

---

# Advanced OO Patterns

## IPC 2011

Tobias Schlitt (@tobySen)

2011-10-10

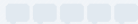
# Take a minute ...

[Events](#)[Talks](#)[About](#)[Help](#)[Blog](#)

## Advanced OO Patterns

[Tobias Schlitt](#) (10.Oct.2011 at 16:00)

Talk at [International PHP Conference / WebTech Con 2011](#) (English - UK)



You already know Singleton, Signal/Observer, Factory and friends. But, which object oriented techniques are en vogue in the PHP world and how can you seize their power? This talk gives you an overview on Dependency Injection, Data Mapper and other OO patterns the PHP world talks about right now, using practical code examples.

Quicklink: <http://joind.in/3847> ([QR code](#))

<http://joind.in/3847>

# Outline

---

Introduction

Dependency Injection

Service Locator

Data Storage

# About me

---

- ▶ Degree in computer science



# About me

---

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



# About me

---

- ▶ Degree in computer science
- ▶ More than 10 years of professional PHP
- ▶ Open source enthusiast
- ▶ Contributing to various FLOSS projects



# About me

---

- ▶ Degree in computer science
- ▶ More than 10 years of professional PHP
- ▶ Open source enthusiast
- ▶ Contributing to various FLOSS projects

Co-founder of



# About me

---

- ▶ Degree in computer science
- ▶ More than 10 years of professional PHP
- ▶ Open source enthusiast
- ▶ Contributing to various FLOSS projects

Co-founder of



**We help people to create  
high quality PHP  
applications.**



# About me

---

- ▶ Degree in computer science
- ▶ More than 10 years of professional PHP
- ▶ Open source enthusiast
- ▶ Contributing to various FLOSS projects

Co-founder of



**We help people to create  
high quality PHP  
applications.**

<http://qafoo.com>

# Disclaimer

---

- ▶ This talk cannot be in depth

# Disclaimer

---

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

# Disclaimer

---

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

# 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

# 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 ...

---

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

# Patterns are **not** ...

---

- ▶ ... applicable to every problem.
- ▶ ... directly transferable to code.
- ▶ ... written in stone.
- ▶ ... 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

**Dependency Injection**

Service Locator

Data Storage

# Goals of good OO design

---

- ▶ Modular

# Goals of good OO design

---

- ▶ Modular
- ▶ Flexible

# Goals of good OO design

---

- ▶ Modular
- ▶ Flexible
- ▶ Reusable

# Goals of good OO design

---

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



# 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
            JabberMessenger();
8         $this->messengers[] = new
            MailMessenger();
9     }
10 }
11
12 class MailMessenger implements Messenger
13 {
14     public function sendMessage( $text )
15     {
16         myLogger::getInstance()->log( $text
17             );
18         $this->sendMail( /*...*/ );
19     }
20 }
```

# 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

# What's the problem here?

---

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

- ▶ Inflexible
- ▶ Not reusable

# What's the problem here?

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

- ▶ 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?

---

- ▶ Inject all dependencies
- ▶ Separate object construction from logic

# What is desired?

---

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

# Builders

---

```
1 <?php
2
3 class MessageDispatcherBuilder
4 {
5     protected $logBuilder;
6     protected $messageDispatcher;
7
8     public function __construct( LogBuilder $logBuilder )
9     { /* ... */ }
10
11    public function createMessageDispatcher()
12    {
13        if ( !isset( $this->messageDispatcher ) )
14        {
15            $this->messageDispatcher = new MessageDispatcher(
16                array(
17                    new JabberMessenger( /* ... */ ),
18                    new MailMessenger(
19                        new MailSmtptTransport( /* ... */ ),
20                        $this->logBuilder->createLogger()
21                    ) ),
22                    $this->logBuilder->createLogger() );
23        }
24        return $this->messageDispatcher;
25    }
26 }
```

# Builders

