

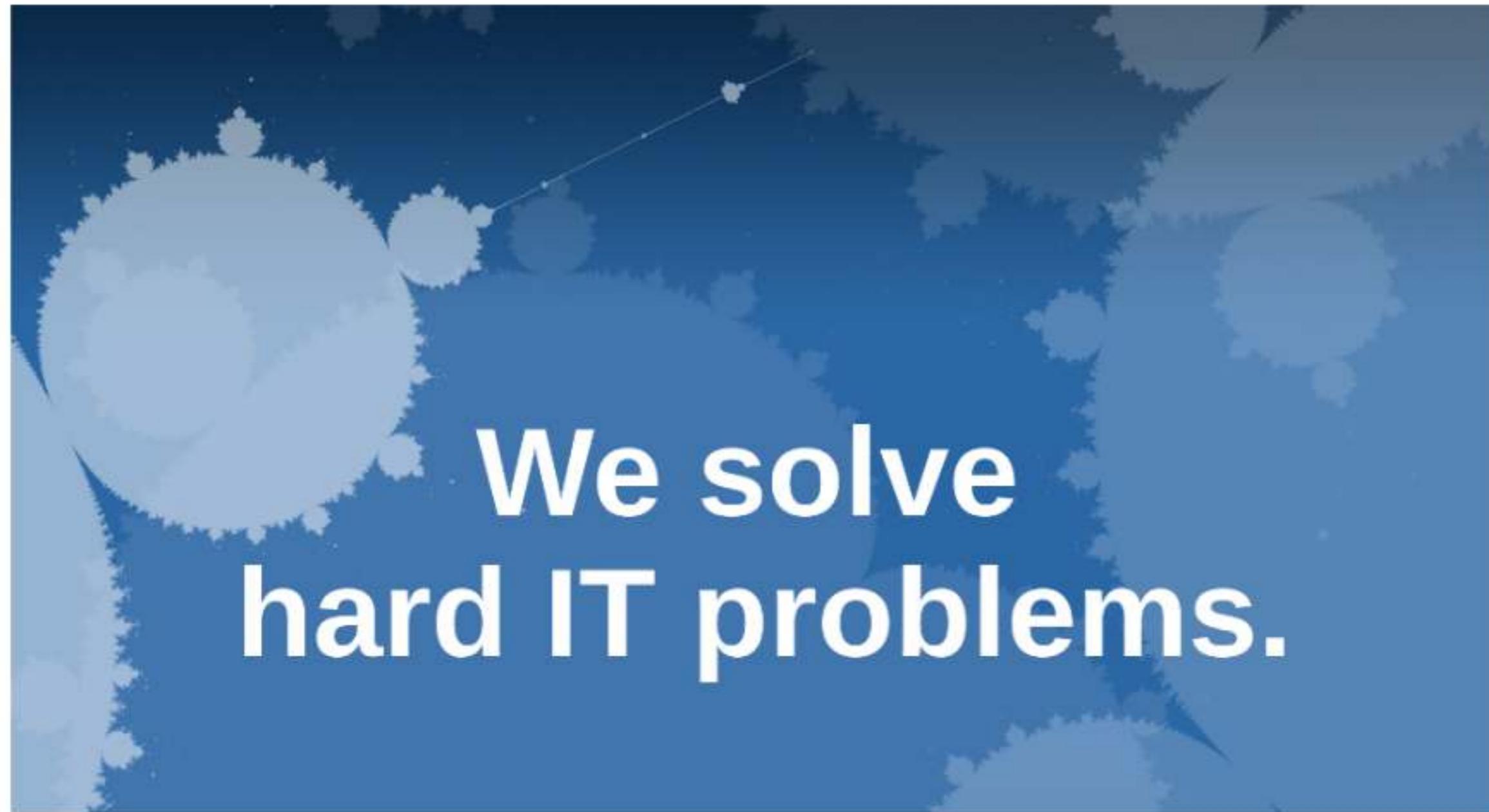


Architecture Refactoring

Moving Towards DDD

Michael Haeuslmann - @michaelhaeu

Michael Haeuslmann @ **TNG** TECHNOLOGY CONSULTING



@michaelhaeu

TNG TECHNOLOGY
CONSULTING

international
PHP
conference

Facts and Figures



Foundation in 2001



Located in Unterföhring
near Munich



99% University Degree Holder
60% with a PhD



2 Internal Training Days per Month



Approx. 275 Employees



41% with a Degree in
Physics

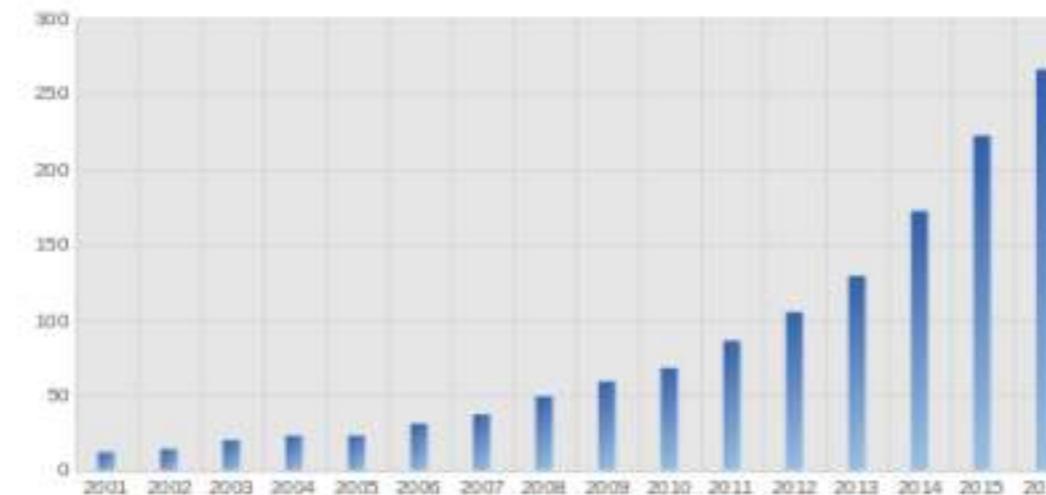


Fluctuation < 6%



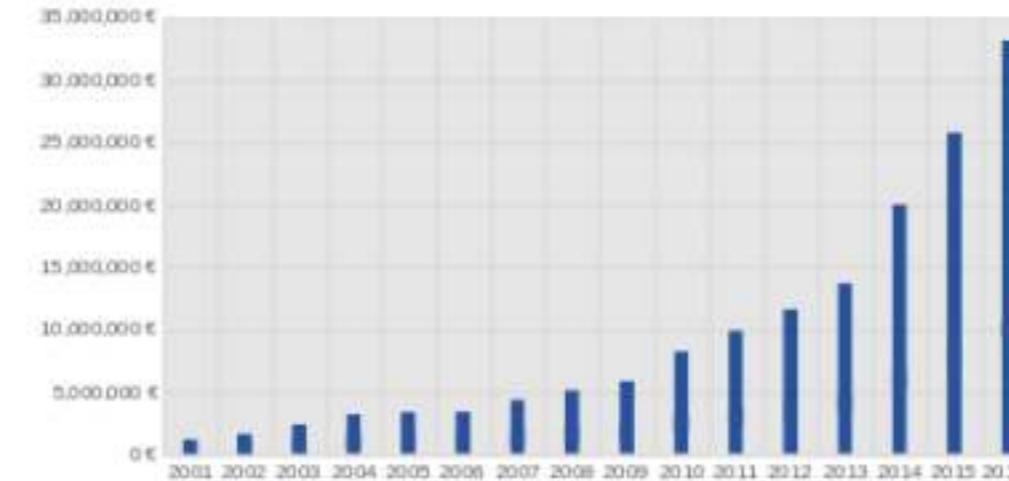
1-4 External Conferences per Year

Number of Employees



Note: The number of employees is stated as of the end of December.

