

---

# Testable Code

## Symfony Live Berlin

Kore Nordmann (@koredn)  
Tobias Schlitt (@tobySen)

November 22, 2012

# About us

---

- ▶ Kore Nordmann
  - ▶ [kore@qafoo.com](mailto:kore@qafoo.com)
  - ▶ [@koredn](#)
  
  - ▶ Degree in computer science
  - ▶ Professional PHP since 2000
  - ▶ Open source enthusiast
- ▶ Tobias (Toby) Schlitt
  - ▶ [toby@qafoo.com](mailto:toby@qafoo.com)
  - ▶ [@tobySen](#)

## Co-founders of

---



**Helping people to create high quality web applications.**

<http://qafoo.com>

- ▶ Expert consulting
- ▶ Individual training

From 2013 on incorporating Doctrine 2 & Symfony2 expertise!

# Task: Who are you?

---

- ▶ Get in touch with your seat neighbor
  - ▶ Who is he?
  - ▶ What is his background?
  - ▶ **What does he expect to learn here?**
- ▶ 2 minutes time
- ▶ Introduce your neighbor to the audience

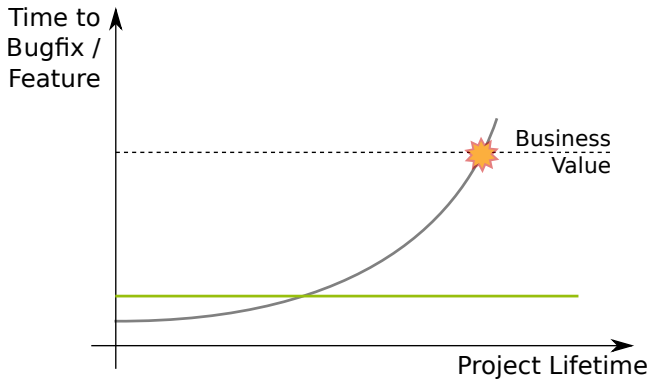
---

# Part I

# Testing

# Why Test?

---



# Outline

---

Types

Unit tests

Example

# Ways of testing

---

- ▶ **Automatic** vs. manual
- ▶ **Developer** vs. tester
- ▶ **Internal** vs. external
- ▶ **Back end** vs. front end
- ▶ **Code** vs. appearance
- ▶ **Functional** vs. non-functional
- ▶ **Dynamic** vs. static



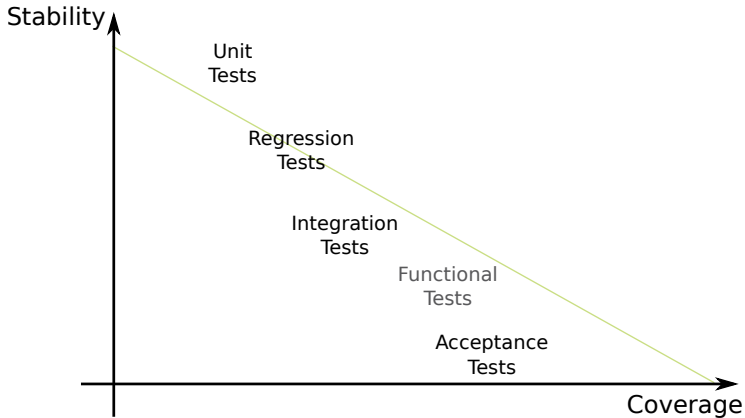
# Test methods

---

- ▶ Unit tests
- ▶ Integration tests
- ▶ Regression tests
- ▶ Acceptance tests

# Test Stability

---



# Outline

---

Types

**Unit tests**

Example

# Unit tests

---

- ▶ Purpose
  - ▶ Validate functionality
  - ▶ Test a single unit of code
  - ▶ Avoid regressions
- ▶ Applications
  - ▶ Verify interfaces (public API)
  - ▶ Test bugs dedicatedly
- ▶ Benefits
  - ▶ Force code modularization
  - ▶ Ensures backwards compability
  - ▶ Migrate safely

# Test Driven Development (TDD)

---

- ▶ Test Driven Development
  - ▶ 1) Write (& document) interfaces
  - ▶ 2) Write tests
  - ▶ 3) Write implementation
- ▶ Benefits
  - ▶ A lot less defects in code
  - ▶ Faster development after a couple of projects
  - ▶ More developer satisfaction
  - ▶ Less code

# Consequences

---

- ▶ Consequences of unit testing
  - ▶ Testing of units requires replacing dependencies
  - ▶ Stable well-designed API
  - ▶ Ensures backwards compability

# Outline

---

Types

Unit tests

**Example**

# Example

---

Developing a weather service



# Requirements

---

- ▶ Fetch weather for a city
- ▶ Relevant data:
  - ▶ Condition
  - ▶ Temperature
  - ▶ Wind
- ▶ Be service-agnostic
  - ▶ Weather service come and go
  - ▶ Data licenses may change
- ▶ Log service failures
- ▶ Make it possible to add service fallbacks later

