

Advanced OO Patterns

ConFoo 2011

Tobias Schlitt <toby@qafoo.com>

March 9, 2011



Outline

Introduction

Lazy Initialization

Dependency Injection

Service Locator

Data Storage



About me

- ▶ Degree in computer science

More than 10 years of
experience with PHP

PHP enthusiast
and OO Data Components

Senior Consultant

at Qafoo

About me

- ▶ Degree in computer science
- ▶ More than 10 years of professional PHP

Enthusiast
of Components

Qafoo

About me

- ▶ Degree in computer science
- ▶ More than 10 years of professional PHP
- ▶ Open source enthusiast
 - ▶ Apache Zeta Components
 - ▶ Arbit
 - ▶ PHPUnit
 - ▶ ...



About me

- ▶ Degree in computer science
- ▶ More than 10 years of professional PHP
- ▶ Open source enthusiast
 - ▶ Apache Zeta Components
 - ▶ Arbit
 - ▶ PHPUnit
 - ▶ ...

Co-Founder of
Qafoo GmbH

<http://qafoo.com>



About me

- ▶ Degree in computer science
- ▶ More than 10 years of professional PHP
- ▶ Open source enthusiast
 - ▶ Apache Zeta Components
 - ▶ Arbit
 - ▶ PHPUnit
 - ▶ ...

Co-Founder of
Qafoo GmbH

<http://qafoo.com>



We help people to produce
high quality PHP code.

Disclaimer

- ▶ This talk cannot be in depth

This talk shall inspire you

This talk will not show UML diagrams

This talk will show you quite some code

This talk can seriously harm your coding habbits

Disclaimer

- ▶ This talk cannot be in depth
- ▶ This talk shall inspire you

I will not show UML diagrams

I will show you quite some code

It can seriously harm your coding habits

Disclaimer

- ▶ This talk cannot be in depth
- ▶ This talk shall inspire you
- ▶ This talk will not show UML diagrams

I will show you quite some code
but it can seriously harm your coding habits

Disclaimer

- ▶ This talk cannot be in depth
- ▶ This talk shall inspire you
- ▶ This talk will not show UML diagrams
- ▶ This talk will show you quite some code

can seriously harm your coding habits

Disclaimer

- ▶ This talk cannot be in depth
- ▶ This talk shall inspire you
- ▶ This talk will not show UML diagrams
- ▶ This talk will show you quite some code
- ▶ This talk can seriously harm your coding habits

Patterns are ...

- ▶ ... universal solutions.

... good habits.

... reusable templates.



Patterns are ...

- ▶ ... universal solutions.

... good habits.

... reusable templates.



Patterns are ...

- ▶ ... universal solutions.
- ▶ ... good habits.

... patterns.



Patterns are ...

- ▶ ... universal solutions.
- ▶ ... good habits.

© 2005 Qafoo



Patterns are ...

- ▶ ... universal solutions.
- ▶ ... good habits.
- ▶ ... coding templates.



Patterns are ...

- ▶ ... universal solutions.
- ▶ ... good habits.
- ▶ ... coding templates.



Patterns are ...

- ▶ ~~... universal solutions.~~
- ▶ ~~... good habits.~~
- ▶ ~~... coding templates.~~

**... names for proven ideas how a certain class of problems
can be solved.**

Patterns are **not** ...

- ▶ ... applicable to every problem.
- ▶ ... directly transferable to code.
- ▶ ... always the best solution.



Pattern classification

- ▶ Creational
- ▶ Structural
- ▶ Behavioural
- ▶ Architectural



Well known patterns

Signal / Observer

Iterator

Visitor

Adapter

Singleton

Factory

Well known patterns

Signal / Observer

Iterator

Visitor

Adapter

Singleton

Factory

Outline

Introduction

Lazy Initialization

Dependency Injection

Service Locator

Data Storage



Motivation

- ▶ Building a full object graph is
 - ▶ Complex
 - ▶ Time consuming
 - ▶ Memory consuming

Do you create all objects in every request?

Do you create the object when needed for the first time?