Sales Growth





@michaelhaeu

TNG TECHNOLOGY
CONSULTING

international
PHP conference

Refactoring Architecture is hard!



@michaelhaeu

TNG TECHNOLOGY
CONSULTING

international
PHP conference

Refactoring
Architecture is
hard!

Just rewrite!



@michaelhaeu

TNG TECHNOLOGY
CONSULTING

international
PHP conference

Dubai 27 years ago



traveL





Case Study: Vodafone Kabeldeutschland

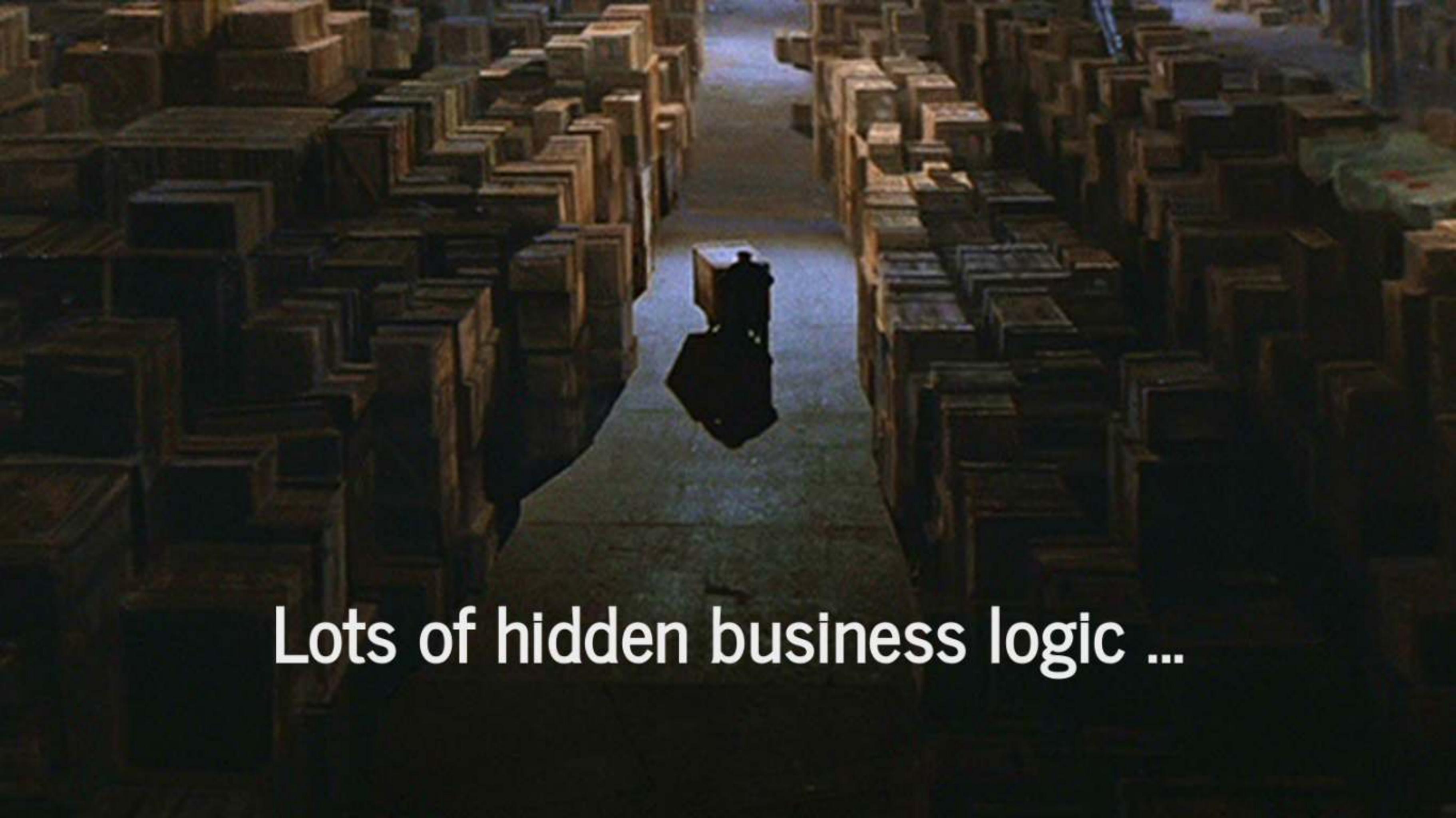
The screenshot shows the homepage of the Vodafone Kabeldeutschland website. At the top, there's a navigation bar with links for 'Privatkunden' (selected), 'Geschäftskunden', 'Beratung & Bestellung: 0800 - 664 94 23', 'Kontakt', and 'Shopfinder'. Below the navigation is a red header bar with links for 'Mobilfunk', 'Internet & Telefon', 'TV', 'Kombi-Pakete', 'Hilfe', and 'MeinVodafone', along with a search bar labeled 'Ich suche nach...'. The main content area features a large image of a man and a young boy sitting on a couch, looking at a tablet together. To the left of the image, there's a promotional offer for 'GigaSpeed': 'Red Internet & Phone 500 Cable' at '19,99 € im Monat ab dem 15. Monat 49,99 €'. Below this offer is a red button with the text 'Bis zu 12 Freimonate'. On the left side of the main content, there's a section for 'GigaSpeed' with the text 'Surfe mit bis zu 500 Mbit/s im Kabel-Glasfasernetz *' and a 'Zum Angebot' button. At the bottom left, there's a logo for 'Internet & Phone' featuring a stylized house and phone icon.

Rewrite
Just
FOR YOU





**Communication
problems ...**



Lots of hidden business logic ...



Big bang integration ...



A photograph of a man with dark hair and a beard, seen from the side and looking upwards towards a whiteboard. He is wearing a dark t-shirt.

We need a plan!

Side bar
Phu.

spot light

What do we want?

Architecture which ...

- supports the use cases and operation of the system
- supports the maintenance of the system
- supports the development of the system
- supports the deployment of the system

Good architecture is ...

