Migrating a Zend Framework application to ZF 2

Bart McLeod

PFcongres, September 14, 2013

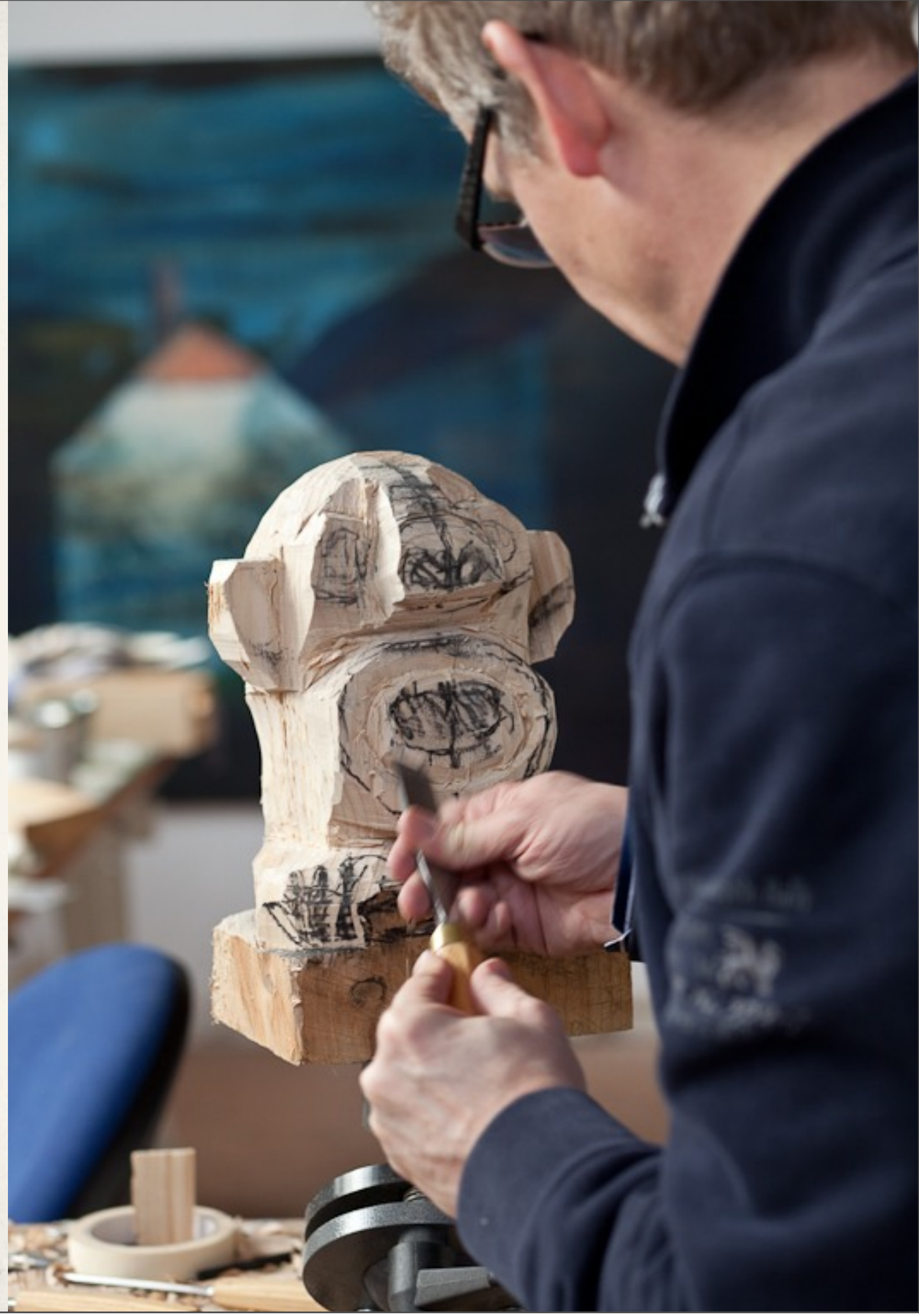
What to think of?