Motivation

- ▶ Building a full object graph is
 - ▶ Complex
 - ▶ Time consuming
 - ▶ Memory consuming
- ▶ Do you use all objects in every request?

create object when needed for the first time

Motivation

- ▶ Building a full object graph is
 - ▶ Complex
 - ▶ Time consuming
 - ▶ Memory consuming
- ▶ Do you use all objects in every request?
- ▶ Idea: Initialize object when needed for the first time

Simple lazy initialization

```
1 <?php
2
3 class DatabaseInitializer
4 {
5     protected $dsn;
6
7     protected $db;
8
9     public function __construct( $dsn )
10    {
11        $this->dsn = $dsn;
12    }
13
14    public function getDatabase()
15    {
16        if ( $this->db == null )
17        {
18            $this->db = new Database( $this->dsn );
19        }
20        return $this->db;
21    }
22 }
```

Simple lazy initialization

```
1 <?php
2
3 class DatabaseInitializer
4 {
5     protected $dsn;
6
7     protected $db;
8
9     public function __construct( $dsn )
10    {
11        $this->dsn = $dsn;
12    }
13
14    public function getDatabase()
15    {
16        if ( $this->db == null )
17        {
18            $this->db = new Database( $this->dsn );
19        }
20        return $this->db;
21    }
22 }
```

Simple lazy initialization

```
1 <?php
2
3 class DatabaseInitializer
4 {
5     protected $dsn;
6
7     protected $db;
8
9     public function __construct( $dsn )
10    {
11        $this->dsn = $dsn;
12    }
13
14    public function getDatabase()
15    {
16        if ( $this->db == null )
17        {
18            $this->db = new Database( $this->dsn );
19        }
20        return $this->db;
21    }
22 }
```

Outline

Introduction

Lazy Initialization

Dependency Injection

Service Locator

Data Storage



Goals of good OO design

- ▶ Modular

◦ Module

◦ Interface

◦ Package

◦ Class

◦ Object

◦ Method

◦ Attribute

◦ Exception

◦ Error

◦ Design

<http://unclebob.com/ArticleS.UncleBob.PrinciplesOfOod>

Goals of good OO design

- ▶ Modular
- ▶ Flexible

<http://unclebob.com/ArticleS.UncleBob.PrinciplesOfOod>

Goals of good OO design

- ▶ Modular
- ▶ Flexible
- ▶ Reusable

<http://unclebob.com/ArticleS.UncleBob.PrinciplesOfOod>

Goals of good OO design

- ▶ Modular
- ▶ Flexible
- ▶ Reusable
- ▶ Testable

<http://unclebob.com/ArticleS.UncleBob.PrinciplesOfOod>

Goals of good OO design

- ▶ Modular
- ▶ Flexible
- ▶ Reusable
- ▶ Testable
- ▶ S.O.L.I.D.

(<http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod>)

What's the problem here?

```
1 <?php
2
3 class MessageDispatcher
4 {
5     public function __construct()
6     {
7         $this->messengers [] = new
8             JabberMessenger();
9         $this->messengers [] = new
10            MailMessenger();
11     }
12 }
13
14 class MailMessenger implements Messenger
15 {
16     public function sendMessage( $text )
17     {
18         myLogger::getInstance()->log( $text
19             );
20         $this->sendMail( /*...*/ );
21     }
22 }
```

- ▶ Inflexible
- ▶ Not reusable
- ▶ Hardly testable



What's the problem here?

```
1 <?php
2
3 class MessageDispatcher
4 {
5     public function __construct()
6     {
7         $this->messengers [] = new
8             JabberMessenger();
9         $this->messengers [] = new
10            MailMessenger();
11     }
12 }
13
14 class MailMessenger implements Messenger
15 {
16     public function sendMessage( $text )
17     {
18         myLogger::getInstance()->log( $text
19             );
20         $this->sendMail( /*...*/ );
21     }
22 }
```

► Inflexible

► Not reusable

► Hardly testable



What's the problem here?

```
1 <?php
2
3 class MessageDispatcher
4 {
5     public function __construct()
6     {
7         $this->messengers[] = new
8             JabberMessenger();
9         $this->messengers[] = new
10            MailMessenger();
11     }
12 }
13
14 class MailMessenger implements Messenger
15 {
16     public function sendMessage( $text )
17     {
18         myLogger::getInstance()->log( $text
19             );
20         $this->sendMail( /*...*/ );
21     }
22 }
```

- ▶ Inflexible
- ▶ Not reusable

▶ Hardly testable



What's the problem here?

```
1 <?php
2
3 class MessageDispatcher
4 {
5     public function __construct()
6     {
7         $this->messengers[] = new
8             JabberMessenger();
9         $this->messengers[] = new
10            MailMessenger();
11     }
12 }
13
14 class MailMessenger implements Messenger
15 {
16     public function sendMessage( $text )
17     {
18         myLogger::getInstance()->log( $text
19             );
20         $this->sendMail( /*...*/ );
21     }
22 }
```

- ▶ Inflexible
- ▶ Not reusable
- ▶ Hardly testable

Injecting dependencies

```
1 <?php
2
3 $messenger = new MessageDispatcher(
4     array(
5         new JabberMessenger( 'jabber.example.org', 'user', 'pass' ),
6         new MailMessenger(
7             new MailSmtptTransport( 'mail.example.org', 'user', 'pass' ),
8             $logger = new Logger(
9                 new LoggingDispatcher(
10                    array(
11                        new SyslogLogger(),
12                        new FileSystemLogger( 'log/errors.log' )
13                    )
14                )
15            )
16        )
17    ),
18    $logger
19 );
```