- ... about intent not about methods/tools
- ... allows major decisions to be deferred

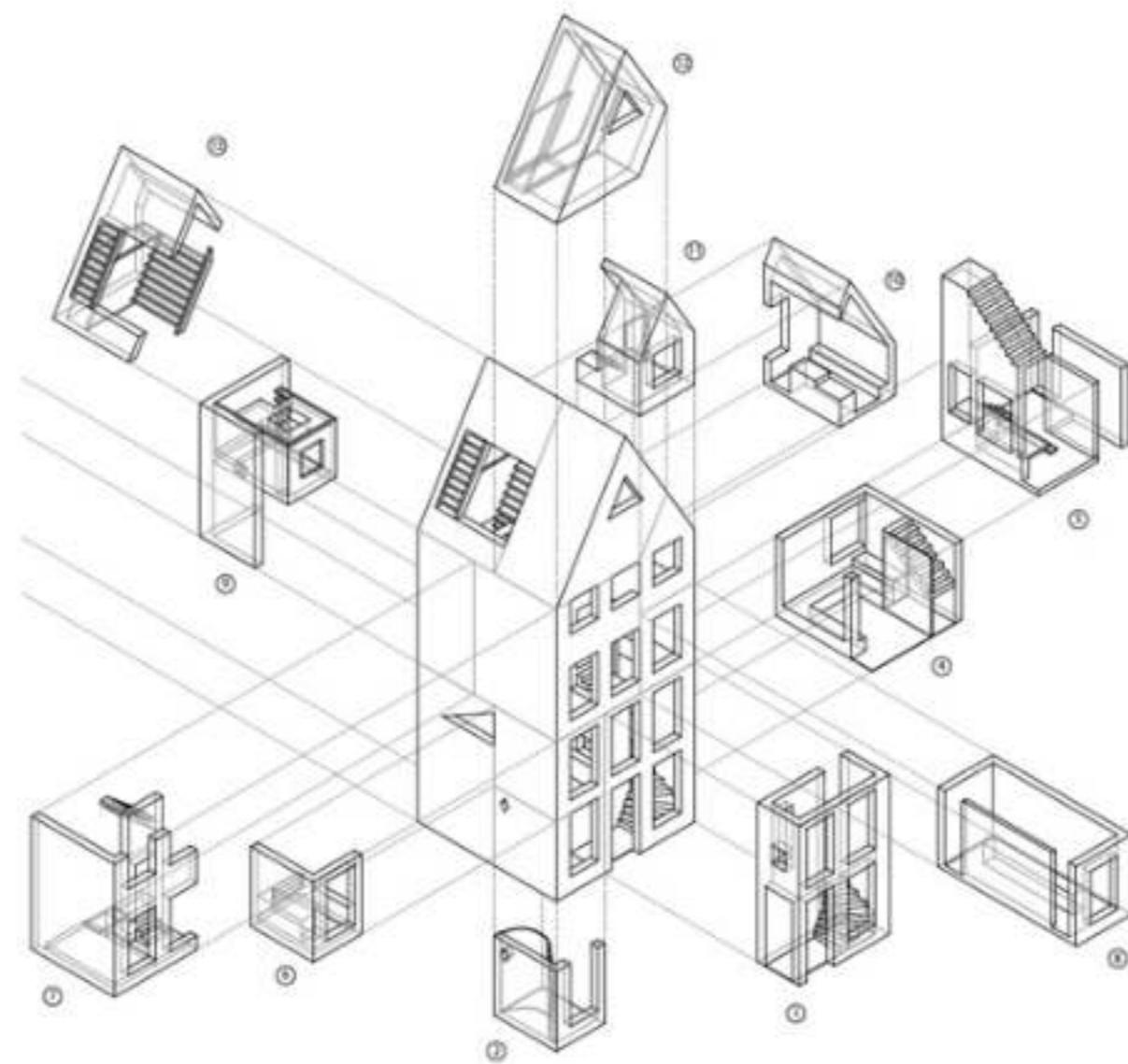


@michaelhaeu

Good architecture is ...

- ... about intent not about methods/tools
 - ... allows major decisions to be deferred
- "Architecture, The Lost Years", Robert C. Martin

What's the intent?

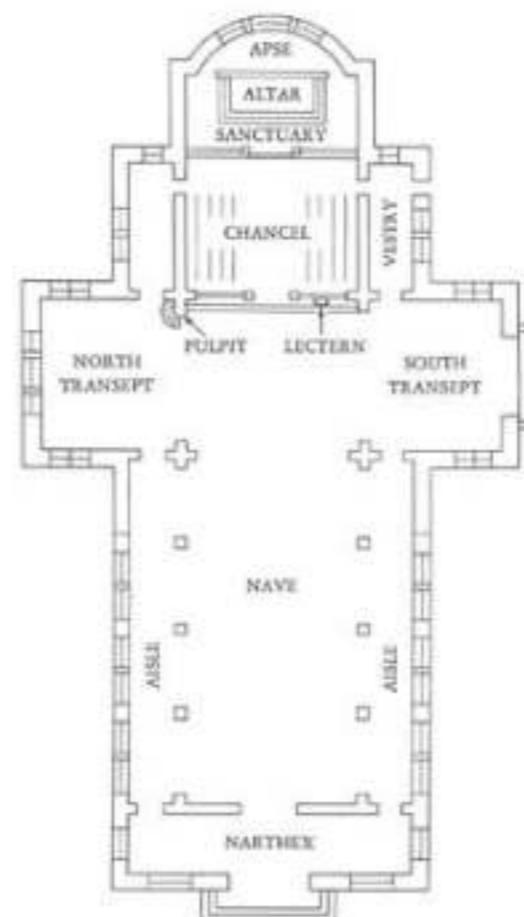


@michaelhaeu

TNG TECHNOLOGY
CONSULTING

international
PHP conference

What's the intent?



@michaelhaeu

What's the intent?

```
. └── application  
    ├── composer.json  
    ├── contributing.md  
    ├── index.php  
    ├── license.txt  
    ├── readme.rst  
    └── system  
        └── user_guide
```

What's the intent?

```
application
├── cache
├── config
├── controllers
├── core
├── helpers
├── hooks
├── index.html
├── language
├── libraries
├── logs
├── models
└── third_party
    └── views
```



@michaelhaeu

*Good architecture is
about intent not about
methods or tools.*

- Robert C. Martin
("Uncle Bob")

*Good architecture
allows major decisions
to be deferred.*

- Robert C. Martin
("Uncle Bob")

How do we find problems in our architecture?

- read ...



STOP



We need some tools!

