

Fixing Legacy Code

FrOSCon 2013

Benjamin Eberlei @beberlei &
Kore Nordmann @koredn
August 24, 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

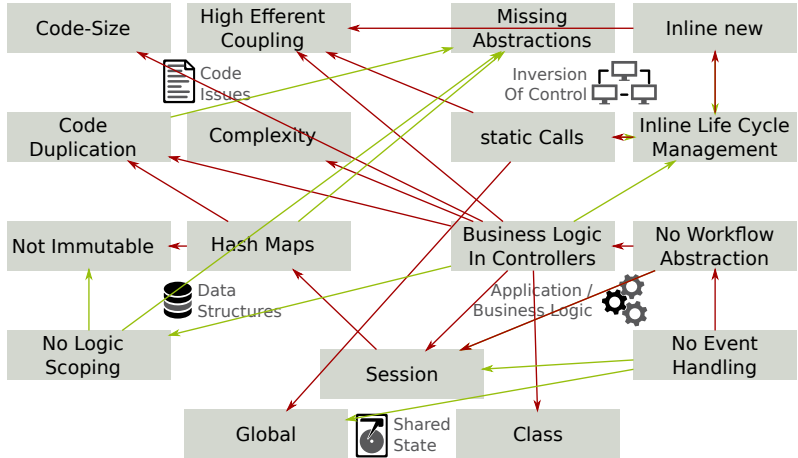
Outline

Issues with Legacy Code

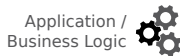
Testing changes

Issues with Legacy Code (2)

Issues with Legacy Code



Issues with Legacy Code



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):

```
1 $ phpm d src/main/ text codesize
2
3 Review/Analyzer/PDepend/Model.php:165
4     The method getAnnotations() has a Cyclomatic Complexity of 12. The configured
5     cyclomatic complexity threshold is 10.
6 Review/Controller/Source.php:225
7     The method toArray() has an NPath complexity of 251. The configured NPath complexity
8     threshold is 200.
9 Review/DIC/Base.php:55
10    The method initialize() has 142 lines of code. Current threshold is set to 100. Avoid
11    really long methods.
12 Review/MySqlLi.php:34
13    The method __construct() has an NPath complexity of 31250. The configured NPath
14    complexity threshold is 200.
```


Code Issues

► Easy to locate (Duplication):

```
1 $ phpcpd src/main/  
2 phpcpd 1.4.1 by Sebastian Bergmann.  
3  
4 0.00% duplicated lines out of 5162 total lines of code.  
5  
6 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)

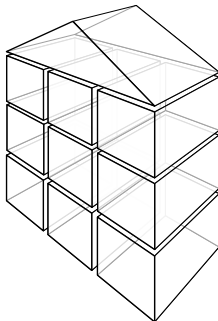
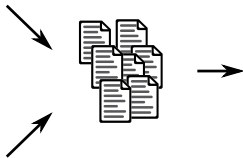
Outline

Issues with Legacy Code

Testing changes

Issues with Legacy Code (2)

Testing Legacy Software