Injecting dependencies

```
1 <?php
2
3 $messenger = new MessageDispatcher(
4     array(
5         new JabberMessenger( 'jabber.example.org', 'user', 'pass' ),
6         new MailMessenger(
7             new MailSmtptTransport( 'mail.example.org', 'user', 'pass' ),
8             $logger = new Logger(
9                 new LoggingDispatcher(
10                    array(
11                        new SyslogLogger(),
12                        new FileSystemLogger( 'log/errors.log' )
13                    )
14                )
15            )
16        )
17    ),
18    $logger
19 );
```

Injecting dependencies

```
1 <?php
2
3 $messenger = new MessageDispatcher(
4     array(
5         new JabberMessenger( 'jabber.example.org', 'user', 'pass' ),
6         new MailMessenger(
7             new MailSmtptTransport( 'mail.example.org', 'user', 'pass' ),
8             $logger = new Logger(
9                 new LoggingDispatcher(
10                    array(
11                        new SyslogLogger(),
12                        new FileSystemLogger( 'log/errors.log' )
13                    )
14                )
15            )
16        )
17    ),
18    $logger
19 );
```

Injecting dependencies

```
1 <?php
2
3 $messenger = new MessageDispatcher(
4     array(
5         new JabberMessenger( 'jabber.example.org', 'user', 'pass' ),
6         new MailMessenger(
7             new MailSmtptTransport( 'mail.example.org', 'user', 'pass' ),
8             $logger = new Logger(
9                 new LoggingDispatcher(
10                    array(
11                        new SyslogLogger(),
12                        new FileSystemLogger( 'log/errors.log' )
13                    )
14                )
15            )
16        )
17    ),
18    $logger
19 );
```

Injecting dependencies

```
1 <?php
2
3 $messenger = new MessageDispatcher(
4     array(
5         new JabberMessenger( 'jabber.example.org', 'user', 'pass' ),
6         new MailMessenger(
7             new MailSmtptTransport( 'mail.example.org', 'user', 'pass' ),
8             $logger = new Logger(
9                 new LoggingDispatcher(
10                    array(
11                        new SyslogLogger(),
12                        new FileSystemLogger( 'log/errors.log' )
13                    )
14                )
15            )
16        )
17    ),
18    $logger
19 );
```

Injecting dependencies

```
1 <?php
2
3 $messenger = new MessageDispatcher(
4     array(
5         new JabberMessenger( 'jabber.example.org', 'user', 'pass' ),
6         new MailMessenger(
7             new MailSmtptTransport( 'mail.example.org', 'user', 'pass' ),
8             $logger = new Logger(
9                 new LoggingDispatcher(
10                    array(
11                        new SyslogLogger(),
12                        new FileSystemLogger( 'log/errors.log' )
13                    )
14                )
15            )
16        )
17    ),
18    $logger
19 );
```

What is desired?

- ▶ Don't have ctors all over the place
- ▶ Manage dependencies
- ▶ Replace dependencies fine grained
- ▶ Lazy initialization



The Arbit DIC

- ▶ Exemplary for a high-end DIC
 - ▶ Shared objects (initialized once)
 - ▶ Closures for lazy initialization
 - ▶ Inherent dependency resolution

▶ Custom DICs

▶ Interceptors

▶ Implementation details,

▶ <http://bit.ly/arbitDIC>

The Arbit DIC

- ▶ Exemplary for a high-end DIC
 - ▶ Shared objects (initialized once)
 - ▶ Closures for lazy initialization
 - ▶ Inherent dependency resolution
- ▶ Base class for custom DICs

▶ [Arbit interceptors](#)

▶ [Implementation details,](#)

▶ <http://bit.ly/arbitDIC>

The Arbit DIC

- ▶ Exemplary for a high-end DIC
 - ▶ Shared objects (initialized once)
 - ▶ Closures for lazy initialization
 - ▶ Inherent dependency resolution
- ▶ Base class for custom DICs
- ▶ Heavy use of interceptors

Implementation details,
see <http://bit.ly/arbitDIC>

The Arbit DIC

- ▶ Exemplary for a high-end DIC
 - ▶ Shared objects (initialized once)
 - ▶ Closures for lazy initialization
 - ▶ Inherent dependency resolution
- ▶ Base class for custom DICs
- ▶ Heavy use of interceptors
- ▶ No implementation details,
see here: <http://bit.ly/arbitDIC>