# Class diagram

---

## **Loader**

```
+__construct(Service, Logger)  
+addFallbackService(Service)  
+getWeatherForLocation(Location)
```

+get

# Task: What tests do you want?

---

- ▶ Group with 3-4 people
- ▶ Discuss:
  - ▶ What types of tests do you have in your projects?
  - ▶ What types of tests do you desire for the future?
- ▶ 5 minutes time
- ▶ Collect the most common answers

---

## Part II

# Testable Code

# Outline

---

Testing issues

Conclusion

# The Example

```
1 <?php
2
3 class WeatherLoader
4 {
5     public function getWeatherForLocation( Location $location )
6     {
7         $xml = $this->fetchData( $location->city );
8         Logger::logDebug( 'Fetched XML', $xml );
9         return $this->parseData( $xml );
10    }
11    protected function fetchData( $city )
12    {
13        $url = sprintf( 'http://...? city=%s', $city );
14        return $this->fetchFromUrl( $url );
15    }
16    protected function parseData( $xml )
17    {
18        $weather = new Weather();
19        $weather->conditions = $this->parseConditions( $xml );
20        $weather->windSpeed = $this->milesToKilometers(
21            $this->parseWindSpeed( $xml )
22        );
23        return $weather;
24    }
25    /* ... */
26 }
```

# Issue #1

```
1 <?php
2
3 class WeatherLoader
4 {
5     public function getWeatherForLocation( Location $location )
6     {
7         $xml = $this->fetchData( $location->city );
8         Logger::logDebug( 'Fetched_XML', $xml );
9         return $this->parseData( $xml );
10    }
11    protected function fetchData( $city )
12    {
13        $url = sprintf( 'http://...? city=%s', $city );
14        return $this->fetchFromUrl( $url );
15    }
16    protected function parseData( $xml )
17    {
18        $weather = new Weather();
19        $weather->conditions = $this->parseConditions( $xml );
20        $weather->windSpeed = $this->milesToKilometers(
21            $this->parseWindSpeed( $xml )
22        );
23        return $weather;
24    }
25    /* ... */
26 }
```

# Protected to Public

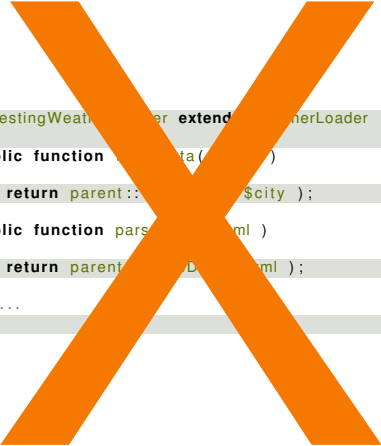
```
1 <?php
2
3 class Weather
4 {
5     public function WeatherForecast ( Location $location )
6     {
7         $xml = $this->fetchData( $location->city );
8         Logger::logDebug( 'Fetch XML', $xml );
9         return $this->parseForecast( $xml );
10    }
11    public function fetchData ( $city )
12    {
13        $url = sprintf( 'http://www.weather.gov/?city=%s', $city );
14        return $this->fetch( $url );
15    }
16    public function parseForecastData( $xml )
17    {
18        $weather = new Weather();
19        $weather->conditions = $this->parseConditions( $xml );
20        $weather->windSpeed = $this->parseWindSpeed( $xml );
21    }
22    private function parseConditions( $xml )
23    {
24        // ...
25    }
26 }
```



# Mocking the Subject

---

```
1 <?php
2
3 class TestingWeatherer extends WeatherLoader
4 {
5     public function getWeatherData( $city )
6     {
7         return parent::getWeatherData( $city );
8     }
9     public function parseWeatherData( $xml )
10    {
11        return parent::parseWeatherData( $xml );
12    }
13    // ...
14 }
```



# Protected to Public

---

- ▶ Exposed functionality will be used
- ▶ Creates public API that is hard to change
- ▶ Internal dependencies might break

# The Real Issue

---

E\_TOO\_MANY\_RESPONSIBILITIES

# The Fix

---

```
1 <?php
2
3 class WeatherLoader
4 {
5     public function __construct( WeatherService $service , WeatherParser $parser )
6     {
7         // ...
8     }
9     public function getWeatherForLocation( Location $location )
10    {
11        $data = $this->service->getWeather( $location );
12        Logger::logDebug( 'Fetched data', $data );
13        return $this->parser->parseData( $data );
14    }
15 }
```

# The Fix

---

- ▶ Never test private/protected explicitly
- ▶ Test them implicitly ...
- ▶ ...or change the code

# Issue #2

---

```
1 <?php
2
3 class WeatherLoader
4 {
5     public function __construct( WeatherService $service, WeatherParser $parser )
6     {
7         // ...
8     }
9     public function getWeatherForLocation( Location $location )
10    {
11        $data = $this->service->getWeather( $location );
12        Logger::logDebug( 'Fetched data', $data );
13        return $this->parser->parseData( $data );
14    }
15 }
```