```
"GET /blog/016_struct_classes_in_php.rss HTTP/1.1" 200
"GET /blog.rss HTTP/1.1" 304
"GET /blog.rss HTTP/1.1" 304
"GET /images/clients/sixclicks.png HTTP/1.1" 200
"GET /blog.rss HTTP/1.1" 304
"GET /services/training/topics/build_... HTTP/1.1" 302
"GET /services/training/topics/build_...html HTTP/1.1" 200
"HEAD / HTTP/1.1" 200
"GET /blog.rss HTTP/1.1" 304
"GET /blog.rss HTTP/1.1" 304
```

Behat

- ▶ BDD test framework for PHP
- ▶ Inspired by Ruby's Cucumber
- ▶ Work with Gherkin language framework
- ▶ <http://behat.org>

- ▶ Web acceptance test framework
- ▶ Abstracts browser emulations / controllers
 - ▶ Fast: Goutte
 - ▶ JavaScript: Sahi
 - ▶ Others: Zombie.js, Selenium, Selenium 2
- ▶ <http://mink.behat.org>

Behat Mink Example

1 **Feature:** Browse Wikipedia

2
3 **Scenario:** Search front page

4 **Given** I am on "/"

5 **When** I fill in "searchInput" with "Kore"

6 **And** I press "searchButton"

7 **Then** I should see "Kore may refer to:"

8
9 **Scenario:** Follow redirect link

10 **Given** I am on "/"

11 **When** I fill in "searchInput" with "Kore"

12 **And** I press "searchButton"

13 **And** I follow "Kore (energy drink)"

14 **Then** the response status code should be 200

Mink Behat-extension

- ▶ Mink integration for Behat
- ▶ Pre-build sentences to browse web applications
- ▶ Extensible with custom sentences
 - ▶ Use to access internals of your (legacy) application
 - ▶ Use to create data base fixtures
- ▶ <http://extensions.behat.org/mink>
- ▶ Use code coverage to check for uncovered features: http://qafoo.com/blog/040_code_coverage_with_behat.html

Outline

Issues with Legacy Code

Testing changes

Issues with Legacy Code (2)

Issues with Legacy Code: Inversion Of Control



Missing Abstractions


Inline new

Inversion Of Control 

static Calls

Inline Life Cycle Management



Application / Business Logic 



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

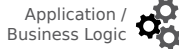
- ▶ Object graph construction is application configuration
 - ▶ It's a different concern!
 - ▶ Extract Class
 - ▶ Extract Package

Issues with Legacy Code: Data Structures



Not Immutable

Hash Maps



No Logic
Scoping



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
 - ▶ Separation of concerns (roles / contextes) on model logic

Data Structure: Refactoring

- ▶ Rename method / class / package
- ▶ Introduce value objects
- ▶ Branch by abstraction
 1. Introduce facade / proxy for old code
 2. Call old code from facade / proxy implementation
 3. Write new shiny facade / proxy implementation
 4. Use new implementation
 5. Delete old code

Issues with Legacy Code: Mixing Logic



Business Logic
In Controllers

No Workflow
Abstraction

Application /
Business Logic



No Event
Handling

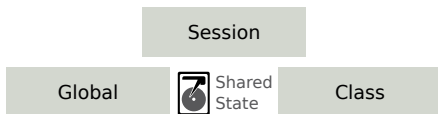
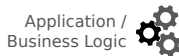


Mixed business logic (with application logic) in large controllers is the most common and pressing issue with legacy code.

Mixing Logic: Refactoring

- ▶ Extract method / class / package
- ▶ Extract business logic: Service Layer Pattern
- ▶ Introduce (domain) events
- ▶ Introduce a workflow abstraction?

Issues with Legacy Code: Shared State



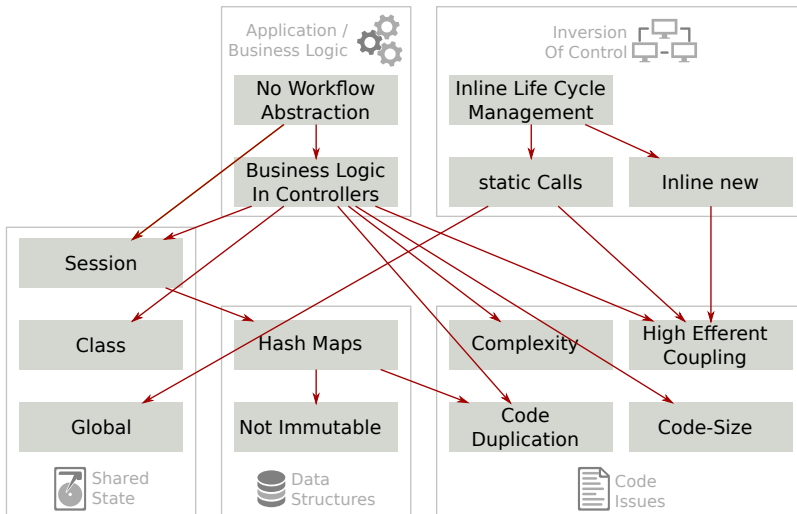
Shared State

- ▶ We **know** that global state is bad.
- ▶ Sessions are (often) cross-request global state in hash maps
- ▶ Method execution pathes depending on current object state are horrible to track down

Shared State: Refactoring

- ▶ Extract **pure** functions (no side effects: object, global, session)
- ▶ Remove side effects from classes
- ▶ Make necessary side effects as explicit as possible (state, output, ...)

Summary (1)



Summary (2)

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



THANK YOU

Rent a quality expert
qafoo.com