Managing dependencies

```
3 class arbitEnvironmentDIC extends arbitDependencyInjectionContainer
4 {
13     public function initialize()
14     {
38     }
39 }
```

Managing dependencies

```
3 class arbitEnvironmentDIC extends arbitDependencyInjectionContainer
4 {
13     public function initialize()
14     {
38     }
39 }
```

Managing dependencies

```
3 class arbitEnvironmentDIC extends arbitDependencyInjectionContainer
4 {
13     public function initialize()
14     {
15         $this->cache = function( $dic )
16         {
17             return new arbitFilesystemCache( $dic->request->controller );
18         };
38     }
39 }
```



Managing dependencies

```
3 class arbitEnvironmentDIC extends arbitDependencyInjectionContainer
4 {
13     public function initialize()
14     {
20         $this->messenger = function( $dic )
21         {
22             return new arbitMessenger( array(
23                 'email' => $dic->mailMessenger,
24             ) );
25         };
38     }
39 }
```

Managing dependencies

```
3 class arbitEnvironmentDIC extends arbitDependencyInjectionContainer
4 {
13     public function initialize()
14     {
27         $this->mailMessenger = function( $dic )
28         {
29             return new arbitMailMessenger(
30                 $dic->mailTransport ,
31                 $dic->configuration->main ,
32                 $dic->configuration->project( $dic->request->controller ) ,
33                 $dic->views->decorator
34             );
35         };
38     }
39 }
```


Dependency Injection Approaches

- ▶ Constructor injection
 - ▶ We just saw that

constructor injection

multiple instances?

factory injection

factory injection to some containers



Dependency Injection Approaches

- ▶ Constructor injection
 - ▶ We just saw that
- ▶ Setter injection
 - ▶ Use set*()
 - ▶ Default instances?

Dependency Injection
not applicable to some containers

Dependency Injection Approaches

- ▶ Constructor injection
 - ▶ We just saw that
- ▶ Setter injection
 - ▶ Use `set*()`
 - ▶ Default instances?
- ▶ Interface injection
 - ▶ Specific to some containers



More about DIC

- ▶ Phemto - A dependency injector for PHP 5
<http://phemto.sourceforge.net/>
- ▶ Pimple - A small PHP 5.3 dependency injection container
<https://github.com/fabpot/Pimple>
- ▶ Bucket - Basic di-container for php
<https://github.com/troelskn/bucket>
- ▶ Symfony Dependency Injection
<http://components.symfony-project.org/dependency-injection/>

► [Martin Fowler on Dependency Injection](http://martinfowler.com/articles/injection.html)

► martinfowler.com/articles/injection.html

More about DIC

- ▶ Phemto - A dependency injector for PHP 5
<http://phemto.sourceforge.net/>
- ▶ Pimple - A small PHP 5.3 dependency injection container
<https://github.com/fabpot/Pimple>
- ▶ Bucket - Basic di-container for php
<https://github.com/troelskn/bucket>
- ▶ Symfony Dependency Injection
<http://components.symfony-project.org/dependency-injection/>
- ▶ **Martin Fowler on Dependency Injection**
<http://martinfowler.com/articles/injection.html>

Outline

Introduction

Lazy Initialization

Dependency Injection

Service Locator

Data Storage



Service Locator

- ▶ Object to retrieve a *Service* from
- ▶ Deliver locator instead of service directly

constructor parameters

Service Locator

- ▶ Object to retrieve a *Service* from
- ▶ Deliver locator instead of service directly
- ▶ Basically saves constructor parameters

Service Locator

```
1 <?php
2
3 class MessageServiceLocator
4 {
5     public function __construct( array $messengers, Logger $logger )
6     { /* ... */ }
7
8     public function getMessengers()
9     { /* ... */ }
10
11     public function getLogger()
12     { /* ... */ }
13 }
14
15 class MessageDispatcher
16 {
17     public function __construct( MessageServiceLocator $serviceLoc )
18     { /* ... */ }
19 }
20
21 ?>
```

Service Locator

```
1 <?php
2
3 class MessageServiceLocator
4 {
5     public function __construct( array $messengers, Logger $logger )
6     { /* ... */ }
7
8     public function getMessengers()
9     { /* ... */ }
10
11     public function getLogger()
12     { /* ... */ }
13 }
14
15 class MessageDispatcher
16 {
17     public function __construct( MessageServiceLocator $serviceLoc )
18     { /* ... */ }
19 }
20
21 ?>
```

Service Locator

```
1 <?php
2
3 class MessageServiceLocator
4 {
5     public function __construct( array $messengers, Logger $logger )
6     { /* ... */ }
7
8     public function getMessengers()
9     { /* ... */ }
10
11     public function getLogger()
12     { /* ... */ }
13 }
14
15 class MessageDispatcher
16 {
17     public function __construct( MessageServiceLocator $serviceLoc )
18     { /* ... */ }
19 }
20
21 ?>
```