- New MVC
- Similar views
- Similarities in controllers
- New Zend\Form
- New Zend\Db
- Drop in Modules
- Less magic
- Relying more on native, such as the native Locale and DateTime classes

Bart McLeod

- Not natively Scottish
- Dutch
- Zend Framework developer, coach, writer and speaker.
- Painter and Sculptor

@bartmcleod
http://bartmcleod.nl



ZF 1

Photo courtesy:
http://wondrouspics.com/new-york-city-of-opportunities/

ZF 2 ?

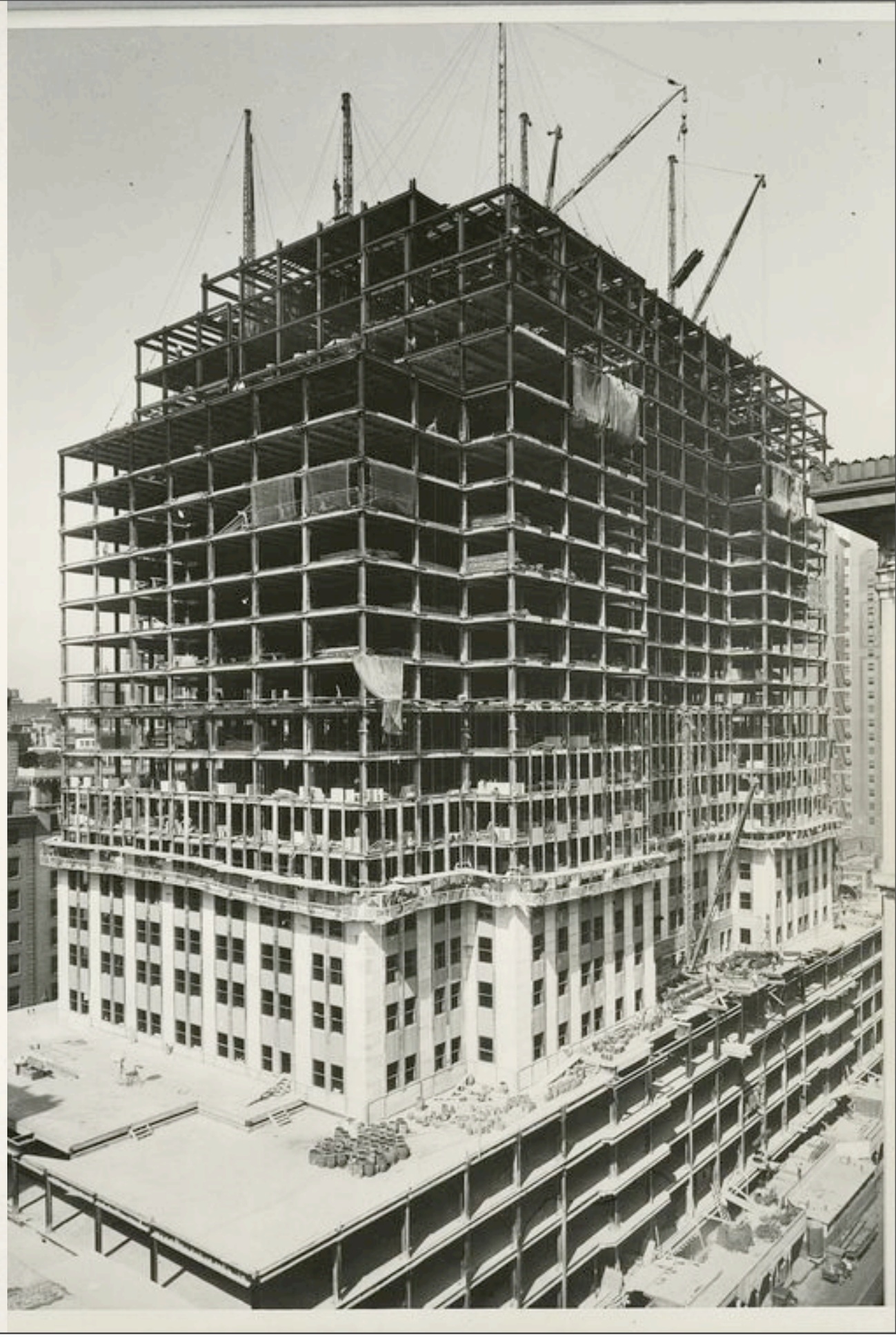
Photo courtesy: http://leslieannrabbon.wordpress.com/

ZF 1 -> ZF 2

How to approach it?