```
1 <?php
2
3 class MessageDispatcherBuilder
4 {
5     protected $logBuilder;
6     protected $messageDispatcher;
7
8     public function __construct( LogBuilder $logBuilder )
9     { /* ... */ }
10
11    public function createMessageDispatcher()
12    {
13        if ( !isset( $this->messageDispatcher ) )
14        {
15            $this->messageDispatcher = new MessageDispatcher(
16                array(
17                    new JabberMessenger( /* ... */ ),
18                    new MailMessenger(
19                        new MailSmtptTransport( /* ... */ ),
20                        $this->logBuilder->createLogger()
21                    ) ),
22                    $this->logBuilder->createLogger() );
23        }
24        return $this->messageDispatcher;
25    }
26 }
```

# Builders

```
1 <?php
2
3 class MessageDispatcherBuilder
4 {
5     protected $logBuilder;
6     protected $messageDispatcher;
7
8     public function __construct( LogBuilder $logBuilder )
9     { /* ... */ }
10
11     public function createMessageDispatcher()
12     {
13         if ( !isset( $this->messageDispatcher ) )
14         {
15             $this->messageDispatcher = new MessageDispatcher(
16                 array(
17                     new JabberMessenger( /* ... */ ),
18                     new MailMessenger(
19                         new MailSmtptTransport( /* ... */ ),
20                         $this->logBuilder->createLogger()
21                     ) ),
22                 $this->logBuilder->createLogger() );
23         }
24         return $this->messageDispatcher;
25     }
26 }
```



# Builders

```
1 <?php
2
3 class MessageDispatcherBuilder
4 {
5     protected $logBuilder;
6     protected $messageDispatcher;
7
8     public function __construct( LogBuilder $logBuilder )
9     { /* ... */ }
10
11     public function createMessageDispatcher()
12     {
13         if ( !isset( $this->messageDispatcher ) )
14         {
15             $this->messageDispatcher = new MessageDispatcher(
16                 array(
17                     new JabberMessenger( /* ... */ ),
18                     new MailMessenger(
19                         new MailSmtptTransport( /* ... */ ),
20                         $this->logBuilder->createLogger()
21                     ) ),
22                 $this->logBuilder->createLogger() );
23         }
24         return $this->messageDispatcher;
25     }
26 }
```

# Builders

```
1 <?php
2
3 class MessageDispatcherBuilder
4 {
5     protected $logBuilder;
6     protected $messageDispatcher;
7
8     public function __construct( LogBuilder $logBuilder )
9     { /* ... */ }
10
11    public function createMessageDispatcher()
12    {
13        if ( !isset( $this->messageDispatcher ) )
14        {
15            $this->messageDispatcher = new MessageDispatcher(
16                array(
17                    new JabberMessenger( /* ... */ ),
18                    new MailMessenger(
19                        new MailSmtptTransport( /* ... */ ),
20                        $this->logBuilder->createLogger()
21                    ) ),
22                    $this->logBuilder->createLogger() );
23        }
24        return $this->messageDispatcher;
25    }
26 }
```

# Builders

```
1 <?php
2
3 class MessageDispatcherBuilder
4 {
5     protected $logBuilder;
6     protected $messageDispatcher;
7
8     public function __construct( LogBuilder $logBuilder )
9     { /* ... */ }
10
11    public function createMessageDispatcher()
12    {
13        if ( !isset( $this->messageDispatcher ) )
14        {
15            $this->messageDispatcher = new MessageDispatcher(
16                array(
17                    new JabberMessenger( /* ... */ ),
18                    new MailMessenger(
19                        new MailSmtptTransport( /* ... */ ),
20                        $this->logBuilder->createLogger()
21                    ) ),
22                    $this->logBuilder->createLogger() );
23        }
24        return $this->messageDispatcher;
25    }
26 }
```

# Builders

```
1 <?php
2
3 class MessageDispatcherBuilder
4 {
5     protected $logBuilder;
6     protected $messageDispatcher;
7
8     public function __construct( LogBuilder $logBuilder )
9     { /* ... */ }
10
11    public function createMessageDispatcher()
12    {
13        if ( !isset( $this->messageDispatcher ) )
14        {
15            $this->messageDispatcher = new MessageDispatcher(
16                array(
17                    new JabberMessenger( /* ... */ ),
18                    new MailMessenger(
19                        new MailSmtptTransport( /* ... */ ),
20                    $this->logBuilder->createLogger()
21                ) ),
22                $this->logBuilder->createLogger() );
23        }
24        return $this->messageDispatcher;
25    }
26 }
```

# Dependency Injection Container

---

- ▶ Takes care of object creation
- ▶ Manages dependencies
- ▶ Allows fine grained replacement

# Manual DIC

---