Service Locator

```
1 <?php
2
3 class MessageServiceLocator
4 {
5     public function __construct( array $messengers, Logger $logger )
6     { /* ... */ }
7
8     public function getMessengers()
9     { /* ... */ }
10
11    public function getLogger()
12    { /* ... */ }
13 }
14
15 class MessageDispatcher
16 {
17     public function __construct( MessageServiceLocator $serviceLoc )
18     { /* ... */ }
19 }
20
21 ?>
```

Pattern purism

DIC vs. Service Locator

Outline

Introduction

Lazy Initialization

Dependency Injection

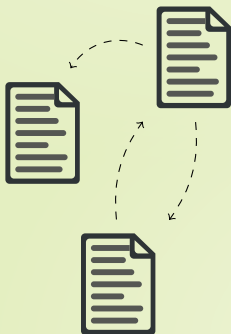
Service Locator

Data Storage



The situation

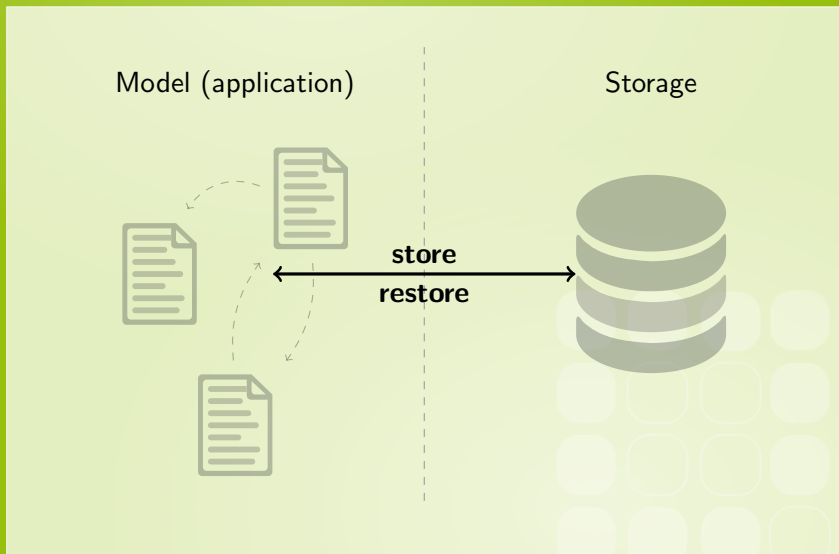
Model (application)



Storage



The situation



Challenges

- ▶ Model and storage structure differ
 - ▶ Object relational impedance mismatch
 - ▶ Different access approaches
 - ▶ Different language differences
 - ▶ Model might change
 - ▶ Storage could even be mixed



Challenges

- ▶ Model and storage structure differ
 - ▶ Object relational impedance mismatch

Different access approaches

different language differences

model and might change

could even be mixed



Challenges

- ▶ Model and storage structure differ
 - ▶ Object relational impedance mismatch
- ▶ Different access approaches

Storage differences

Access might change

Access could even be mixed



Challenges

- ▶ Model and storage structure differ
 - ▶ Object relational impedance mismatch
- ▶ Different access approaches
 - ▶ Query language differences

Model and storage structure might change
Access approaches could even be mixed



Challenges

- ▶ Model and storage structure differ
 - ▶ Object relational impedance mismatch
- ▶ Different access approaches
 - ▶ Query language differences
- ▶ Storage back end might change

could even be mixed

Challenges

- ▶ Model and storage structure differ
 - ▶ Object relational impedance mismatch
- ▶ Different access approaches
 - ▶ Query language differences
- ▶ Storage back end might change
- ▶ Back ends could even be mixed



Challenges

- ▶ Model and storage structure differ
 - ▶ Object relational impedance mismatch
- ▶ Different access approaches
 - ▶ Query language differences
- ▶ Storage back end might change
- ▶ Back ends could even be mixed
- ▶ ...



