



# Architecture Refactoring

Moving Towards DDD

Michael Haeusmann - @michaelhaeu

Michael Haeuslmann @ **TNG**  TECHNOLOGY  
CONSULTING



We solve  
hard IT problems.

## 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

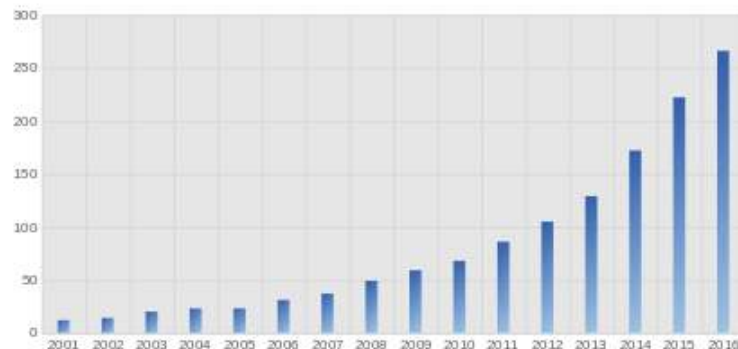


Fluctuation < 6%



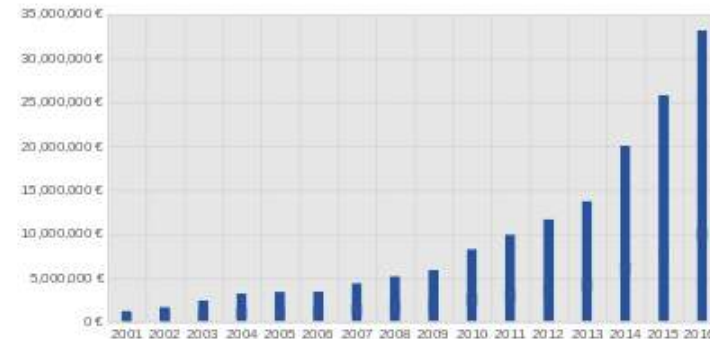
1-4 External Conferences per Year

Number of Employees



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

Sales Growth





@michaelhaeu

# Refactoring Architecture is hard!

Just rewrite!



@michaelhaeu

TNG



TECHNOLOGY  
CONSULTING



# Dubai 27 years ago



*travels*









# Case Study: Vodafone Kabeldeutschland

The screenshot shows the Vodafone website interface. At the top, there is a navigation bar with the Vodafone logo on the left and links for 'Privatkunden' and 'Geschäftskunden' on the right. Below this is a red navigation bar with links for 'Mobilfunk', 'Internet & Telefon', 'TV', 'Kombi-Pakete', 'Hilfe', and 'MeinVodafone'. A search bar is located on the right side of this bar. The main content area features a large image of a man and a woman looking at a tablet. Overlaid on this image is a circular graphic for the 'Red Internet & Phone 500 Cable' offer. The graphic displays the price '19.99 €' per month, with a note 'ab dem 13. Monat 19.99 €'. A red circular badge indicates a 'Bis zu 12 Freimonate' (up to 12 free months) promotion. To the left of the graphic, the text 'GigaSpeed' is prominently displayed, followed by 'Surfe mit bis zu 500 Mbit/s im Kabel-Glasfasernetz'. Below this text is a button labeled 'Zum Angebot'. At the bottom left, there is a small icon of a house with a Wi-Fi signal.

Privatkunden Geschäftskunden

Bestellung & Bestellung 0800 - 664 94 25

Kontakt Shopfinder

Mobilfunk Internet & Telefon TV Kombi-Pakete Hilfe MeinVodafone

Ich suche nach

**GigaSpeed**

Surfe mit bis zu 500 Mbit/s im Kabel-Glasfasernetz

**Red Internet & Phone 500 Cable**

**19.99 €** im Monat  
ab dem 13. Monat 19.99 €

Bis zu **12** Freimonate

Zum Angebot



A close-up photograph of a gift box. The box is made of brown cardboard and is wrapped with a patterned paper featuring green, blue, and white geometric shapes. A large, textured, golden-brown ribbon is tied around the box. A white tag with a gold border is attached to the ribbon. The tag has the text "Rewrite Just FOR YOU" written on it in a mix of bold and regular fonts. The background is a patterned paper with various geometric shapes and colors.