# Test Code in Production

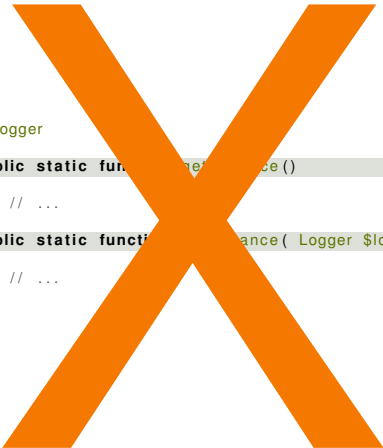
---



```
1 <?php
2
3 class Logger
4 {
5     public static function log( $message, $data )
6     {
7         // ...
8     }
9     public static function forTesting ()
10    {
11        // ...
12    }
13 }
```

# Test Code in Production - continued

---



```
1 <?php
2
3 class Logger
4 {
5     public static function getInstance ()
6     {
7         // ...
8     }
9     public static function getInstance ( Logger $logger )
10    {
11        // ...
12    }
13 }
```



# The Real Issue

---

E\_STATIC\_DEPENDENCY

# The Fix

---

```
1 <?php
2
3 class WeatherLoader
4 {
5     public function __construct(
6         WeatherService $service ,
7         WeatherParser $parser
8         Logger $logger )
9     {
10         // ...
11     }
12     public function getWeatherForLocation( Location $location )
13     {
14         $data = $this->service->getWeather( $location ) ;
15         $this->logger->logDebug( 'Fetched_data', $data ) ;
16         return $this->parser->parseData( $data ) ;
17     }
18 }
```

# The Fix

---

- ▶ Never use static access
- ▶ Always inject dependencies
- ▶ Maybe use a dependency injection container (DIC)

# Issue #3

---

```
1 <?php
2
3 class WeatherService
4 {
5     public function __construct( AppRegistry $registry )
6     {
7         // ...
8     }
9     public function getWeather( Location $location )
10    {
11        $httpClient = $this->appRegistry->get( 'http_client' );
12        $url = sprintf( 'http://...?city=%s', $city );
13        return $httpClient->get( $url );
14    }
15 }
```

# Mocking to Mock

```
1 <?php
2
3 class WeatherServiceTest extends PHPUnit\Framework\TestCase
4 {
5     public function testGetWeather()
6     {
7         $httpClientMock = $this->getMock( 'HttpClient' );
8         $httpClientMock->expects( $this->once() )
9             ->method( 'get' );
10         /* ... */
11
12         $appRegistryMock = $this->getMock( 'AppRegistry' );
13         $appRegistryMock->expects( $this->once() )
14             ->method( 'getWeatherService' );
15         /* ... */
16
17         $service = new WeatherService( $appRegistryMock );
18         $this->assertEquals(
19             'OK',
20             $service->getWeather( new Request() ) );
21     };
22 }
23 }
```

# Using Productive Code in Tests

```
1 <?php
2
3 class WeatherTest extends PHPUnit_Framework_TestCase
4 {
5     public function testGetWeather()
6     {
7         $httpClientMock = $this->getMock( 'HttpClient' );
8         $httpClientMock->expects( $this->once() )
9             ->method(
10                 /* ... */;
11
12         $appRegistry = new AppRegistry();
13         $appRegistry->register( 'HttpClient', $httpClientMock );
14
15         $service = new WeatherService( $appRegistry );
16         $this->assertEquals(
17             /* ... */;
18             $service->getWeather( new Location() )
19         );
20     }
21 }
```

# The Real Issue

---

E\_REACHING\_THROUGH\_OBJECTS

# The Fix

---

```
1 <?php
2
3 class WeatherService
4 {
5     public function __construct( HttpClient $httpClient )
6     {
7         // ...
8     }
9     public function getWeather( Location $location )
10    {
11        $url = sprintf( 'http://...? city=%s', $city );
12        return $this->httpClient->get( $url );
13    }
14 }
```



# The Fix

---

- ▶ Do not pull dependencies . . .
- ▶ . . . push them
- ▶ Do not reach through objects

# Issue #4

---

```
1 <?php
2
3 class Logger
4 {
5     public function __construct( $fileName )
6     {
7         // ... error checks ...
8         $this->fileHandle = fopen( $fileName, 'a' );
9     }
10    public function logDebug( $message, $data )
11    {
12        fwrite(
13            $this->fileHandle ,
14            sprintf(
15                "%s_(%s)\n",
16                $message,
17                $data
18            )
19        );
20    }
21 }
```

# Accessing File System in Tests

---

```
1 <?php
2
3 class LoggerTest extends PHPUnit_Framework_TestCase
4 {
5     public function setUp()
6     {
7         $tmpLogFile = tempnam( sys_get_temp_dir(), 'logfile' );
8
9         $logger = new Logger( $tmpLogFile );
10        $logger->logDebug( 'message.', 'with_data' );
11
12        $this->assertExpectedFileContents(
13            "Some_message\n",
14            file_get_contents( $tmpLogFile )
15        );
16        unlink( $tmpLogFile );
17    }
18 }
```