- Think of a multistory building
- Replace the foundation and the construction?
- You can’t, it would collapse
- So build a new multistory first
- And then move the tenants from the old one
- Yes, that might be a lot of work

Hard Work



Photograph by Lewis Wickes Hine / NYPL Digital Gallery

You need a plan

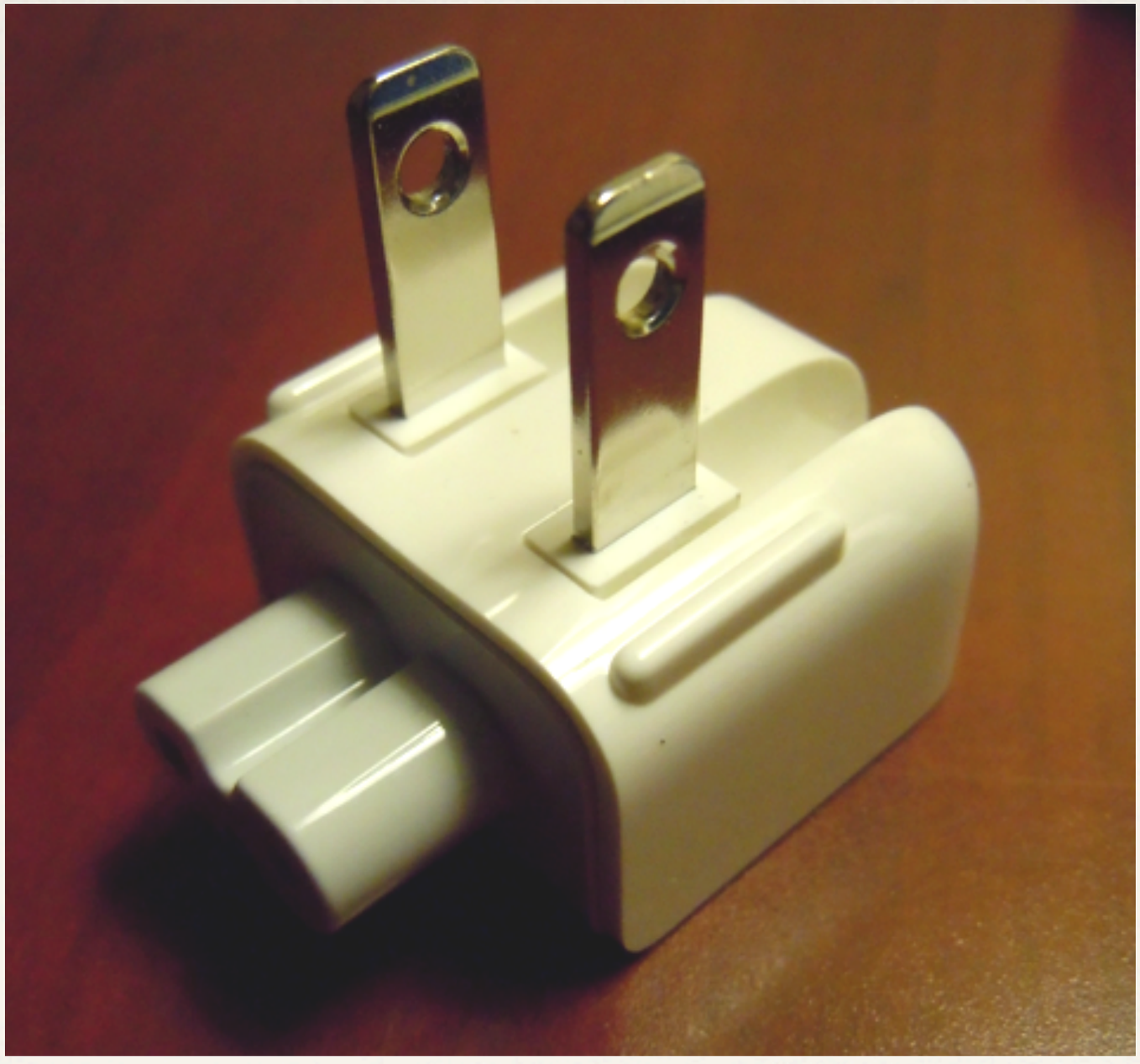
- If I just hack my way through my old code and try to get it working on ZF 2, I get stuck. How about you?
- Study ZF 2 concepts
- Understand the concepts
- Try them
- Use them to fit your business logic
- Plan a migration like you would plan any other project
- Start with the ZendSkeletonApplication