dePHPend



Visualizations & Assertions



@michaelhaeu

TNG TECHNOLOGY
CONSULTING

international
PHP conference

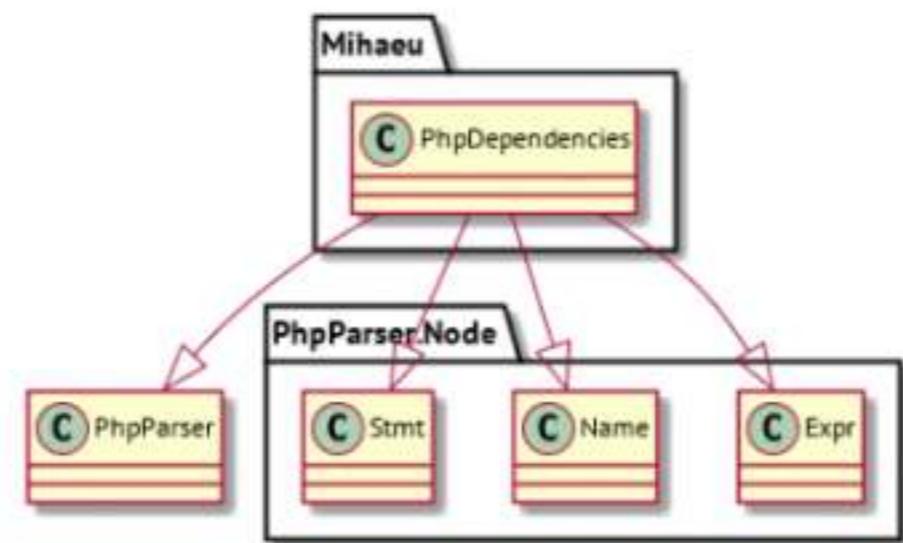


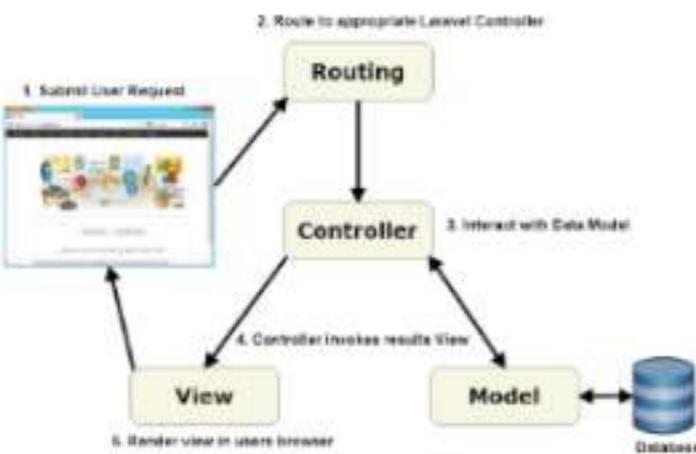
@michaelhaeu

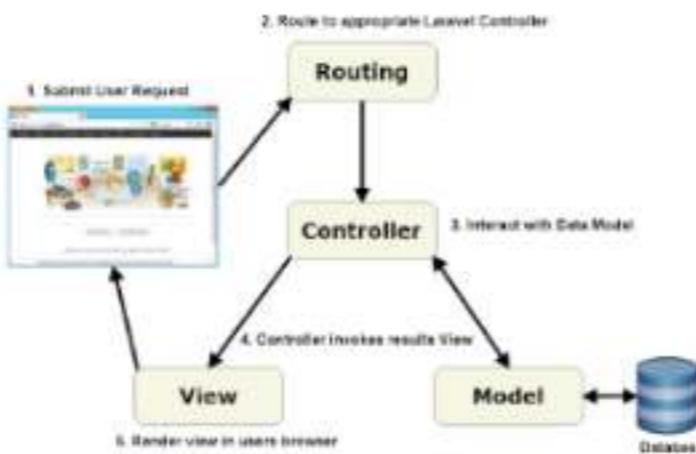
TNG TECHNOLOGY CONSULTING



Visualizations







```
<?php

$cmd = shell_exec('dephpend text src --no-classes');
$constraints = [
    'Model.* --> .*View',
    'View.* --> .*Model',
];
$regex = '/('.implode('|', $constraints).')/x';

if (preg_match($regex, $cmd)) {
    echo 'Architecture violation'.PHP_EOL;
    exit(1);
}
```



@michaelhaeu



<https://github.com/mihaeu/dephpend>

Whatever you end up using:
Test your architecture!



Metrics

- number of dependencies, dependents
- package size
- abstractness of code

pmd and PhplnpectionsEA to detect other architectural smells

- gives you hints along the way
- too many dependencies in a class
- inheritance hierarchy too deep
- static calls



<https://phpmd.org/>

<https://github.com/kalessil/phpinspectionsea>

PhpStorm for refactorings and renaming

- no vim is not an alternative
- no neither is emacs
- and don't get me started on Netbeans



@michaelhaeu

Refactoring Browser by Qafoo

The screenshot shows a Mac OS X style window titled "PHP Refactoring Browser". The window contains two main sections: a left sidebar and a right content area.

Left Sidebar:

- Title:** PHP Refactoring Browser
- Description:** A command line refactoring tool for PHP
- Links:** View the Project on GitHub (QafooLabs/php-refactoring-browser)
- Buttons:** Download PHAR, View On GitHub, Built by Qafoo
- Note:** This software is under development and in alpha state. Refactorings do not contain all necessary pre-conditions and might mess up your code. Check the diffs carefully before applying the patches.

Right Content Area:

- Title:** PHP Refactoring Browser
- Note:** This software is under development and in alpha state. Refactorings do not contain all necessary pre-conditions and might mess up your code. Check the diffs carefully before applying the patches.
- Status:** build passing
- Description:** Automatic Refactorings for PHP Code by generating diffs that describe the refactorings steps. To prevent simple mistakes during refactorings, an automated tool is a great.
- Text:** The library is standing on the shoulder of giants, using multiple existing libraries:
- [PHP Parser](#) by Nikic
 - [PHP Token Reflection](#) from Ondřej Nešpor

For JavaScript

- Visualizations: grep for require/import + dot (graphviz)
- Assertions: grep for require/import + scripts
- pmd for JavaScript
- WebStorm or VSCode

One more thing: Functional Tests

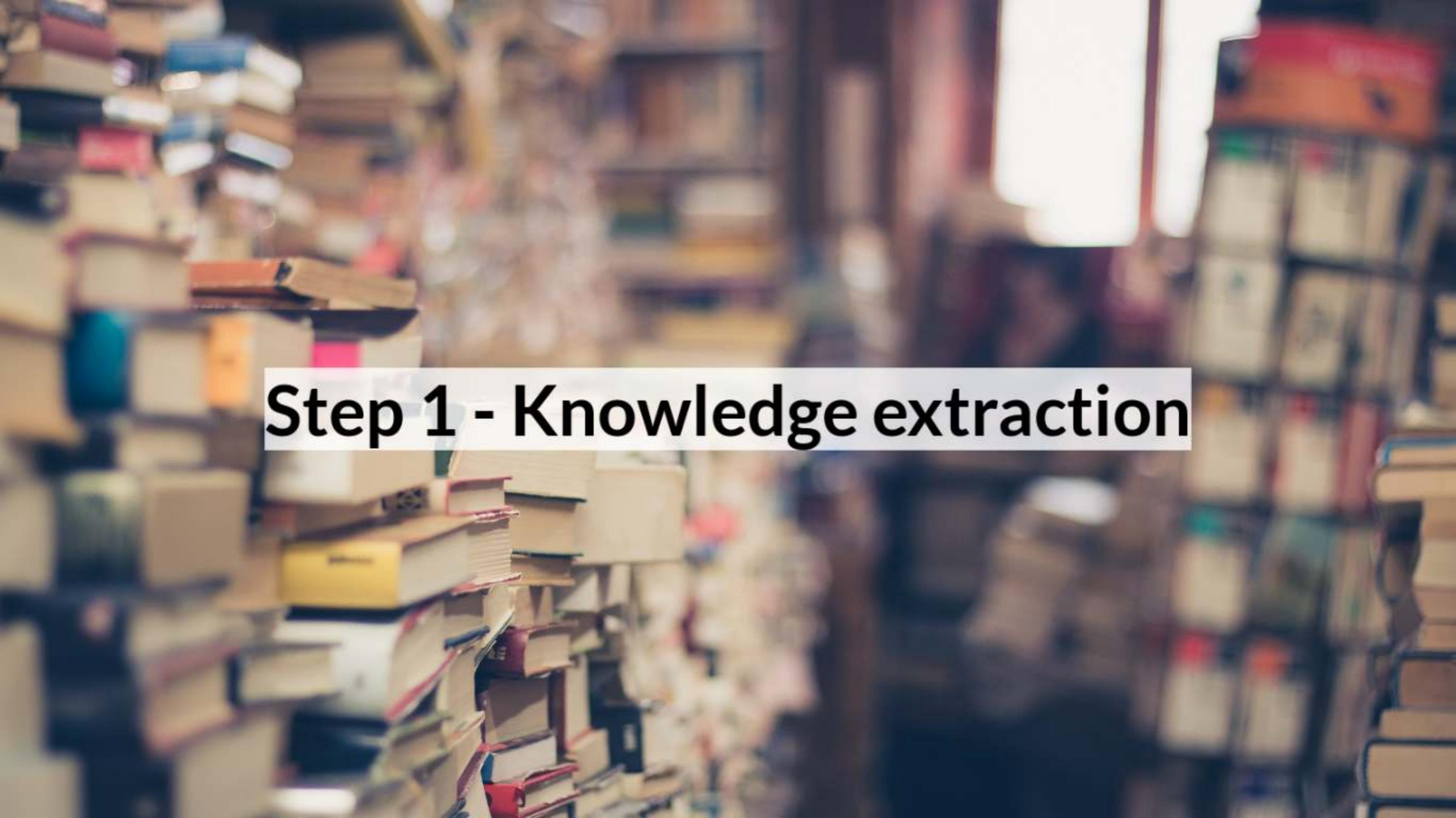
- ⇒ refactoring should not break existing functionality
- ⇒ functional tests are more resilient towards architectural changes

