Escaping the Legacy Hell

International PHP Conference 2013

Benjamin Eberlei @beberlei &
Kore Nordmann @koredn

28. Oct 2013

Legacy Code

- Legacy Code is code, which is . . .
  - . . . hard / impossible to change / adapt / adopt.
  - . . . hard / impossible to put under test.

We will talk about:

- We will talk about:
  - Common Legacy Code issues
  - How to verify we do not break everything
  - Refactoring Legacy Code

We are

Helping people to create high quality web applications.

`http://qafoo.com`

- Trainings, Workshops and Consulting
- We help teams rescue legacy applications
- Twitter `@qafoo`

Outline

Refactoring

Testing changes

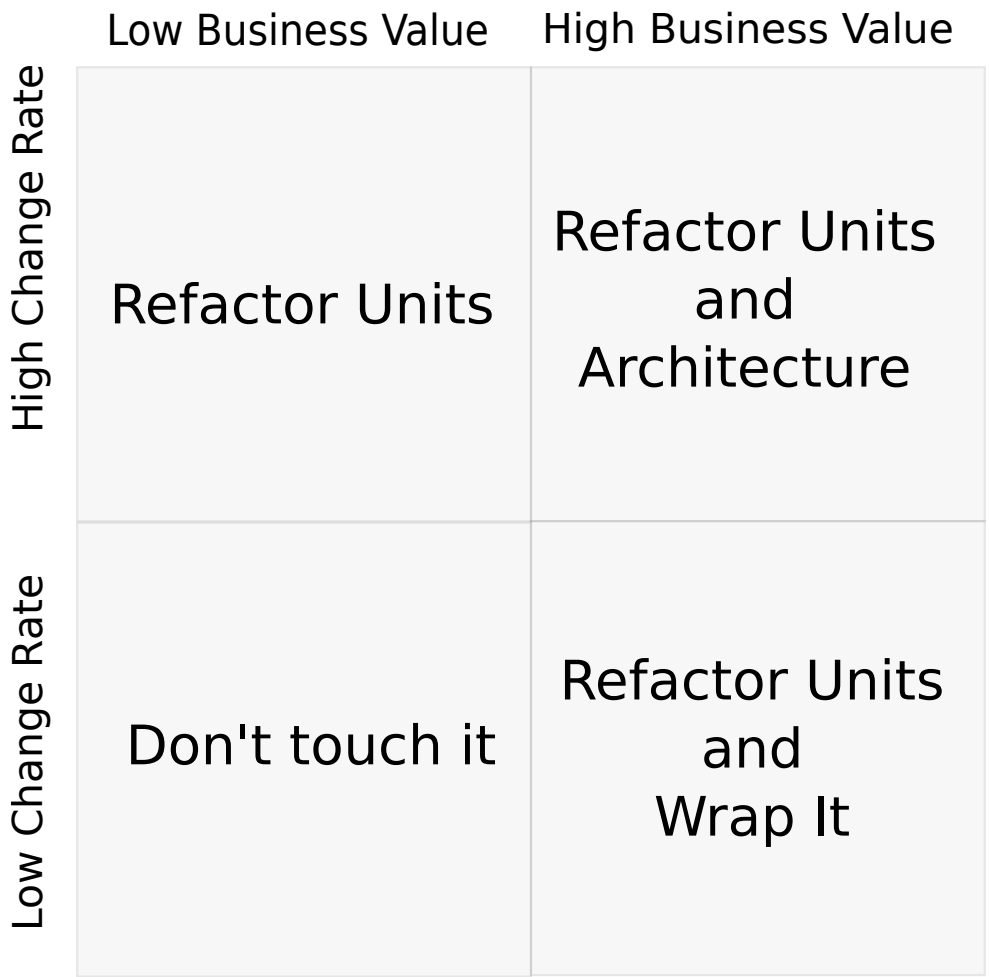
Issues with Legacy Code

What is Refactoring?

Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior. *(*Martin Fowler, refactoring.com*)*

- Internal structures are:
  - functions
  - classes
  - components
  - applications

Where to start Refactoring?