# Accessing File System in Tests

---

- ▶ No file access in unit tests (slow!)
- ▶ Maintaining temporary files sucks
  - ▶ Creating
  - ▶ Cleanup
  - ▶ System differences

# The Virtual File System

```
1 <?php
2
3 class LoggerTest extends PHPUnit_Framework_TestCase
4 {
5     public function testLogDebugSuccess()
6     {
7         vfsStream::create( 'test' );
8         $logFile = vfsStream::url( 'test' ) . '/message.log';
9
10        $logger = new Logger( $logFile );
11        $logger->logDebug( 'Some message.', 'with_data' );
12
13        $this->assertThat(
14            vfsStream::url( 'test' )->hasChild( 'message.log' )
15        );
16        $this->assertThat(
17            "Some message. (with_data)",
18            file_get_contents( $logFile )
19        );
20    }
21 }
```

# The Virtual File System

---

- ▶ Works, but ...

# The Real Issue

---

E\_HARD\_SYSTEM\_DEPENDENCY

# The Fix

---

```
1 <?php
2
3 class Logger
4 {
5     public function __construct( FileHandler $fileHandler )
6     {
7         $this->fileHandler = $fileHandler;
8     }
9     public function logDebug( $message, $data )
10    {
11        $this->fileHandler->write(
12            sprintf(
13                "%s_(%s)\n",
14                $message,
15                $data
16            )
17        );
18    }
19 }
```



# The Fix

---

- ▶ Abstract system dependencies ...
- ▶ ... as low as possible

# Outline

---

Testing issues

Conclusion

# What have we seen?

---

- ▶ Single Responsibility Principle
- ▶ Open Close Principle
- ▶ Law of Demeter
- ▶ Dependency Inversion Principle

# Conclusion

---

Testable Code  
↕  
Good OOD

# SOLID

---

- S Single Responsibility Principle
- O Open / Close Principle
- L Liskov Substitution Principle
- I Interface Segregation Principle
- D Dependency Inversion Principle

# Task: Define a Component to be Designed

---

- ▶ Full audience
- ▶ Define a component to design
  - ▶ Not too big for today
  - ▶ Not too small to be trivial
- ▶ **Ideas?**

# Task: Design the Component

---

- ▶ Group with 3-4 people
- ▶ For the component we just defined
- ▶ Create an Object Oriented Design (OOD)
  - ▶ Name classes
  - ▶ Method stubs
  - ▶ Draw fancy pictures
- ▶ Pay attention to
  - ▶ SOLID
  - ▶ Law of Demeter
  - ▶ Testability

---

# Part III

# Metrics



# Outline

---

## What are metrics?

Classic software metrics

Object oriented software metrics

Conclusion

# Software metrics

---

- ▶ A software metrics is a measure for a quality aspect of object oriented software
  - ▶ “A software metric is a measure of some property of a piece of software or its specifications” (Wikipedia)
  - ▶ “You cannot control what you cannot measure.” (Tom DeMarco)
    - ▶ Has been relativized by now.

# Outline

---

What are metrics?

**Classic software metrics**

Object oriented software metrics

Conclusion

# Scale metrics

---

- ▶ Sums over software artifacts

- ▶ Lines Of \*

- LOC Lines Of Code

- ELOC Executable Lines Of Code

- CLOC Comment Lines Of Code

- NCLOC Non-Comment Lines Of Code

- ▶ Number Of \*

- NOC Number Of Classes

- NOM Number Of Methods

- NOP Number Of Packages

# Lines Of \*, Number Of \*

---

```
1 <?php
2 namespace foo\bar;
3
4 abstract class FooBar {
5     abstract function bar();
6 }
7
8 class Foo extends FooBar {
9     /* Does this ... */
10    public function bar() {}
11    /* Does that ... */
12    public function baz() {}
13 }
14
15 class Bar extends Foo {
16    public function foo(Foo $f) {}
17 }
```

## ► Lines Of \*

LOC	16
ELOC	3
CLOC	2
NCLOC	14

## ► Number Of \*

NOC	3
NOM	4
NOP	1

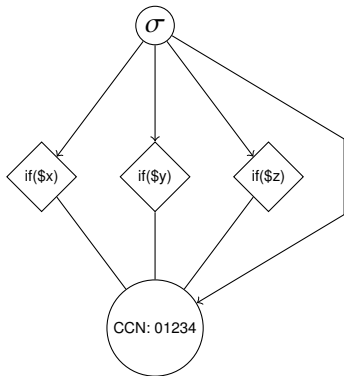
# Complexity metrics

---

- ▶ Control structures are the key point to complexity
  - ▶ if, elseif, for, while, foreach, catch, case, xor, and, or, &&, ||, ?:
- ▶ Cyclomatic Complexity (CCN)
  - ▶ Number of *branches*
- ▶ NPath Complexity
  - ▶ Number of *execution paths*
  - ▶ Minds the structure of blocks

