) {
        $ny = $strSysMatImgW * $sy / $sx;
        $nx = $strSysMatImgW;
    }

    if ($ny > $strSysMatImgH) {
        $nx = $strSysMatImgH * $sx / $sy;
        $ny = $strSysMatImgH;
    }

} else {
    if ($sy > $strSysMatImgH) {
        $nx = $strSysMatImgH * $sx / $sy;
        $ny = $strSysMatImgH;
    }

    if($sx > $strSysMatImgW) {
        $ny = $strSysMatImgW * $sy / $sx;
        $nx = $strSysMatImgW;
    }
}
```

Why?

Its **repeated** many times, and i'm **lazy**.

Underlying Problem!

You need to transfer those operations into a separate class.

Why?

more then one responsibility?

```
function processResponseHeadersAndDefineOutput($response) { ... }
```

This **method name** is too **long** to type, and i'm **lazy**.

get from where?

```
function getPage($data) { ... }
```

Use clearer names:
fetchPage()
downloadPage()

```
function startProcess() { ... }
```

Use a thesaurus:
fork, create, begin, open

Table row?

```
$tr->process("site.login");
```

Easy understanding, complete scope:
$translatorService

Key Benefits

- Clearer communication
- Indicates underlying problems

ADAPTED

OC #6

“Keep your classes small”

Increased to include **docblocks**

15-20 lines per method

100 lines per class

15 classes per package

read this as **namespace** or **folder**

Key Benefits

- Single Responsibility
- Objective methods
- Slimmer namespaces
- Less clunky folders

ADAPTED

OC #7

“Limit the number of instance variables in a class (max: ~~2~~ 5)”

```php
class MyRegistrationService
{

    protected $userService;
    protected $passwordService;
    protected $logger;
    protected $translator;
    protected $entityManager;
    protected $imageCropper;

    // ...
}
```

All DB interaction should be in userService

Use and event based system and move this to listener

Limit: 5

Key Benefits

- Shorter dependency list
- Easier Mocking for unit test

OC #8

“Use first class collections”

Doctrine:
ArrayCollection

```
$collection->getIterator();

$collection->filter(...);

$collection->append(...);

$collection->map(...);
```

Key Benefits

- Implements collection operations
- Uses SPL interfaces

DROPPED

OC #9

"Do ~~not~~ use accessors (getter/setter)"

```php
/**
 * THIS CLASS WAS GENERATED BY THE DOCTRINE ORM. DO NOT EDIT THIS FILE.
 */
class DoctrineTestsModelsCMSCmsUserProxy
    extends \Doctrine\Tests\Models\CMS\CmsUser
    implements \Doctrine\ORM\Proxy\Proxy
{

    public function getId()
    {
        $this->__load();
        return parent::getId();
    }

    public function getStatus()
    {
        $this->__load();
        return parent::getStatus();
    }
```

Example: Doctrine uses getters to inject lazy loading operations

Key Benefits

- Injector operations
- Encapsulation of transformations

CREATED!

OC #10 (bonus!)

“Document your code!”

really?

```
//check to see if the section above set the $overall_pref variable to void
if ($overall_pref == 'void')
```

```
// implode the revised array of selections in group three into a string
// variable so that it can be transferred to the database at the end of the
// page
$groupthree = implode($groupthree_array, "\n\r");
```

Documenting because i'm doing it ~~wrong~~ in an anusual way

```
$priority                                                    0;

if (!isse
    throw
}
```

Add a **simple** comment:
//Strips special chars and camel cases to onXxx

```
if (!isset($event['method'])) {
    $event['method'] = 'on'.preg_replace(array(
      '/(?<=\b)[a-z]/ie',
      '/[^a-z0-9]/i'
    ), array('strtoupper("\\0")', ''), $event['event']);
}
```

Don't **explain** bad code, **fix it**!

```
$definition->addMethodCall(
    'addListenerService',
    array($event['event'],
    array($listenerId,
    $event['method']),
    $priority
));
```

What does this **do**?

Source: Symfony2

Do a **mind dump**, then clean it up.

A note on **cost** of running function

```
/**
* Checks whether an element is contained in the collection.
* This is an O(n) operation, where n is the size of the collection.
*
* @todo implement caching for better performance
* @param mixed $element The element to search for.
* @return boolean TRUE if the collection contains the element, or FALSE.
*/
function contains($element);
```

mark **todo** items so the changes don't get lost

Generate API docs
ex: docBlox

Key Benefits

- Automatic API documentation
- Transmission of “line of thought”
- Avoids confusion

Recap

- #1 - Only one indentation level per method.
- #2 - Do not use the 'else' keyword.
- #3 - Wrap primitive types and string, if it has behavior.
- #4 - Only one -> per line, if not getter or fluent.
- #5 - Do not Abbreviate.
- #6 - Keep your classes small
- #7 - Limit the number of instance variables in a class (max: 5)
- #8 - Use first class collections
- #9 - Use accessors (getter/setter)
- #10 - Document your code!

@rdohms
rafael **@doh.ms**

http://doh.ms
http://**slides**.doh.ms

Questions?

https://joind.in/6125

Recommended Links:

https://joind.in/6125

The ThoughtWorks Anthology

http://goo.gl/OcSNx

The Art of Readable Code

http://goo.gl/unrij

DISCLAIMER: This talk re-uses some of the examples used by **Guilherme Blanco** in his original Object Calisthenic talk. These principles were studied and applied by us while we worked together in previous jobs. The result taught us all a lesson we really want to spread to other developers.