Active Record

```
1 <?php
2
3 class Invoice extends ActiveRecord
4 {
5     protected $id;
6     protected $positions;
7     protected $vat;
8     // ...
9
10    public function calculateValue()
11    { /* business logic */ }
12 }
13
14 class ActiveRecord
15 {
16     public function insert()
17     { /* ... */ }
18     public function update()
19     { /* ... */ }
20 }
```



Active Record

```
1 <?php
2
3 class Invoice extends ActiveRecord
4 {
5     protected $id;
6     protected $positions;
7     protected $vat;
8     // ...
9
10    public function calculateValue()
11    { /* business logic */ }
12 }
13
14 class ActiveRecord
15 {
16     public function insert()
17     { /* ... */ }
18     public function update()
19     { /* ... */ }
20 }
```

Active Record

```
1 <?php
2
3 class Invoice extends ActiveRecord
4 {
5     protected $id;
6     protected $positions;
7     protected $vat;
8     // ...
9
10    public function calculateValue()
11    { /* business logic */ }
12 }
13
14 class ActiveRecord
15 {
16     public function insert()
17     { /* ... */ }
18     public function update()
19     { /* ... */ }
20 }
```

Active Record

```
1 <?php
2
3 class Invoice extends ActiveRecord
4 {
5     protected $id;
6     protected $positions;
7     protected $vat;
8     // ...
9
10    public function calculateValue()
11    { /* business logic */ }
12 }
13
14 class ActiveRecord
15 {
16     public function insert()
17     { /* ... */ }
18     public function update()
19     { /* ... */ }
20 }
```

Active Record

- ▶ Combines storage and business logic
- ▶ Commonly storage logic in base class
- ▶ Model class corresponds to database record

Evaluation

Pros

- ▶ Very easy to use
- ▶ Very few code to write

Cons

- ▶ Model structure ≠ storage structure
- ▶ Classes become complex
 - ▶ Business logic
 - ▶ Storage logic
- ▶ Broken object semantics
- ▶ Changes ripple over
- ▶ Hard to exchange storage logic
- ▶ Really hard to test!

Evaluation

Pros

- ▶ Very easy to use
- ▶ Very few code to write

Cons

- ▶ Model structure ≠ storage structure
- ▶ Classes become complex
 - ▶ Business logic
 - ▶ Storage logic
- ▶ Broken object semantics
- ▶ Changes ripple over
- ▶ Hard to exchange storage logic
- ▶ Really hard to test!

Evaluation

Pros

- ▶ Very easy to use
- ▶ Very few code to write

Cons

- ▶ Model structure = storage structure
- ▶ Classes become complex
 - ▶ Business logic
 - ▶ Storage logic
- ▶ Broken object semantics
- ▶ Changes ripple over
- ▶ Hard to exchange storage logic
- ▶ Really hard to test!

Evaluation

Pros

- ▶ Very easy to use
- ▶ Very few code to write

Cons

- ▶ Model structure = storage structure
- ▶ Classes become complex
 - ▶ Business logic
 - ▶ Storage logic
- ▶ Broken object semantics
- ▶ Changes ripple over
- ▶ Hard to exchange storage logic
- ▶ Really hard to test!

Evaluation

Pros

- ▶ Very easy to use
- ▶ Very few code to write

Cons

- ▶ Model structure = storage structure
- ▶ Classes become complex
 - ▶ Business logic
 - ▶ Storage logic
- ▶ **Broken object semantics**
- ▶ Changes ripple over
- ▶ Hard to exchange storage logic
- ▶ Really hard to test!

Evaluation

Pros

- ▶ Very easy to use
- ▶ Very few code to write

Cons

- ▶ Model structure = storage structure
- ▶ Classes become complex
 - ▶ Business logic
 - ▶ Storage logic
- ▶ Broken object semantics
- ▶ **Changes ripple over**
- ▶ Hard to exchange storage logic
- ▶ Really hard to test!

Evaluation

Pros

- ▶ Very easy to use
- ▶ Very few code to write

Cons

- ▶ Model structure = storage structure
- ▶ Classes become complex
 - ▶ Business logic
 - ▶ Storage logic
- ▶ Broken object semantics
- ▶ Changes ripple over
- ▶ Hard to exchange storage logic
- ▶ Really hard to test!

Evaluation

Pros

- ▶ Very easy to use
- ▶ Very few code to write

Cons

- ▶ Model structure = storage structure
- ▶ Classes become complex
 - ▶ Business logic
 - ▶ Storage logic
- ▶ Broken object semantics
- ▶ Changes ripple over
- ▶ Hard to exchange storage logic
- ▶ Really hard to test!

Lesson learned ...

De-couple business and storage logic!