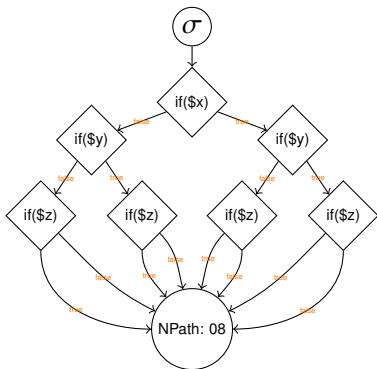
# Cyclomatic Complexity

```
1 <?php
2 class Foo {
3     public function foo() {
4         if ($x) { }
5         if ($y) { }
6         if ($z) { }
7         return $x;
8     }
9 }
```



# NPath Complexity

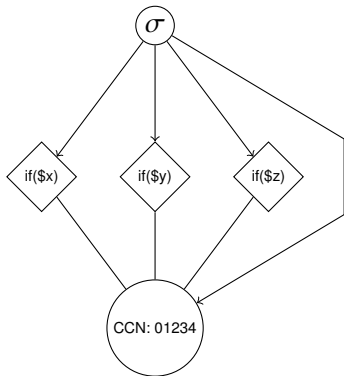
```
1 <?php
2 class Foo {
3     public function foo() {
4         if ($x) { }
5         if ($y) { }
6         if ($z) { }
7         return $x;
8     }
9 }
```





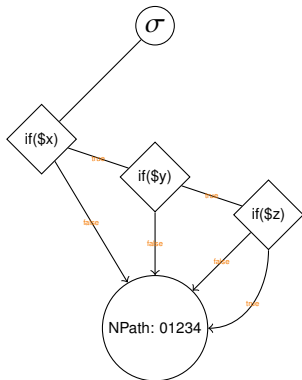
# Cyclomatic Complexity

```
1 <?php
2 class Foo {
3     public function foo() {
4         if ($x) {
5             if ($y) {
6                 if ($z) { }
7             }
8         }
9         return $x;
10    }
11 }
```



# NPath Complexity

```
1 <?php
2 class Foo {
3     public function foo() {
4         if ($x) {
5             if ($y) {
6                 if ($z) { }
7             }
8         }
9         return $x;
10    }
11 }
```



# Sensible limits

---

- ▶ Numbers do not tell anything by themselves
- ▶ To judge you need limiting values
  - ▶ Cyclomatic Complexity
    - ▶ 1-4: low, 5-7: medium, 8-10: high, 11+: hell
  - ▶ NPath Complexity
    - ▶ 200: critical mass
- ▶ Limiting values are at your discretion

# Combine metrics

---

Combined metrics allow deep insight in complex products

- ▶ LOC: 300; CCN: 42; NOC: 5; NOM: 15
  - ▶  $CCN / LOC = 0,14$ 
    - ▶ Every sixth line is a control structure
  - ▶  $LOC / NOC = 60$ 
    - ▶ Procedural code or big classes
  - ▶  $LOC / NOM = 20$ 
    - ▶ Big methods or procedural code
  - ▶  $CCN / NOM = 2,8$ 
    - ▶ Highly complex methods

# Combine metrics: CRAP

---

Is your code CRAP?

$$CRAP(m) = \begin{cases} ccn(m)^2 + ccn(m), & \text{if } cov(m) = 0 \\ ccn(m), & \text{if } cov(m) \geq .95 \\ ccn(m)^2 * (1 - cov(m))^3 + ccn(m), & \text{else} \end{cases}$$

- Change Risk Anti Patterns

## Task: Analyze your source with PDepend

---

- ▶ Group with 3-4 people
- ▶ Install PDepend
  - ▶ <http://pdepend.org/>
- ▶ Analyze your code
  - ▶ What are the top / bottom methods?
  - ▶ Did you expect the complexity?
  - ▶ How could you attempt to solve it?
- ▶ How to?
  - ▶ `$ pdepend --summary=summary.xml srcDir/`
  - ▶ Helpful script: <http://stuff.qafoo.com/pListTop.txt>

# Outline

---

What are metrics?