```
1 <?php
2
3 class DependencyInjectionContainer
4 {
5     protected $cache;
6
7
8
9
10    public function getCache()
11    {
12        if ( !isset( $this->cache ) )
13        {
14            $this->cache = new FilesystemCache( /* ... */ );
15        }
16        return $this->cache;
17    }
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42 }
```

# Manual DIC

---

```
1 <?php
2
3 class DependencyInjectionContainer
4 {
5     protected $messenger;
6
7     public function __construct(
8         array $mailMessengers
9     ) {
10        $this->mailMessengers = $mailMessengers;
11    }
12
13    public function getMessenger()
14    {
15        if ( !isset( $this->messenger ) )
16        {
17            $this->messenger = new Messenger( array(
18                'email' => $this->getMailMessenger(),
19            ) );
20        }
21        return $this->messenger;
22    }
23
24    public function getMailMessenger()
25    {
26        return $this->mailMessengers[0];
27    }
28
29    public function setMailMessenger(
30        Messenger $messenger
31    ) {
32        $this->mailMessengers[0] = $messenger;
33    }
34
35    public function getMailMessengers()
36    {
37        return $this->mailMessengers;
38    }
39
40    public function setMailMessengers(
41        array $mailMessengers
42    ) {
```

# Manual DIC

---

```
1 <?php
2
3 class DependencyInjectionContainer
4 {
5     protected $mailMessenger;
6
7     protected $mailTransport;
8
9     public function getMailTransport()
10    {
11        if ( !isset( $this->mailTransport ) )
12        {
13            $this->mailTransport = new MailTransport(
14                $this->getMailMessenger()
15            );
16        }
17        return $this->mailTransport;
18    }
19
20    public function getMailMessenger()
21    /* ... */
22
23    public function setMailMessenger( MailMessenger $mailMessenger )
24    {
25        $this->mailMessenger = $mailMessenger;
26    }
27
28    public function setMailTransport( MailTransport $mailTransport )
29    {
30        $this->mailTransport = $mailTransport;
31    }
32
33    public function getMailTransportAndMessenger()
34    {
35        return array( $this->getMailTransport(), $this->getMailMessenger() );
36    }
37
38    public function setMailTransportAndMessenger( MailTransport $mailTransport, MailMessenger $mailMessenger )
39    {
40        $this->setMailTransport( $mailTransport );
41        $this->setMailMessenger( $mailMessenger );
42    }
43 }
```



# The Arbit DIC

---

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

# The Arbit DIC

---

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

# The Arbit DIC

---

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

# The Arbit DIC

---

- ▶ Exemplary
  - ▶ 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>

# The Arbit DIC

---

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

# The Arbit DIC

---

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

# The Arbit DIC

---

```
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 }
```

# The Arbit DIC

---

```
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 }
```



# The Arbit DIC

---

```
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

# Dependency Injection Approaches

---

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

# 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/>

## 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

Dependency Injection

**Service Locator**

Data Storage

# Motivation

---

- ▶ Registry for service objects
- ▶ Clients ask for object to perform a task
- ▶ Originates from compiled languages
- ▶ Sometimes used similar to DI
- ▶ Often used to reduce parameter lists
- ▶ More or less a registry!