Row Data Gateway

```
1 <?php
2
3 class Invoice
4 {
5     protected $data;
6     // ...
7
8     public function __construct( InvoiceGateway $data )
9     { $this->data = $data; }
10
11     public function calculateValue()
12     { /* business logic */ }
13 }
14
15 class InvoiceGateway
16 {
17     protected $id;
18     protected $positions;
19     protected $vat;
20
21     public function insert()
22     { /* ... */ }
23     public function update()
24     { /* ... */ }
25 }
```

Row Data Gateway

```
1 <?php
2
3 class Invoice
4 {
5     protected $data;
6     // ...
7
8     public function __construct( InvoiceGateway $data )
9     { $this->data = $data; }
10
11     public function calculateValue()
12     { /* business logic */ }
13 }
14
15 class InvoiceGateway
16 {
17     protected $id;
18     protected $positions;
19     protected $vat;
20
21     public function insert()
22     { /* ... */ }
23     public function update()
24     { /* ... */ }
25 }
```

Row Data Gateway

```
1 <?php
2
3 class Invoice
4 {
5     protected $data;
6     // ...
7
8     public function __construct( InvoiceGateway $data )
9     { $this->data = $data; }
10
11     public function calculateValue()
12     { /* business logic */ }
13 }
14
15 class InvoiceGateway
16 {
17     protected $id;
18     protected $positions;
19     protected $vat;
20
21     public function insert()
22     { /* ... */ }
23     public function update()
24     { /* ... */ }
25 }
```


Row Data Gateway

```
1 <?php
2
3 class Invoice
4 {
5     protected $data;
6     // ...
7
8     public function __construct( InvoiceGateway $data )
9     { $this->data = $data; }
10
11     public function calculateValue()
12     { /* business logic */ }
13 }
14
15 class InvoiceGateway
16 {
17     protected $id;
18     protected $positions;
19     protected $vat;
20
21     public function insert()
22     { /* ... */ }
23     public function update()
24     { /* ... */ }
25 }
```

Row Data Gateway

- ▶ Decouples storage from business logic
- ▶ Still convenient OO interface for storage
- ▶ Still some objects coupled to storage

(Row Data Gateway is a somewhat similar approach)

Row Data Gateway

- ▶ Decouples storage from business logic
- ▶ Still convenient OO interface for storage
- ▶ Still some objects coupled to storage
- ▶ (Table Data Gateway is a somewhat similar approach)

Evaluation

Pros

- ▶ Easy to use
- ▶ Few code to write
 - ▶ Gateways can be generated

Cons

- ▶ Some objects model DB structure
- ▶ Changes still ripple over
- ▶ Still hard to exchange storage logic

Evaluation

Pros

- ▶ Easy to use
- ▶ Few code to write
 - ▶ Gateways can be generated
- ▶ Easier to test

Cons

- ▶ Some objects model DB structure
- ▶ Changes still ripple over
- ▶ Still hard to exchange storage logic

Evaluation

Pros

- ▶ Easy to use
- ▶ Few code to write
 - ▶ Gateways can be generated
- ▶ Easier to test

Cons

- ▶ Some objects model DB structure
- ▶ Changes still ripple over
- ▶ Still hard to exchange storage logic

Evaluation

Pros

- ▶ Easy to use
- ▶ Few code to write
 - ▶ Gateways can be generated
- ▶ Easier to test

Cons

- ▶ Some objects model DB structure
- ▶ Changes still ripple over
- ▶ Still hard to exchange storage logic

Evaluation

Pros

- ▶ Easy to use
- ▶ Few code to write
 - ▶ Gateways can be generated
- ▶ Easier to test

Cons

- ▶ Some objects model DB structure
- ▶ Changes still ripple over
- ▶ Still hard to exchange storage logic

Data Mapper

```
1 <?php
2
3 class Invoice
4 {
5     protected $positions;
6     protected $vat;
7     // ...
8
9     public function calculateValue()
10    { /* business logic */ }
11 }
12
13 interface InvoiceMapper
14 {
15     public function store( Invoice $invoice );
16     public function update( Invoice $invoice );
17     public function findByCustomer( Customer $customer );
18 }
19
20 class DbInvoiceMapper implements InvoiceMapper
21 {
22     // ...
23 }
24
25 ?>
```

Data Mapper

```
1 <?php
2
3 class Invoice
4 {
5     protected $positions;
6     protected $vat;
7     // ...
8
9     public function calculateValue()
10    { /* business logic */ }
11 }
12
13 interface InvoiceMapper
14 {
15     public function store( Invoice $invoice );
16     public function update( Invoice $invoice );
17     public function findByCustomer( Customer $customer );
18 }
19
20 class DbInvoiceMapper implements InvoiceMapper
21 {
22     // ...
23 }
24
25 ?>
```

Data Mapper

```
1 <?php
2
3 class Invoice
4 {
5     protected $positions;
6     protected $vat;
7     // ...
8
9     public function calculateValue()
10    { /* business logic */ }
11 }
12
13 interface InvoiceMapper
14 {
15     public function store( Invoice $invoice );
16     public function update( Invoice $invoice );
17     public function findByCustomer( Customer $customer );
18 }
19
20 class DbInvoiceMapper implements InvoiceMapper
21 {
22     // ...
23 }
24
25 ?>
```