Behat





Let's begin

A blurred background image showing numerous stacks of books of various sizes and colors, creating a dense, textured appearance.

Step 1 - Knowledge extraction

What's this?

```
/** @var string */  
private $address;
```

- what does it contain?
- how is it validated?
- where is it validated?
- is the validation always the same?
- when is it populated?
- who uses it?



@michaelhaeu

Or that?

```
/** @var array */
private $address;
```

- not much better
- most questions remain unanswered



@michaelhaeu

Is this okay?

```
class AddressStruct
{
    public $street;
    public $zipCode;
    public $city;
}
```

- tiny step in the right direction
- helps with usages
- state is unknown and mutable
- still difficult to refactor

Immutable Domain Objects

```
class Address {  
    private $street;  
    private $zipCode;  
    private $city;  
  
    public function __construct(  
        Street $street, ZipCode $zipCode, City $city  
    ) {  
        // ...  
    }  
  
    public function street() : Street { return $this->street; }  
}
```



Immutable Domain Objects

- is the Address valid?
- ⇒ as long as it exists it's valid
- unused attributes?
- ⇒ the IDE will know
- adding/removing properties?
- ⇒ all related unit tests will break
(which is great)
- what if a value changes?
- ⇒ they can't



@michaelhaeu

Immutable Domain Objects

```
class ZipCode
{
    private $zipCode;

    public function __construct(int $zipCode)
    {
        $this->ensureValidGermanZipCode($zipCode);

        $this->zipCode = $zipCode;
    }

    private function ensureValidGermanZipCode(int $zipCode) : void
    {
        if ($zipCode < 10000 || $zipCode > 99999) {
            throw new InvalidZipCodeException($zipCode);
        }
    }
}
```

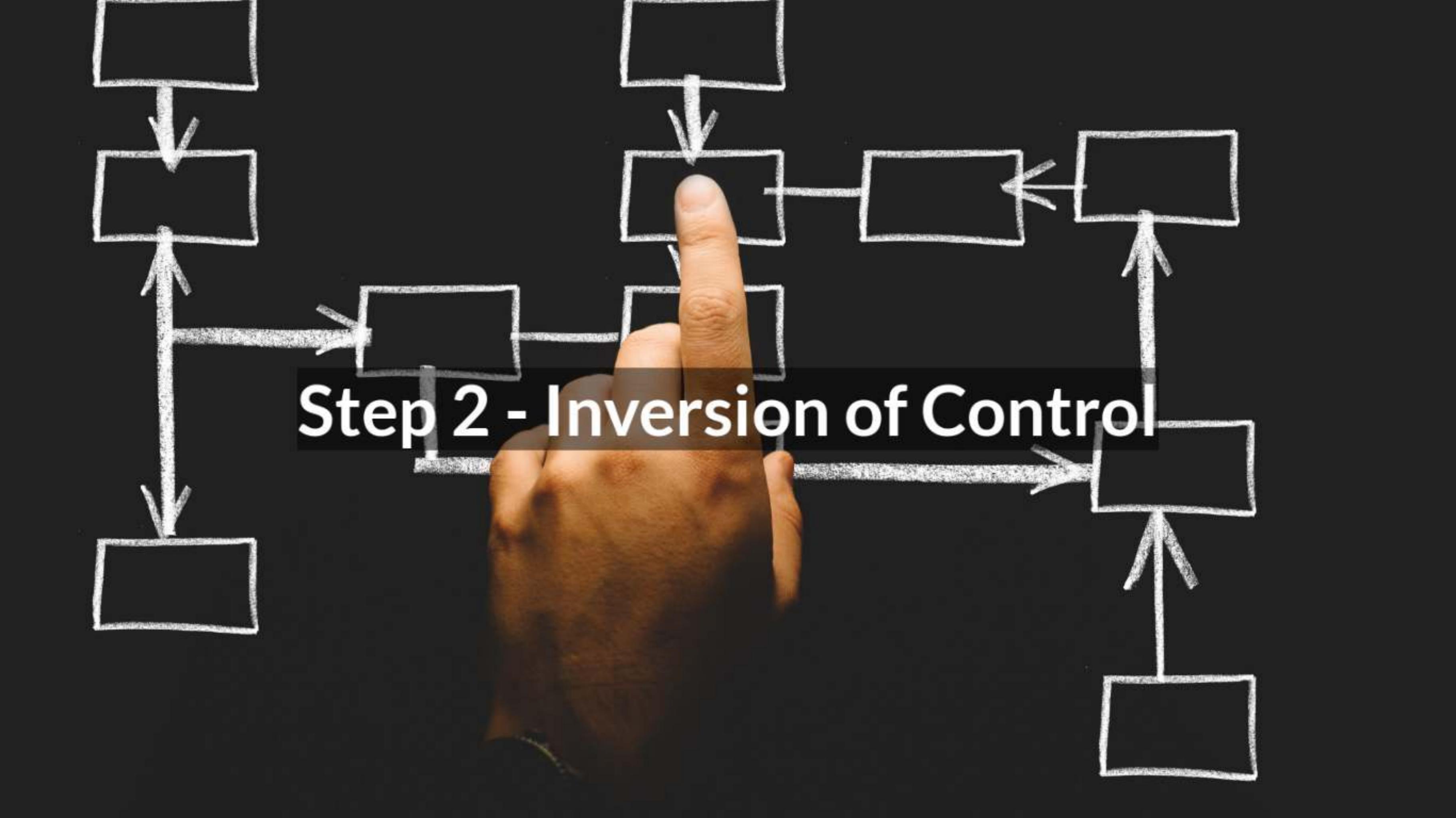
Summary: Immutable Domain Objects

- not an architectural refactoring, but important to reason about the business logic
- looks easy, but very painful process
- you'll find many inconsistencies along the way
- Knowledge gain: "who talks to whom about what?"
- Myth: arrays are more performant (*)

Summary: Immutable Domain Objects

- not an architectural refactoring, but important to reason about the business logic
- looks easy, but very painful process
- you'll find many inconsistencies along the way
- Knowledge gain: "who talks to whom about what?"
- Myth: arrays are more performant (*)

(*) [PHP data structures and the impact of PHP 7 on them](#)



Step 2 - Inversion of Control

What do I depend on?

```
class X
{
    public function __construct($dependencyContainer)
    {
        $this->dependencyContainer = $dependencyContainer;
    }

    // ... a few hundred lines later

    public function someFunction()
    {
        $userRepository = $this->dependencyContainer->get('UserRepository');
        $users = $userRepository->getAll();
    }
}
```

What do I depend on?

```
class X
{
    // ... a few hundred lines later

    public function someFunction()
    {
        $users = DB::connection('foo')->select(** ... *);
    }
}
```

What do I depend on?

```
class X
{
    // ... a few hundred lines later

    public function someFunction()
    {
        $users = DB::connection('foo')->select(** ... *);
    }
}
```

Don't make me think about stupid things!