Outline

Refactoring

Testing changes

Issues with Legacy Code

Testing Legacy Software

- Start with an inverted test-pyramid
  - Many integration tests
  - Few unit-tests (no units exist)
- Slowly move towards a more healthy test-mix
- Use shared fixtures and append data
  - Isolated tests are too complex in legacy code
  - Use counters to create isolated records if necessary
- Blackbox-testing much easier than Unit-testing
  - Browser emulation with Behat + Mink

Outline

Refactoring

Testing changes

Issues with Legacy Code

Issues with Legacy Code


Code Issues

Inversion Of Control



Data Structures

Application / Business Logic



Shared State

Issues with Legacy Code: Code Issues

Code-Size

High Efferent Coupling

Code Issues

Inversion Of Control

Code Duplication

Complexity

Data Structures

Application / Business Logic

Shared State

Code Issues

- Easy to locate (Size & Complexity):

```
$ phpmd src/main/ text codesize

Review/Analyzer/PDepend/Model.php:165
    The method getAnnotations() has a Cyclomatic Complexity of 12. The configured
        cyclomatic complexity threshold is 10.
Review/Controller/Source.php:225
    The method toArray() has an NPath complexity of 251. The configured NPath complexity
        threshold is 200.
Review/DIC/Base.php:55
    The method initialize() has 142 lines of code. Current threshold is set to 100. Avoid
         really long methods.
Review/MySQLi.php:34
    The method __construct() has an NPath complexity of 31250. The configured NPath
        complexity threshold is 200.
```

Code Issues

- Easy to locate (Duplication):

```
$ phpcpd src/main/
phpcpd 1.4.1 by Sebastian Bergmann.

0.00% duplicated lines out of 5162 total lines of code.

Time: 0 seconds, Memory: 5.75Mb
```

Code Issues

- Easy to locate (Efferent Coupling):

```
1  $ pdepend --summary-xml=summary.xml src/main/
```

Class	Efferent Coupling
Base	22.00
PDepend	18.00
CodeSniffer	15.00
PhpAnalyzer	12.00
Phplint	12.00
Phpmd	12.00
UML	11.00
Phpcpd	11.00

Code Issues: Refactoring

- Extract method
  - Tool support: OK
  - Try to extract pure method (no side effects on object)
- Extract class (Seperation Of Concerns)
  - Tool support: Bad
  - Usually introduces abstractions (Dependency Inversion)

Extract Method

```php
<?php
class SearchController
{
    public function searchAction(Request $req)
    {
        if ($req->has('q')) {
            $solarium = new SolariumClient('localhost:8080');
            $select = $solarium->createSelect();

            // configure dismax
            $dismax = $select->getDisMax();
            $dismax->setQueryFields(array('name^2', 'description'));
            $dismax->setPhraseFields(array('description'));
            $dismax->setMinimumMatch(1);
            $dismax->setQueryParser('edismax');

            if ($req->has('q')) {
                $escapedQuery = $select->getHelper()->escapeTerm($req->get('q'));
                $select->setQuery($escapedQuery);
            }

            $paginator = new Pagerfanta(new SolariumAdapter($solarium, $select));
            $paginator->setMaxPerPage(15);
            $paginator->setCurrentPage($req->get('page', 1), false, true);

            // ... here be 100 lines
        }
    }
```

Extract Method

```php
<?php

class SearchController
{
    public function searchAction(Request $req)
    {
        if ($req->has('q')) {
            $query= $req->get('q');
            $products = $this->findProducts($query);
        }
    }

    protected function findProducts($query)
    {
        // ... here be 100 lines
    }
```

Extract Class

```php
<?php

class SearchController
{
    public function searchAction(Request $req)
    {
        if ($req->has('q')) {
            $query= $req->get('q');
            $finder = new ProductFinder();
            $products = $finder->search($query);
        }
    }
}
```

Extract Class

```
1  <?php
2
3  class ProductFinder
4  {
5      public function search($query)
6      {
7          // ... here be 100 lines
8      }
9  }
```

Issues with Legacy Code: Inversion Of Control

Code Issues

Missing Abstractions

Inline new

Inversion Of Control

static Calls

Inline Life Cycle Management

Data Structures

Application / Business Logic

Shared State

Inversion of Control

- Inline `new`
  - Only for “newables” – use Dependency Injection for everything else.
- `static` calls
  - **Never** use `static` – use Dependency Injection
- Object Life Cycle Management
  - Let the DIC / user construct objects
  - Constructors (object dependencies) are implementation specific
  - **Never** pass class names, pass objects
- Missing abstractions
  - “Details should depend upon abstractions.” – Uncle Bob