Data Mapper

```
1 <?php
2
3 class Invoice
4 {
5     protected $positions;
6     protected $vat;
7     // ...
8
9     public function calculateValue()
10    { /* business logic */ }
11 }
12
13 interface InvoiceMapper
14 {
15     public function store( Invoice $invoice );
16     public function update( Invoice $invoice );
17     public function findByCustomer( Customer $customer );
18 }
19
20 class DbInvoiceMapper implements InvoiceMapper
21 {
22     // ...
23 }
24
25 ?>
```

Data Mapper

- ▶ Decouple storage from model
- ▶ No OO modelling of DB structure
- ▶ Model does not even know a database exists

Evaluation

Pros

- ▶ Complete decoupling
 - ▶ Client is not aware of storage
 - ▶ Single interface
 - ▶ Can use different storages!
 - ▶ Changes only affect storage layer
 - ▶ Changes only affect storage layer
 - ▶ Easy testing

Cons

- ▶ Quite simple to write
- ▶ Mapping can become complex

Evaluation

Pros

- ▶ Complete decoupling
- ▶ Model is not aware of storage
- ▶ Single interface
- ▶ Support for different storages!
- ▶ Changes only affect
storage layer
- ▶ Changes only affect
storage layer
- ▶ Easy testing

Cons

- ▶ Quite simple to write
- ▶ Mapping can become complex

Evaluation

Pros

- ▶ Complete decoupling
- ▶ Model is not aware of storage
- ▶ Clean storage interface

▶ Can use different storages!

▶ Changes only affect

storage layer

▶ Changes only affect

storage layer

▶ Easier testing

Cons

▶ Quite simple to write

▶ Mapping can become complex

Evaluation

Pros

- ▶ Complete decoupling
- ▶ Model is not aware of storage
- ▶ Clean storage interface
 - ▶ Implement different storages!

▶ Changes only affect

storage layer

▶ Changes only affect

storage layer

▶ Easier testing

Cons

▶ Quite simple to write

▶ Mapping can become complex

Evaluation

Pros

- ▶ Complete decoupling
- ▶ Model is not aware of storage
- ▶ Clean storage interface
 - ▶ Implement different storages!
- ▶ DB changes only affect mapping layer
- ▶ Model changes only affect mapping layer

easy testing

Cons

- ▶ Quite some code to write
- ▶ Mapping can become complex

Evaluation

Pros

- ▶ Complete decoupling
- ▶ Model is not aware of storage
- ▶ Clean storage interface
 - ▶ Implement different storages!
- ▶ DB changes only affect mapping layer
- ▶ Model changes only affect mapping layer
- ▶ Nice for testing

Cons

- ▶ Quite some code to write
- ▶ Mapping can become complex

Evaluation

Pros

- ▶ Complete decoupling
- ▶ Model is not aware of storage
- ▶ Clean storage interface
 - ▶ Implement different storages!
- ▶ DB changes only affect mapping layer
- ▶ Model changes only affect mapping layer
- ▶ Nice for testing

Cons

- ▶ Quite some code to write
- ▶ Mapping can become complex

Evaluation

Pros

- ▶ Complete decoupling
- ▶ Model is not aware of storage
- ▶ Clean storage interface
 - ▶ Implement different storages!
- ▶ DB changes only affect mapping layer
- ▶ Model changes only affect mapping layer
- ▶ Nice for testing

Cons

- ▶ Quite some code to write
- ▶ Mapping can become complex

Attribution

Large parts of this talk are inspired by

Patterns of Enterprise Application Architecture

by Martin Fowler

ISBN 978-0321127426 — <http://amzn.to/PofEAA>

Highly recommended!

Conclusion

- ▶ Patterns are not the holy grail!

- ▶ Patterns are just design names to good ideas
- ▶ Patterns are just names to talk about concepts
- ▶ Patterns are just words to describe you



Conclusion

- ▶ Patterns are not the holy grail!
- ▶ They assign names to good ideas
- ▶ They help you to talk about concepts
- ▶ They can inspire you



Q / A

Are there any questions left?



Thanks for listening

Please rate this talk at
<http://joind.in/2774>
and / or give me some feedback right now!



Thanks for listening

Please rate this talk at
<http://joind.in/2774>
and / or give me some feedback right now!

This is very important for ...

- ▶ Speakers
- ▶ Organizers
- ▶ You!



Thanks for listening

Please rate this talk at
<http://joind.in/2774>
and / or give me some feedback right now!

Stay in touch

- ▶ Tobias Schlitt
- ▶ toby@qafoo.com
- ▶ @tobySen / @qafoo

Rent a PHP quality expert:

<http://qafoo.com>