New MVC

- Fully flexible
- Services
- Events
- DI Configurations by using ServiceLocator pattern
- Explicit: you need more code, you get more power
- You might not recognize it as Zend Framework at first

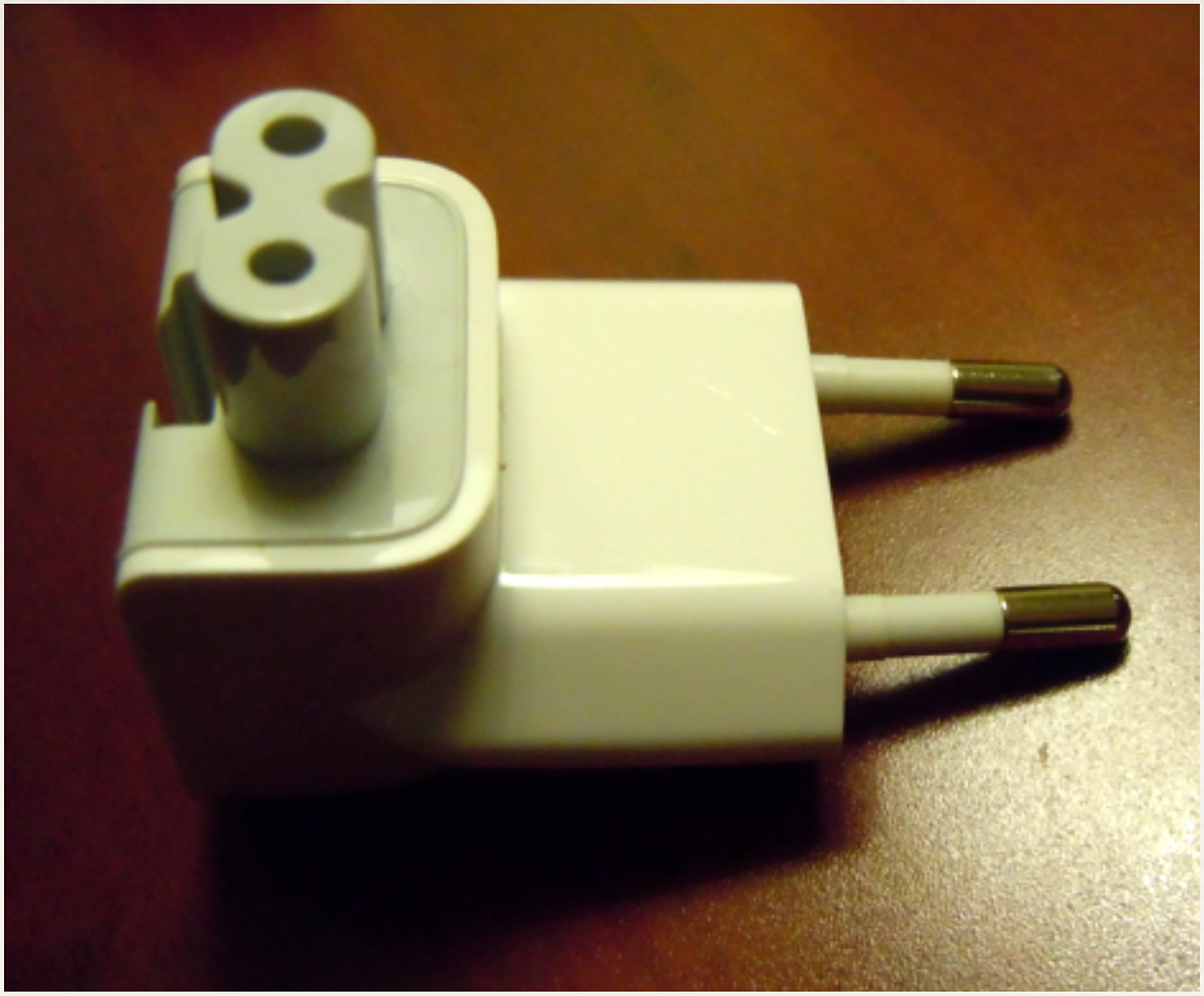
Apple\Outlet\AdapterAwareInterface

public function setAdapter(Apple\Outlet\AdapterInterface $adapter){}



Apple\Outlet\Adapter\Us

implements Apple\Outlet\AdapterInterface



Apple\Outlet\Adapter\Europe

implements Apple\Outlet\AdapterInterface

Modular by Nature

- Modules that you just use and do not change go into the ‘vendor’ directory
- Your own application modules live in the ‘module’ directory
- A Module solves a single business problem
- Is re-usable across ZF 2 projects
- When planning a migration you think of what functionality belongs where
- Modules may depend on other modules
- Module.php is the only requirement

Configuration

- You configure the application by writing php code
- No environment specific sections in *.ini files
- Per module configuration in Module::getConfig()
- Override module configuration {.*}global.php
- Per environment configuration in {.*}local.php, overrides all others
- You may still use ini files, but arrays are faster and you will find more examples of them

Example configuration

```php
<?php
return array(
    'controllers' => array(
        'factories' => array(
            'frontend' => 'CfFrontend\Controller\ControllerFactory',
        ),
    ),
    'view_manager' => array(
        'template_path_stack' => array(
            'cf-frontend' => __DIR__ . '/../view',
        ),
        'template_map' => array(
            'layout/cf-frontend' => __DIR__ . '/../view/layout/cf.phtml',
        ),
    ),
);
```

Autoloading

```
public function getAutoloaderConfig()
{
    return array(
        'Zend\Loader\ClassMapAutoloader' => array(
            'CuddleFish' => 'vendor/CuddleFish/library/class_map.php',
            'SpaceCMS' => 'vendor/SpaceCMS/autoload_classmap.php',
            'ZendX' => 'vendor/ZendX/autoload_classmap.php',