Classic software metrics

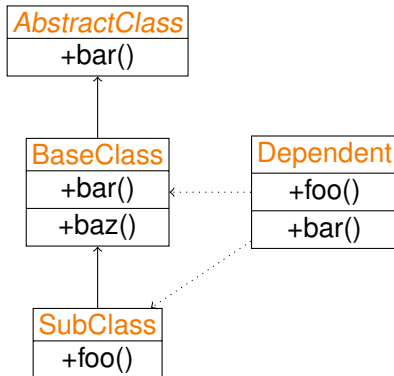
**Object oriented software metrics**

Conclusion

- ▶ A Metrics Suite for Object Oriented Design
  - ▶ Weighted Methods per Class (WMC)
    - ▶ Sum of method complexities
    - ▶ Limiting value: 20 - 50
  - ▶ Number Of Children (NOC)
    - ▶ Number of class extension
    - ▶ Indicator for wrong use of abstraction / inheritance
  - ▶ Depth of Inheritance Tree (DIT)
    - ▶ Inheritance can increase software complexity
    - ▶ Limiting value:  $\leq 5$
    - ▶ Commonly limited at component boundary



# Object oriented software metrics



- ▶ *AbstractClass*

WMC 0

DIT 0

- ▶ *BaseClass*

WMC 2

DIT 1

- ▶ *SubClass*

WMC 1

DIT 2

- ▶ *Dependent*

WMC 2

DIT 0

# Coupling

---

- ▶ Excessive coupling is one of the key problems in modern Object-Oriented Softwaresystems
- ▶ So what kinds of dependencies do we know?
  - ▶ Artifacts that utilize other artifacts (outgoing dependency)
  - ▶ Artifacts that are used by other parts of the system (incoming dependency)
  - ▶ Poorly designed artifacts do both excessively
- ▶ Dependencies between artifacts are established by:
  - ▶ Object instantiations
  - ▶ Static method calls
  - ▶ Method parameters
  - ▶ Thrown and caught exceptions

# Coupling

---

- ▶ Coupling Between Objects (CBO)
  - ▶ Describes the number of outgoing dependencies
  - ▶ Introduced in Chidamber's & Kemerer's *Metrics Suite for OOD*
  - ▶ Limiting value:  $\leq 14$
  - ▶ Classes and interfaces within the same inheritance hierarchy are no dependencies
  - ▶ Software artifacts with high coupling tend to be error-prone
  - ▶ This metric is also known as Efferent Coupling ( $C_E$ )
    - ▶ Because UncleBob has used that name in his *Design Quality Metrics* paper

# Coupling

---

- ▶ Afferent Coupling ( $C_A$ )
  - ▶ Describes the number of incoming dependencies
  - ▶ High afferent coupling indicates code reuse
  - ▶ But also ~~implies~~ requires stable and well defined APIs
  - ▶ Classes and interfaces within the same inheritance hierarchy are no dependencies

# Instability

---

- ▶ What other coupling based quality aspects can we measure?
- ▶ Idea:
  - ▶ A component becomes more error-prone the higher the coupling to other objects is
    - ▶ We can say, the component reacts more and more instable to external changes
  - ▶ A component with a high afferent coupling has a greater impact
    - ▶ We can say, the component is responsible for the entire systems stability
- ▶ A new metric, which is an indicator for the component's responsibility in the entire system
  - ▶ Instability:

$$I = Efferent / (Efferent + Afferent)$$

- ▶ The range of this metric is [0,1]

# Abstractness

---

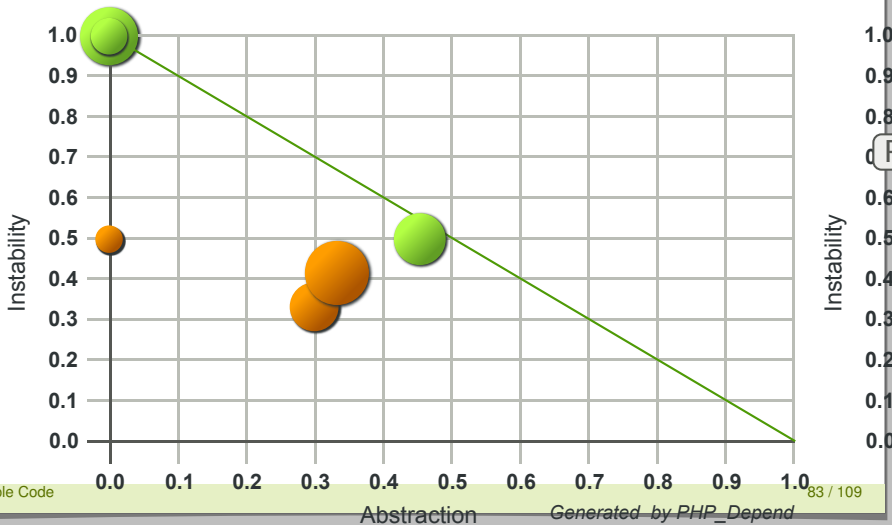
- ▶ Define a formula for abstractness:

- ▶ Abstractness:

$$A = \text{Abstracts} / (\text{Abstracts} + \text{Concretes})$$

- ▶ The range of this metric is [0,1]
- ▶ What stability could we expect for an abstract class or an interface?
  - ▶ An instability of 0, because something totally abstract describes normally an API (no outgoing dependencies)
- ▶ On the other hand we have a 100% concrete component, what stability can we expect here?
  - ▶ Here we can expect an instability of nearly 1 (no incoming dependencies)
- ▶ This means that there is an expected relation between the instability of a component and its abstraction