# Service locator example

---

```
1 <?php
2
3 class GatewayLocator
4 {
5     public function __construct(
6         UserGateway $userGateway, ProductGateway $productGateway )
7     { /* ... */ }
8
9     public function getUserGateway()
10    { /* ... */ }
11
12    public function getProductGateway()
13    { /* ... */ }
14 }
15
16 class UserController
17 {
18     public function __construct( GatewayLocator $gatewayLocator )
19     { /* ... */ }
20 }
21
22 ?>
```

# Service locator example

---

```
1 <?php
2
3 class GatewayLocator
4 {
5     public function __construct(
6         UserGateway $userGateway, ProductGateway $productGateway )
7     { /* ... */ }
8
9     public function getUserGateway()
10    { /* ... */ }
11
12    public function getProductGateway()
13    { /* ... */ }
14 }
15
16 class UserController
17 {
18     public function __construct( GatewayLocator $gatewayLocator )
19     { /* ... */ }
20 }
21
22 ?>
```

# Service locator example

---

```
1 <?php
2
3 class GatewayLocator
4 {
5     public function __construct(
6         UserGateway $userGateway, ProductGateway $productGateway )
7     { /* ... */ }
8
9     public function getUserGateway()
10    { /* ... */ }
11
12    public function getProductGateway()
13    { /* ... */ }
14 }
15
16 class UserController
17 {
18     public function __construct( GatewayLocator $gatewayLocator )
19     { /* ... */ }
20 }
21
22 ?>
```

# Problems

---

- ▶ Hides dependencies
- ▶ Hides code smells
- ▶ Enables usage of arbitrary objects

# Universal Constructors

---

- ▶ Ctor expects only single parameter
- ▶ An array with everything the object needs
- ▶ Potentially merged with default values

# Universal Constructors Example

---

```
1 <?php
2
3 class UserController
4 {
5     public function __construct( $config = null )
6     { /* ... */ }
7 }
```

~~Service Locator~~

# Outline

---

Introduction

Dependency Injection

Service Locator

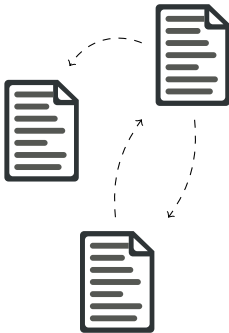
Data Storage



# The situation

---

Model (application)

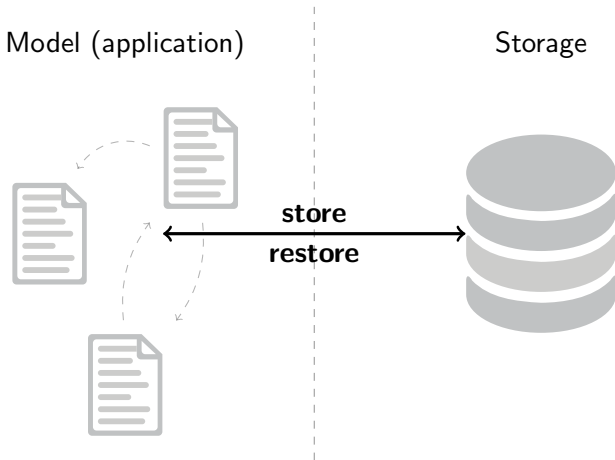


Storage



# The situation

---



# Challenges

---

- ▶ Model and storage structure differ

# Challenges

---

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

# Challenges

---

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

# Challenges

---

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

# Challenges

---

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

# 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

# Evaluation

---

## Pros

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

## Cons



# Evaluation

---

## Pros

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

## Cons

- ▶ Model structure = storage structure

# Evaluation

---

## Pros

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

## Cons

- ▶ Model structure = storage structure
- ▶ Classes become complex
  - ▶ Business logic
  - ▶ Storage logic

# 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

# 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

# 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

# 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!

# Approaches

---

- ▶ Table data gateway
- ▶ Row data gateway



# 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

## Cons

# Evaluation

---

## Pros

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

## Cons

# Evaluation

---

## Pros

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

## Cons



# Evaluation

---

## Pros

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

## Cons

# 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

## Cons

# 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

# 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

# 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

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!

# Conclusion

---

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



Are there any questions left?

# Thanks for listening

---

Please rate this talk at  
<http://joind.in/3847>

(Slides will be linked there)

# Thanks for listening

---

Please rate this talk at  
<http://joind.in/3847>

## Stay in touch

- ▶ Tobias Schlitt
- ▶ [toby@qafoo.com](mailto:toby@qafoo.com)
- ▶ @tobySen / @qafoo

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