**Rewrite**  
Just  
**FOR**  
**YOU**





Revenue

Yc



Communication  
problems ...



A person is walking away from the camera down a long, narrow aisle in a large warehouse. The aisle is flanked by tall, neat stacks of wooden crates or boxes that reach up to the ceiling. The floor is a light-colored, polished concrete. The lighting is somewhat dim, with a brighter area at the end of the aisle where the person is walking. The overall atmosphere is one of a vast, organized space.

Lots of hidden business logic ...



Big bang integration ...









**We need a plan!**



# What do we want?

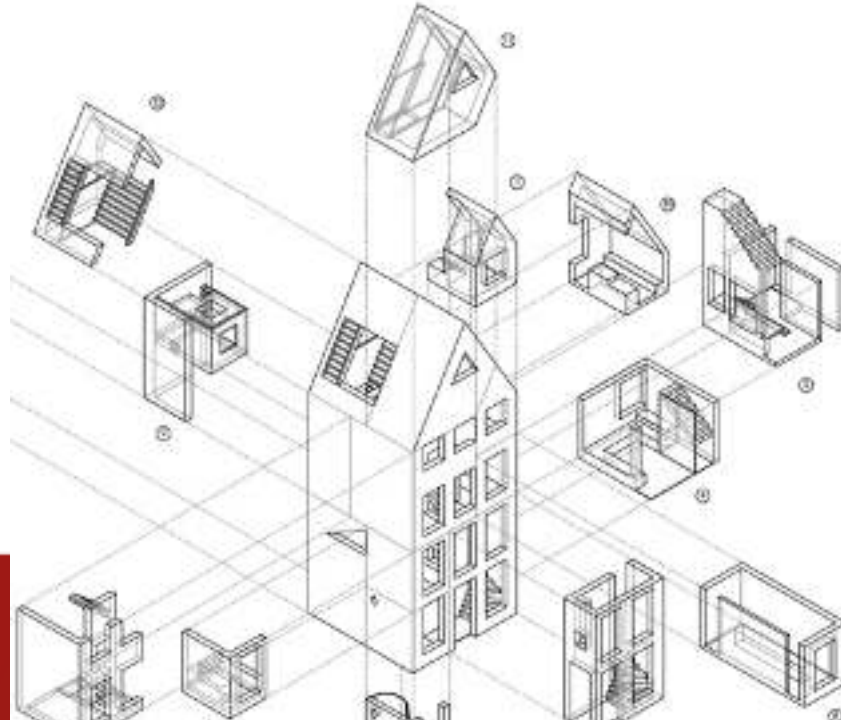
Architecture which ...

- 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 ...

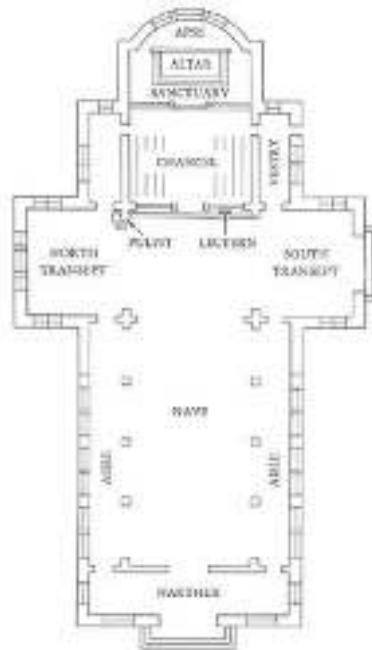
- ... about intent not about methods/tools
  - ... allows major decisions to be deferred
- "Architecture, The Lost Years", Robert C. Martin

# What's the intent?



@michaelhaeu

# What's the intent?



@michaelhaeu

TNO TECHNOLOGY CONSULTING





# What's the intent?

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



@michaelhaeu

# 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")*



@michaelhaeu

*Good architecture  
allows major decisions  
to be deferred.*

*- Robert C. Martin  
("Uncle Bob")*



# How do we find problems in our architecture?

- read ...
- read ...
- read ...
- read ...
- read ...