@michaelhaeu

Dependency Injection

- always use constructor dependency injection
- never inject the dependency container itself
- domain objects are "newables"
- factories are fine
- use a lib or roll your own, the less magic the better



@michaelhaeu

Common Problems

Side Effects in the Constructor

```
function __construct($db) {  
    $this->db = $db;  
    $this->init(); // oh oh  
}
```

- Option 1: lazy instantiation using e.g. factories
- Option 2: Extract side effects (temporal coupling 

Common Problems

Static Usage

```
class UserHelper
{
    public static function isAuthorized() { /** ... */ }
```

Common Problems

Static Usage

```
class UserHelper
{
    public static function isAuthorized() { /** ... */ }
```

Wrap it!

Common Problems

Static Usage

```
class UserHelper
{
    public static function isAuthorized() { /** ... */ }
```

Wrap it!

```
class UserHelperWrapper
{
    public function isAuthorized() {
        return UserHelper::isAuthorized();
    }
}
```

Common Problems

Framework uses magic

- common problem with most frameworks
- magic instantiation of controllers
 - ⇒ if possible write a custom dispatcher
 - ⇒ strangler pattern
 - ⇒ some dependency containers support legacy frameworks



Step 3 - Show Intent

What?

Be explicit!

- move away from the old-school MVC folder structure
- your folder structure should reflect your architecture
- your architecture should reflect your intent (domain)

Be explicit!

- move away from the old-school MVC folder structure
- your folder structure should reflect your architecture
- your architecture should reflect your intent (domain)

IT is hard, don't make me think about stupid things!



@michaelhaeu

Cohesion

Measure of how well modules fits together.

- Coincidental cohesion
- Logical association
- Temporal cohesion
- Sequential cohesion
- Functional cohesion



@michaelhaeu

Cohesion

Measure of how well modules fits together.

- Coincidental cohesion
- Logical association
- Temporal cohesion
- Sequential cohesion
- Functional cohesion

In other words: things that are highly cohesive should be grouped together

A man wearing a white helmet and dark clothing is riding a light-colored motorcycle. He is looking down at something in his hands. The background is heavily blurred, suggesting motion, which creates a sense of speed and delivery. The overall color palette is warm, with yellows and browns.

**Step 4 - Abstract the
delivery mechanism**

Abstract the framework/database/delivery mechanism

- the web is a delivery mechanism
- your framework is a delivery mechanism
- delivery mechanisms have nothing to do with your core application
- the source of data should not matter
(relational, document, service call, cache, ...)



@michaelhaeu

Common Problems and Questions

- using active record? ⇒ hide them behind a **Repository**
- do your controllers know what happens to the data?
- do your controllers know where the data comes from?
- how expensive would it be to implement event sourcing?
- the source of data should not matter
(relational, document, service call, cache, ...)



@michaelhaeu

Common Problems and Questions

- using active record? ⇒ hide them behind a **Repository**
- do your controllers know what happens to the data?
- do your controllers know where the data comes from?
- how expensive would it be to implement event sourcing?
- the source of data should not matter
(relational, document, service call, cache, ...)

Could you move your core code out of your "web repository"?



@michaelhaeu



Let the fun begin!