            'Zend' => 'vendor/Zend/autoload_classmap.php',
        ),
        'Zend\Loader\StandardAutoloader' => array(
            'namespaces' => array(
                __NAMESPACE__ => __DIR__ . '/src/' . __NAMESPACE__,
            ),
        ),
    );
}
```

Authentication

- Don't write it yourself anymore
- Use ZfcUser or another drop in module
- You may need to alter your user table, if you use a database
- Or you might want to specify different columns to be used by ZfcUser
- No longer use md5 for passwords, but you already knew
- Set new passwords for ZfcUser with encrypt in MySQL

Routing

- Route types: Literal, Segment, Wildcard, Part, Hostname, Query, Method, Regex
- Collision: /[:code] and /admin, What if /:[code] matches first? Priority! Higher priority means earlier match, higher means higher as in numbers.
- Constraints: array('code' => "[a-z]*")
- SimpleRouteStack
- TreeRouteStack (default)
- RoutePluginManager and route plugins

Routing example

```
'router' => array(
    'routes' => array(
        'cf-frontend' => array(
            'type' => 'Segment',
            'options' => array(
                'route'    => '/[:code]',
                'defaults' => array(
                    'controller' => 'frontend',
                    'action'     => 'index',
                    'code'       => 'home',
                ),
            ),
            'priority'      => 100,
        ),
    ),
),
```

Compare with ZF 1 route

```
;ZF 1, typically in application.ini or routes.ini
routes.article.type = "Zend_Controller_Router_Route"
routes.article.route = ":subject"
routes.article.defaults.module = "default"
routes.article.defaults.code = "home"

// ZF 2 (snippet from previous example)
'cf-frontend' => array(
    'type' => 'Segment',
    'options' => array(
        'route'    => '/[:code]', // the first slash matters!
        'defaults' => array(
            'controller' => 'frontend',
            'action'     => 'index',
            'code'       => 'home',
        ),
    ),
    'priority'      => 100,
),
```