# Abstractness & Instability



Testable Code

83 / 109

# CodeRank

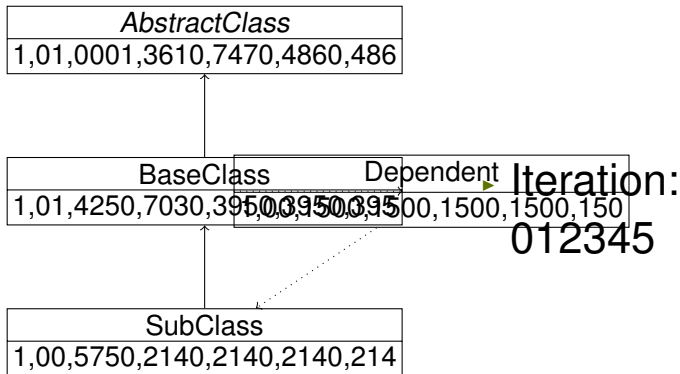
---

- ▶ Based on Googles PageRank™
- ▶ Maps software to a graph
  - ▶ A node ( $\pi$ ) for each software artifact
    - ▶ Package, Class, Method
  - ▶ An edge ( $\rho$ ) for each relation
    - ▶ Inheritance, Call, Parameter, Exceptions
- ▶ CodeRank:

$$CR(\pi_i) = \sum_r r((1 - d) + d \sum_r r(CR(\pi_r)/\rho_r))$$



# CodeRank



# CodeRank

---

- ▶ Incorporates indirect dependencies
- ▶ Locates elements with high effect on the whole system
- ▶ Reverse CodeRank:
  - ▶ Shows dependent components

# Outline

---

What are metrics?

Classic software metrics

Object oriented software metrics

**Conclusion**

# Metrics are ...

---

- ▶ ... no magic, but simple measured values
- ▶ ... useless without limiting values
- ▶ ... scalable – grow with project growth
- ▶ ... reproducible and automatable
- ▶ ... objective – since calculated by software
- ▶ ... highly interpretable – interpretation depends on viewer

# Task: Code Review

---

- ▶ Give us some code
- ▶ Don't be shy

---

# Part IV

## Refactoring

# Refactoring

---

- ▶ Code refactoring is the “disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior”
  - ▶ Change code, but do not break it
  - ▶ (Functional) tests are *really, really* useful during refactoring.
- ▶ Goals
  - ▶ Increase maintainability (reduce complexity)
  - ▶ Increase testability
  - ▶ Increase re-usability

# Common techniques

---

- ▶ Rename method
  - ▶ Readability / maintainability
- ▶ Extract method
  - ▶ Move reused code into its own methods
  - ▶ Reduces complexity
- ▶ Extract class
  - ▶ Move code segments into its own class / implementation
  - ▶ See: *Separation of Concerns, Interface Segregation Principle*
- ▶ Extract module / component
  - ▶ Make code reusable across projects
  - ▶ See: *Separation of Concerns, Interface Segregation Principle, Open Closed Principle*



# static

---

- ▶ static is the single most pressing issue when it comes to testability
  - ▶ static access
  - ▶ Registries
  - ▶ Singletons

# The Problem

---

- ▶ static access is the problem, not static methods.

```
1 class UserModel {
2     // ...
3     public function login()
4     {
5         // ...
6         Logger::log( "User_{ $this->name }_has_just_logged_in." );
7         // ...
8     }
9     // ...
10 }
```

- ▶ Most common use cases:
  - ▶ Logging
  - ▶ Configuration access
  - ▶ Cache access
  - ▶ Data storage access

# Locating the evil

---

## ► Shell

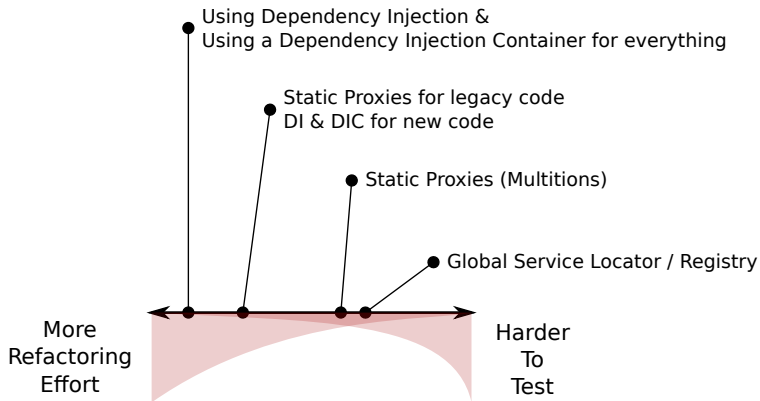
```
1 ack-grep --php '[A-Za-z0-9_]+::' | \  
2   sed -e 's/.*[^A-Za-z0-9_]([A-Za-z0-9_]\+):.*\/\1/' | \  
3   sort | uniq -c | sort -nr
```