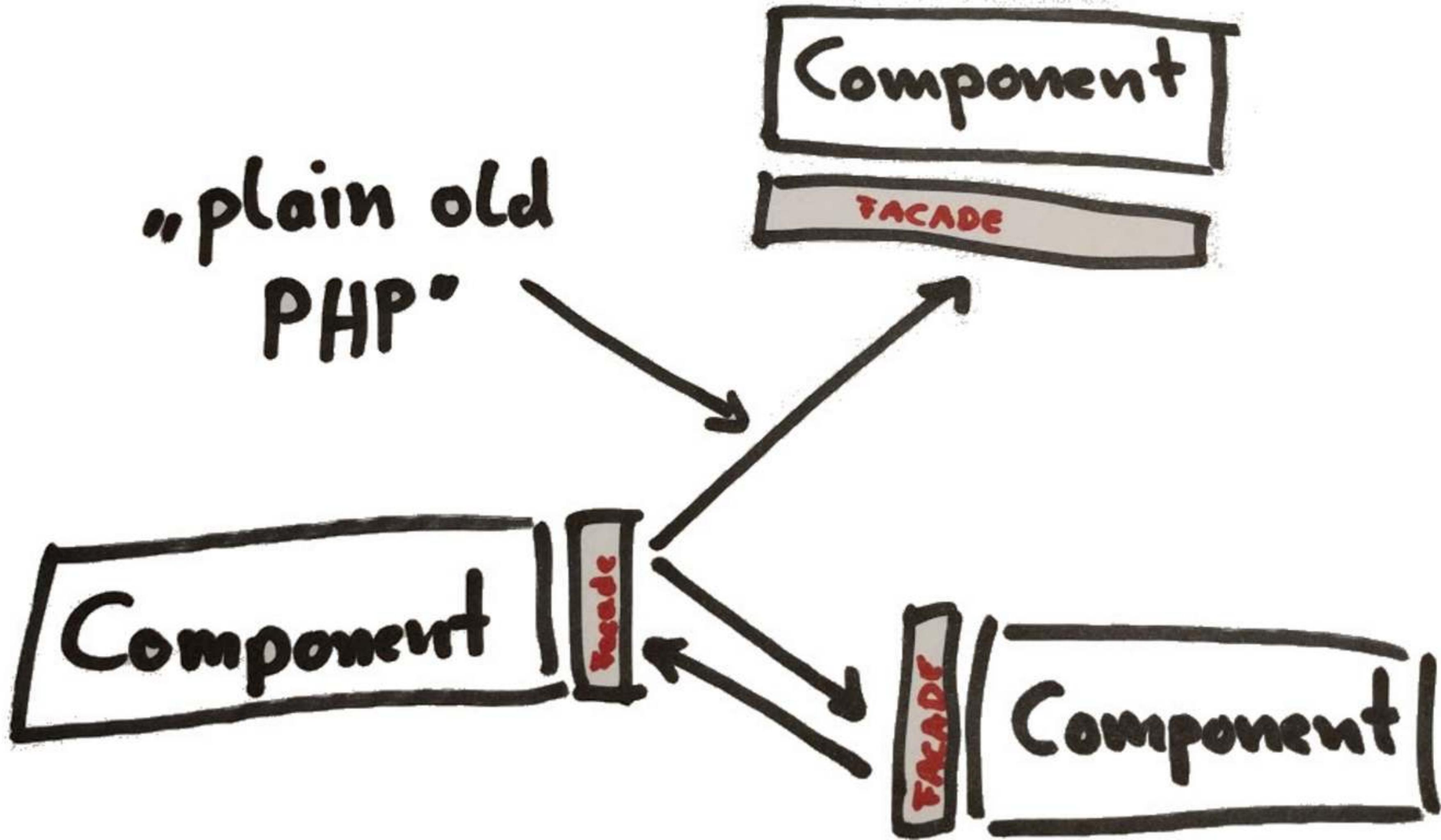
Microservices

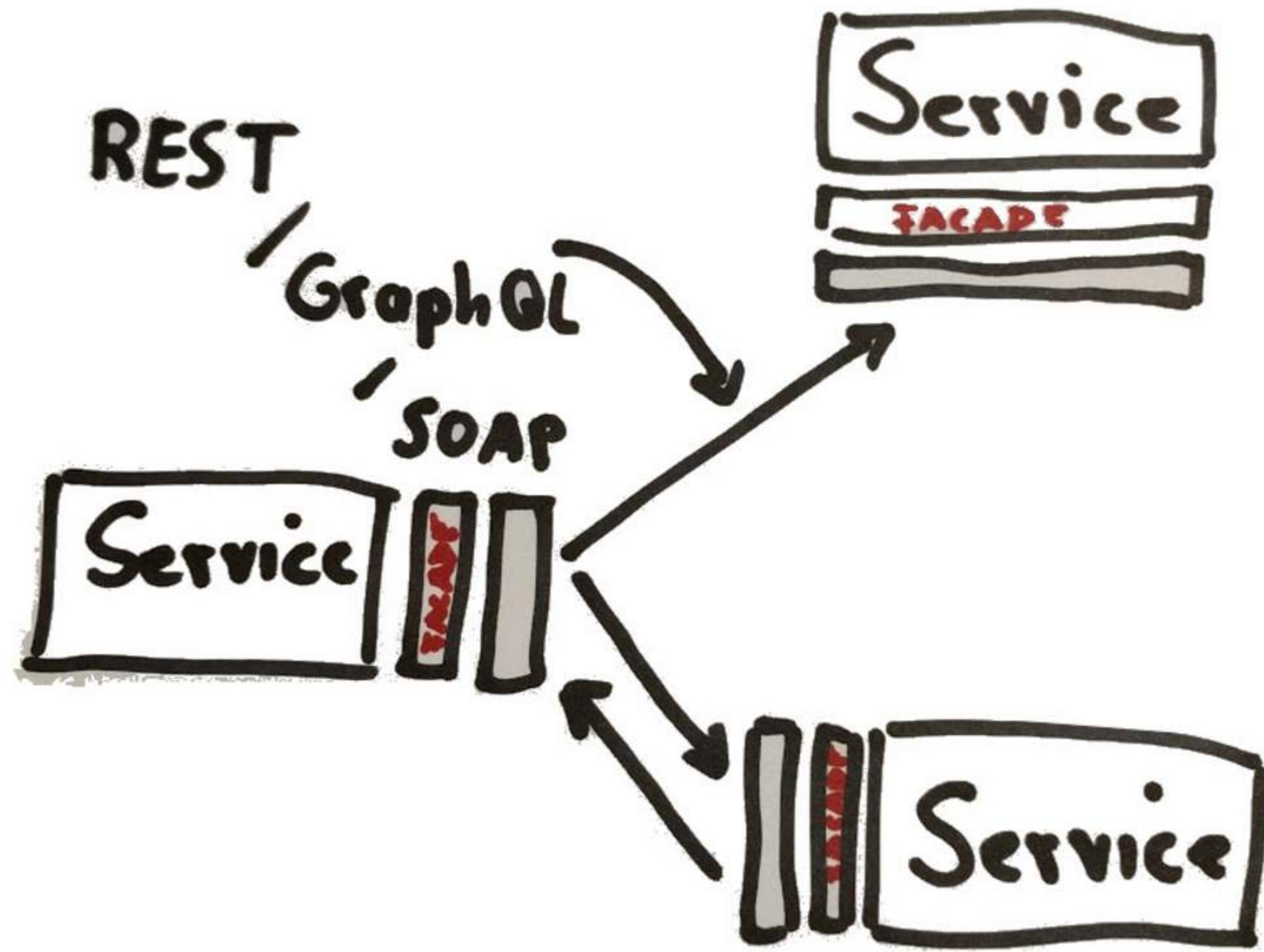
- you already have your components with clear interfaces
- those interfaces have to be mapped to REST/GraphQL/SOAP requests/responses
- that's about it
(given the previous steps were successful)



@michaelhaeu

“plain old
PHP”





Fallacies of distributed computing

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.



@michaelhaeu

Domain Driven Design (DDD)

- we already built domain objects
- aggregates should be easy to identify between the different components
 - it's what your facades should be using anyways
- layered architecture has been ensured using constraints
- service access is encapsulated
- storage access is encapsulated
- however: introducing events is not free

Domain Driven Design (DDD)

- we already built domain objects
- aggregates should be easy to identify between the different components
 - it's what your facades should be using anyways
- layered architecture has been ensured using constraints
- service access is encapsulated
- storage access is encapsulated
- however: introducing events is not free

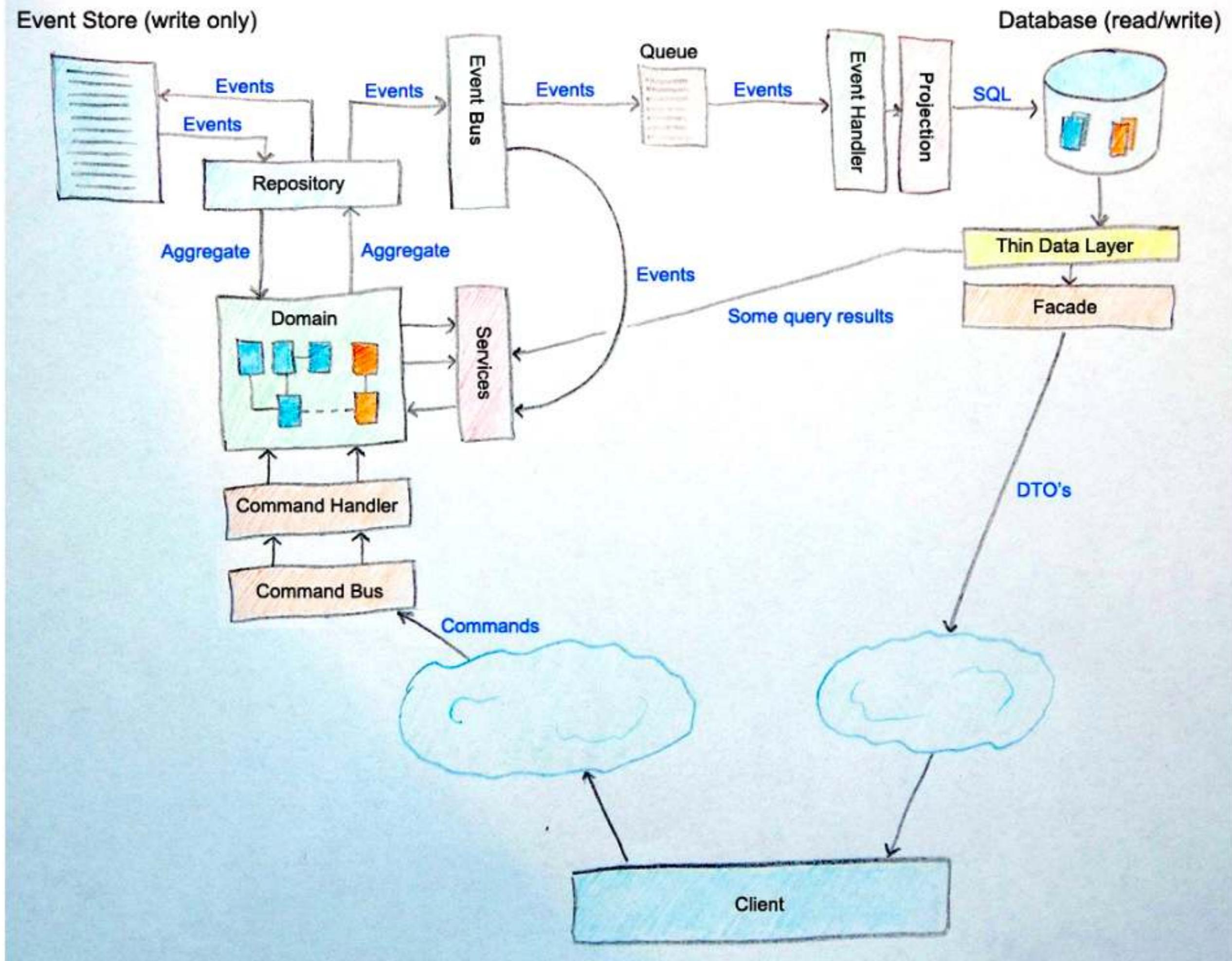
A photograph of a group of military personnel in camouflage uniforms standing in formation, looking forward. The soldiers are wearing caps and uniforms, and the background shows a hallway with other personnel.