Layout

- Configure your module
- Setup module specific layout if you need to
- Or configure with key ‘layout/layout’ in a *.global.php
- Put the full path in the template_map configuration
- Replace baseUrl() view helper by basePath()
- Comment out all of your own view helpers until you migrated them
- Inside action: $view->layout()->title = ‘ZF 1 -> ZF 2 Migration’;

Example layout (snippet)

```
<div id="breadcrumbs" >
    <?php
        echo $this->cfNavigation()
            ->setRoute($route) // route does not mean ZF route here
            ->setCurrentSubject($currentSubject)
            ->breadCrumbs();
    ?>
</div>
<div id="nav">
    <?php
       echo $this->cfNavigation();
    ?>
</div>
<div id="content">
    <h1><?= ucfirst($title) ?></h1>
    <?= $this->content ?>
 </div>
```

View Scripts

- Use $variable instead of $this->variable
- $this->variable still works
- Converting is relatively easy with find and replace
- Optionally comment out postponed view helpers
- Different escape methods for each context escape() is now escapeHtml()

Passing view variables

- Return an associative array from the controller
- Or return a ViewModel from the controller
- A ViewModel allows for customization, for example setting a different template
- We used to write $this->render('alternative'); // inside action
- Now we write $view->setTemplate('alternative'); // inside action
- Or: $view->setTemplate('book/index/hello.phtml'); // to be found in path stack
- Configure the alternative view script key in the template_map
- Views and ViewModels have 'grown up'. There are many new possibilities, RTFM.

Preparing a view

```
// option 1: returning an associative array
return array('fruits' => array('orange', 'kiwi'));

// option 2: returning a ViewModel instantiated with an array
return new ViewModel(array('fruits' => array('orange', 'kiwi')));

// option 3: create a ViewModel and populate it using overloading
$view = new ViewModel();
$view->fruits = array('orange', 'kiwi');

// disable layout for this action:
$view->setTerminal(true);

// change the template for the index action:
// this will render fruit.phtml (the alternative script)
// it is configured in the template_map key of the view_manager
$view->setTemplate('fruit');
return $view;
```

View Helpers

- Extend Zend\View\Helper\AbstractHelper for convenience
- __invoke Method instead of a 'classname' method (constructor conflict)
- Simple view helpers may use __toString() only
- Inject dependencies into your view helpers
- Shipped view helpers are available like before: $this->view->url()
- Attention: order of arguments for url() view helper changed. Route comes first.
- Postpone migration: return __CLASS__ from __invoke()

View helper example

```php
<?php
namespace CfFrontend\View\Helper;
use Zend\View\Helper\HelperInterface;
use Zend\View\Renderer\RendererInterface;

class Hello implements HelperInterface {
    protected $view;

    public function __toString(){
        return 'Hello from ViewHelper Hello!';
    }

    public function setView(RendererInterface $view)
    {
        $this->view = $view;
    }

    public function getView()
    {
        return $this->view;
    }
}
```

View helper simpler

```php
<?php
namespace CfFrontend\View\Helper;
use Zend\View\Helper\AbstractHelper;

class HelloSimpler extends AbstractHelper {

    public function __toString(){
        return 'Hello from ViewHelper HelloSimpler!';
    }
}
```

‘Advanced’ view helper

```php
<?php
namespace CfFrontend\View\Helper;
use Zend\View\Helper\AbstractHelper;

class LanguageChoice extends AbstractHelper
{
    protected $languageService;

    public function __invoke()
    {
        return $this;
    }

    public function setLanguageService($ls)
    {
        $this->languageService = $ls;
        return $this;
    }

    public function __toString(){[...]} // produces actual output
}
```

View helper config

```
public function getViewHelperConfig()
{
    return array(
        'invokables' => array(
            'hello'   => 'CfFrontend\View\Helper\Hello',
        ),
        'factories' => array(
            'languageChoice' => function(HelperPluginManager $pm){
                $chooser = new View\Helper\LanguageChoice();
                $sl = $pm->getServiceLocator();
                $languageService = $sl->get('CfFrontend\LanguageService');
                $chooser->setLanguageService($languageService);
                return $chooser;
            },
        ),
    );
}
```