# Push vs. Pull

---

- ▶ Pushing dependencies
  - ▶ The dependencies are provided during object construction
  - ▶ Legacy code might implicitly create objects, though.
- ▶ Pulling dependencies
  - ▶ Requesting global variable / Singleton
  - ▶ Requesting from a Single Type Container
    - ▶ Replacing dependencies is possible
  - ▶ Requesting from a Multiple Type Container (Service Locator)
    - ▶ Implicit dependency on the full system
  - ▶ Containers may be injected or accessed statically

# Refactoring Approaches



# Dependency Injection

---

- ▶ *All* dependencies are provided through constructor or setter injection
- ▶ No object construction except in the Dependency Injection Container (DIC)
  - ▶ Except for value objects and exceptions
- ▶ Bootstrap requests objects from DIC
  - ▶ The DIC is not passed to *any* class
    - ▶ Maybe except controllers

# Dependency Injection – Summary

---

- ▶ Benefits
  - ▶ Simple and effective Unit Testing
  - ▶ Exposes S.O.L.I.D. violations
- ▶ Drawbacks
  - ▶ Designing proxies for legacy code is *a lot* of work
  - ▶ Refactoring all code to use dynamic access is *a lot* of work
  - ▶ Solving S.O.L.I.D. violations is additional work

# Static Proxies

---

- ▶ Wrap code into static proxies with replacement option
- ▶ Class names of static calls must be adapted
- ▶ Optionally exchange used code in tests



# Static Proxies – Base Class

```
1  trait Proxy {  
2      use Singleton;  
3  
4      protected static $target;  
5  
6      final public static function getTarget()  
7      {  
8          return isset( static::$target )  
9              ? static::$target  
10             : static::$target = static::getDefaultInstance();  
11      }  
12  
13      protected static function getDefaultInstance()  
14      {  
15          throw new \RuntimeException( "No_default_target_specified." );  
16      }  
17  
18      final public static function setTarget( $target )  
19      {  
20          static::$target = $target;  
21      }  
22  
23      public static function __callStatic( $method, array $parameters )  
24      {  
25          return call_user_func_array( array( self::getTarget(), $method ), $parameters );  
26      }  
27  }
```

# Static Proxies – Extensions

---

```
1 class DebugProxy {
2   use Proxy;
3
4   protected static function getDefaultInstance()
5   {
6     return Debug::getInstance();
7   }
8
9   // Optionally define methods explicitly and proxy calls to make
10  // mocking more obvious.
11 }
12
13 // Debug::log( "Hello World!" );
14 // changes to:
15 DebugProxy::log( "Hello World!" );
```

# Static Proxies – Summary

---

- ▶ **Benefits**
  - ▶ Somehow testable, with dedicated replacements
  - ▶ Dependencies are still possible to extract
- ▶ **Drawbacks**
  - ▶ Complex test setups for proxied classes

# Global Service Locator

---

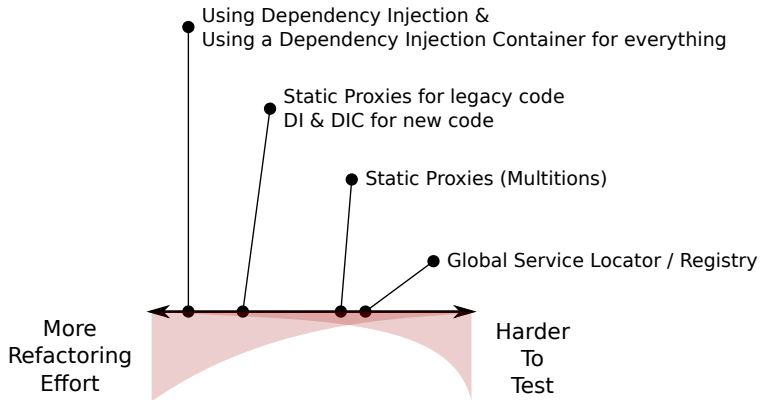
- ▶ One service locator to receive everything from
- ▶ Usually accessed statically

# Global Service Locator – Summary

---

- ▶ **Benefits**
  - ▶ Somehow testable, but requires mocking the full system
- ▶ **Drawbacks**
  - ▶ Complex test setups
  - ▶ Implicit full system dependency

# Refactoring Approaches



# Summary

---

- ▶ All approaches suck in some way.
  - ▶ Choose wisely depending on ambitions and requirements

# Refactoring

---

- ▶ Give us some code
- ▶ ... let's look at it together.



# Thanks for Listening

---

Rate this talk: <https://joind.in/7575>

## Stay in touch

- ▶ Kore Nordmann
- ▶ [kore@qafoo.com](mailto:kore@qafoo.com)
- ▶ [@koredn](https://twitter.com/koredn)
- ▶ Tobias (Toby) Schlitt
- ▶ [toby@qafoo.com](mailto:toby@qafoo.com)
- ▶ [@tobySen](https://twitter.com/tobySen)

Rent a PHP quality expert:  
<http://qafoo.com>