Optional Step 5 - CQRS

Command Query
Responsibility Segregation

Event Sourcing

- the data source is abstract (e.g. repository)
- the rest of the application doesn't care about the source
- send requests from a client through the command bus
- application deals with commands and talks to repository
- repository maps what happened to events



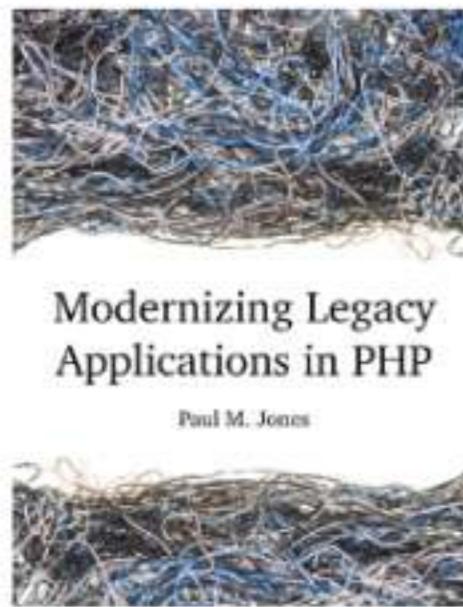
Summary

- architecture refactorings are costly and risky
 - make sure your goals are measurable
 - define and test architecture constraints
-
1. Knowledge Extraction (Domain modeling)
 2. Inversion of Control
 3. Abstract the delivery mechanism
 4. Show Intent



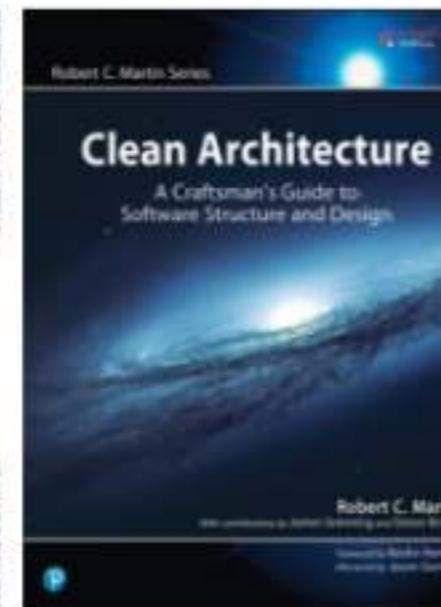
@michaelhaeu

Reading Recommendations



Modernizing Legacy
Applications in PHP

Paul M. Jones



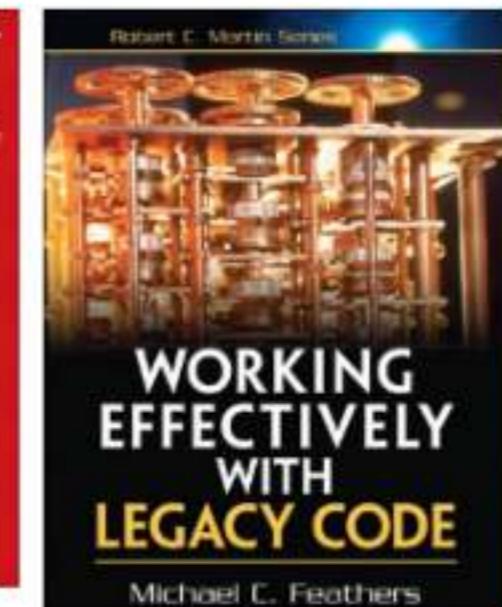
Clean Architecture
A Craftsman's Guide to
Software Structure and Design

Robert C. Martin



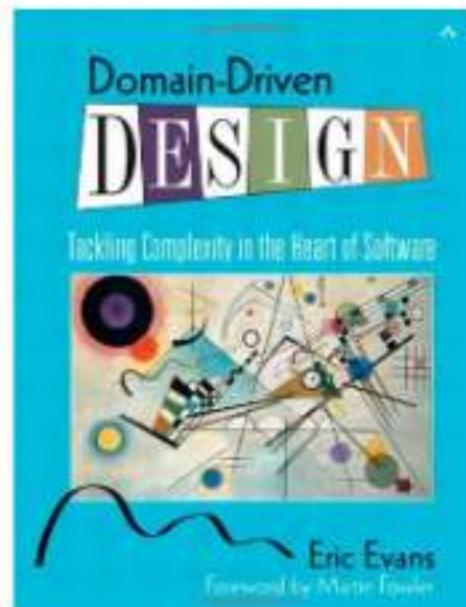
PATTERNS OF
ENTERPRISE
APPLICATION
ARCHITECTURE

MARTIN FOWLER
Ward Cunningham
Kent Beck
Martin Fowler
Eric Evans
Robert C. Martin
Steve Freeman



WORKING
EFFECTIVELY
WITH
LEGACY CODE

Michael C. Feathers



Domain-Driven
DESIGN

Tackling Complexity in the Heart of Software



Eric Evans

Foreword by Mike Feathers

Modernizing
Legacy
Applications in
PHP
(Paul M. Jones)

Clean
Architecture
(Robert C. Martin)

Patterns of
Enterprise
Application
Architecture
(Martin Fowler)

Working
Effectively with
Legacy Code
(Michael C. Feathers)

Domain Driven
Design
(Eric Evans)



@michaelhaeu



Thank you!

Michael Haeuslmann - @michaelhaeu



Credits

Stock photos from pexels.com

Indiana Jones <http://raven.theraider.net/showthread.php?t=8100&page=1>

Big Bang <https://www.youtube.com/watch?v=hDcWqidxvz4>

CQRS à la Greg Young - Mark Nijhof

Cohesion <https://courses.cs.washington.edu/courses/cse403/96sp/coloring.html>