Inversion of Control: Refactoring

- Object graph construction is application configuration
  - It's a different concern!
  - Extract Class
  - Extract Package

Factories: Start with static

```
1   <?php
2
3   class SearchController
4   {
5       public function searchAction(Request $req)
6       {
7           if ($req->has('q')) {
8               $query= $req->get('q');
9               $finder = AppFactory::createProductFinder();
10              $products = $finder->search($query);
11          }
12      }
13  }
```

Factories: Move new to Factory

```php
<?php

class AppFactory
{
    static public function createProductFinder()
    {
        return new ProductFinder();
    }
}
```

Factories: Use Dependency Injection

```php
<?php

class AppFactory
{
    static public function createProductFinder()
    {
        return new ProductFinder(
            self::createSolariumClient()
        );
    }

    static public function createSolariumClient()
    {
        return new SolariumClient('localhost:8080');
    }
}
```

Factories: Dependency Injection

```php
<?php

class ProductFinder
{
    private $solarium;

    public function __construct(SolariumClient $solarium)
    {
        $this->solarium = $solarium;
    }

    public function search($query)
    {
        // ...
    }
}
```

Factories: Repeat

```php
<?php

class AppFactory
{
    static public function createSearchController()
    {
        return new SearchController(
            self::createProductFinder()
        );
    }

    // ...
}
```

Factories: End with Testable Code

```php
<?php
class SearchController
{
    private $finder;
    public function __construct($finder)
    {
        $this->finder = $finder;
    }

    public function searchAction(Request $req)
    {
        if ($req->has('q')) {
            $q = $req->get('q');
            $products = $this->finder->search($q);
        }
    }
```

Issues with Legacy Code: Data Structures

Code
Issues

Inversion
Of Control

Not Immutable

Hash Maps

Data
Structures

Application /
Business Logic

No Logic
Scoping

Shared
State

Data Structures

- Hash Maps (array)
  - Really bad documentation (what properties?, what values?)
  - No errors when setting wrong properties / values
- Not immutable
  - Value changes have side effects in other application parts
  - Horrible to track down
- (Different) logic coupled with models
  - Big models
  - Seperation of concerns (roles / contextes) on model logic

Data Structure: Refactoring

- Rename method / class / package
- Introduce data objects

Introduce Data Objects

```php
<?php

class SearchResult extends Struct
{
    public $products = array();
    public $total = 0;
}

class Product extends Struct
{
    public $title;
    public $price;
}
```

Introduce Data Objects

```php
<?php

abstract class Struct
{
    public function __construct(array $data)
    {
        foreach ($data as $k => $v) {
            if (!property_exists($this, $k)) {
                throw new Exception;
            }

            $this->$k = $v;
        }
    }
}
```

Issues with Legacy Code: Mixing Logic

Code Issues

Inversion Of Control

Data Structures

Business Logic In Controllers

No Workflow Abstraction

Application / Business Logic

No Event Handling

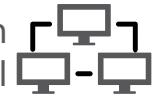
Shared State

Mixing Logic

Mixed business logic (with application logic) in large controllers is the most common and pressing issue with legacy code.