Action Helpers

- Now used as controller plugins
- There are some pretty powerful predefined action helpers
- To use the built-in ones, it is convenient to extend your controller from one of the abstract controllers
- You can build your own plugin, but you won't find an example in the manual yet
- But y'll get one right now

Controller plugin example

```php
<?php
namespace CfFrontend\Controller\Plugin;

use Zend\Mvc\Controller\Plugin\AbstractPlugin;

class NavigationInfo extends AbstractPlugin
{
    public function __invoke()
    {
        $match = $this->getController()->getEvent()->getRouteMatch();
        $code = $match->getParam('code');
        $route = $match->getParam('route');
        return compact('code' , 'route');
    }
}
```

Controller plugin usage

```
// inside module.config.php [ returned by Module::getConfig() ] :

'controller_plugins' => array(
    'invokables' => array(
        'navigationInfo'
           => 'CfFrontend\Controller\Plugin\NavigationInfo',
    )
),

// usage inside controller action :

extract($this->navigationInfo());
```

Events vs. Event Hooks

- ZF 1 used event hooks: routeStartup, routeShutdown, preDispatch, postDispatch
- ZF 2 uses EventManager and you attach your events
- AbstractController still has a dispatch hook for convenience

Event example

```
// http://blog.evan.pro/module-specific-layouts-in-zend-framework-2

$sharedEvents = $moduleManager->getEventManager()->getSharedManager();

$layout = $this->layout; // key of layout in template_map

$sharedEvents->attach(
    $this->namespace,
    'dispatch',
    function($e) use ($layout) {
        $controller = $e->getTarget();
        $controller->layout($layout);
    },
    100
);
```

New Zend\Db

- Zend_Db_Table is now Zend\Db\TableGateway\AbstractTableGateway
- Look into Zend\Db\RowGateway\RowGateway for methods like save() and delete()
- Convenience methods gone? FetchOne, FetchRow, FetchColumn? All we get back is a ResultSet, $result->current() gives RowGateway
- You might want to re-use old code, but that uses 'bad' practices: Zend_Registry and worse: Globals. Refactor everything to use ServiceLocator!
- Study the concepts and do it gradually.
- Unit tests should help ensuring that the code still behaves
- You might want to look at Doctrine 2 instead

Zend\Db example

- CuddleFish had a DAGO (Data Access Gateway Object)
- It could do 'smart' things, like getting all articles for a page
- Just like Globals, it attracted features like a magnet
- In ZF 2, the recommended pattern is different – it already was for ZF 1 :-)
- Your code is more lightweight and re-usable if you implement the recommended pattern
- Old: ask Globals class for Db adapter, inject that into DAGO, ask DAGO for articles
- New: Create an ArticleService, an ArticleEntity and an ArticleTable that extends AbstractTableGateway. The service will use the table to get the articles. Create a factory for both the table and the service and configure these. The controller gets a setArticleService method.

Beware of name changes

- \Zend\Session\Namespace is not possible, because Namespace is a reserved word in php > 5.3.
- The replacement in this case is: \Zend\Session\SessionContainer.
- Find out how to use it: there is more to it then there used to be.

```
use Zend\Session\Container;

$session = new Container('foo');

$session->bar = 'baz';
```

Magic is back?

- Zend\View\Helper\Placeholder\Container\AbstractStandalone
- Zend\View\Helper\HeadTitle extends it
- $view->headTitle()->setSeparator(' | '); //works
- Because it calls getSeparator() internally...
- It gets the separator from the PlaceHolder container without defining a $separator property or setSeparator method... magic right?

View rendering strategies

- Replaces context switching
- Strategies for PHP, JSON, feed
- Alternative: return a response object from the controller action and set the body of the response and the headers

Demo time / Q & A

- user: bart
- pass: *******
- ZF 2: user: mcleod@spaceweb.nl
- Time left? Zend\Form\Form
- Otherwise: find the webinar about forms by Rob Allen
- Or else: read my latest column in php|a (Februari 2013 issue)

Bart McLeod

@bartmcleod

http://bartmcleod.nl

http://joind.in/8953

Your feedback is appreciated.