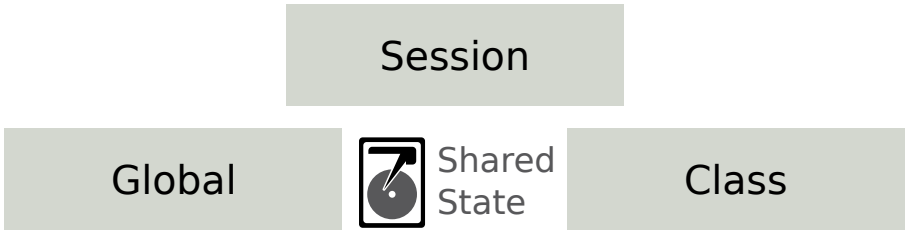
Issues with Legacy Code: Shared State


Code Issues

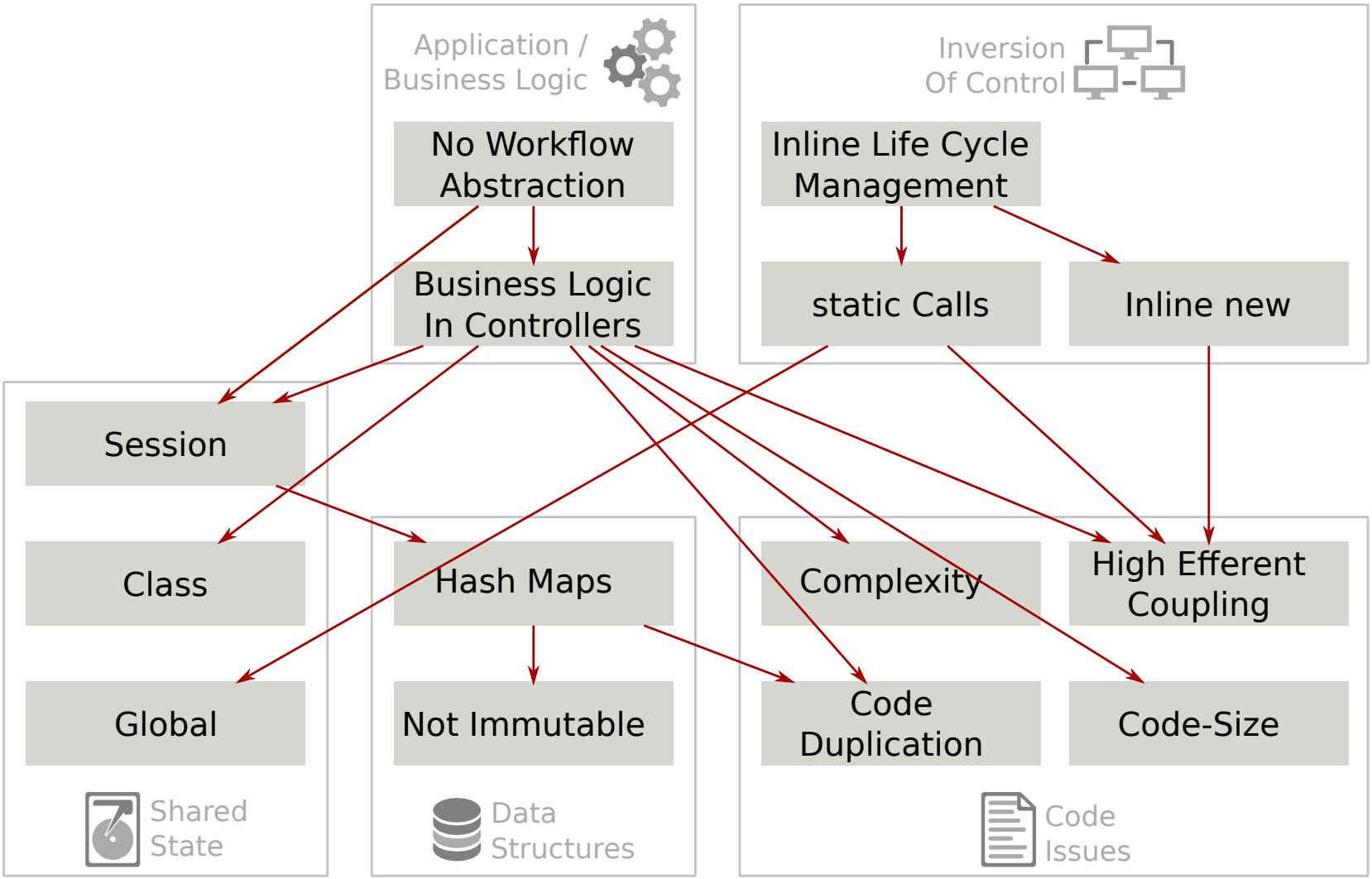

Inversion Of Control


Data Structures


Application / Business Logic



Summary (1)



Summary (2)

- One step at a time
- Test your primary use cases
- Be bold (enough)

Resources

- `http://qafoo.com`
- Book: Refactoring (Martin Fowler)
- Book: Working Effectively with Legacy Code (Michael Feathers)
- Book: Refactoring to Patterns (Joshua Kerievsky)

THANK YOU

Rent a quality expert
qafoo.com