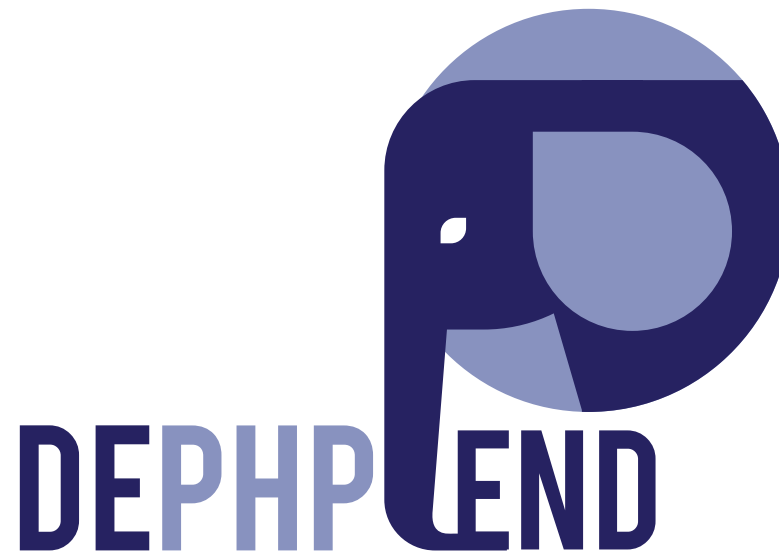


A collection of various tools is displayed on a blue, textured background. The tools include numerous open-end and combination wrenches of different sizes, several screwdrivers with red and black handles, a pair of green-handled pliers, a pair of green-handled scissors, a set of hex keys in a holder, and a large metal caliper. The tools are arranged in a somewhat organized manner, with wrenches at the top and bottom, and other tools in the middle and bottom right. A semi-transparent white rectangular box is overlaid in the center of the image, containing the text "We need some tools!".

**We need some tools!**



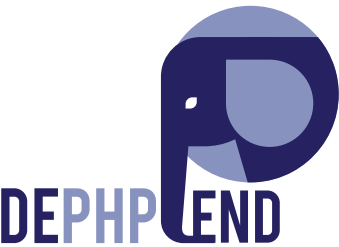
# dePHPend



@michaelhaeu

Visualizations & Assertions





```
λ wget http://phar.dephpend.com/dephpend.phar -O ~/bin/dephpend
```

```
λ dephpend --help
```

```

      _      _      _      _      _
      | |      | _ \ | | | | _ \      | |
      _| | _| |_) | | | | |_) | _ _ _ _| |
      / _ \ / _ \ _| | _ | _| _ \ _ \ / _ \
      | (| | _| | | | | | | | _| | | (| |
      \ _ \ \ _ \   | | | | \ _ \ | | \ _ \
      version 0.4

```

Usage: ...

```
λ dephpend text ~/workspace/dephpend/src
```

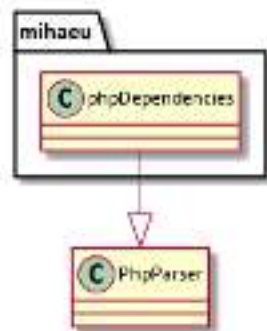
```
Mihaeu\PhpDependencies\Util\AbstractMap --> Mihaeu\PhpDependencies\Util\Collection
```

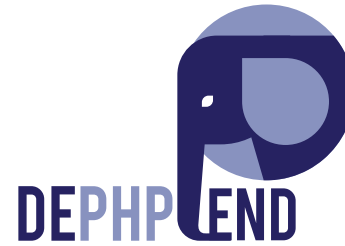
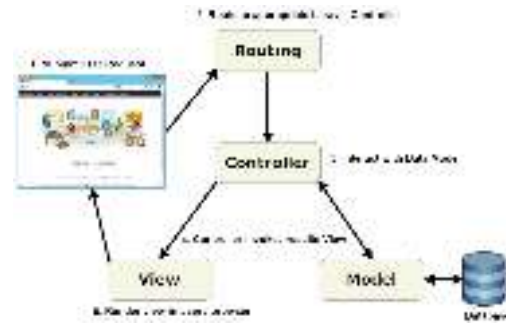
```
Mihaeu\PhpDependencies\Util\DI --> Mihaeu\PhpDependencies\Analyser\Analyser
```

• •



# Visualizations

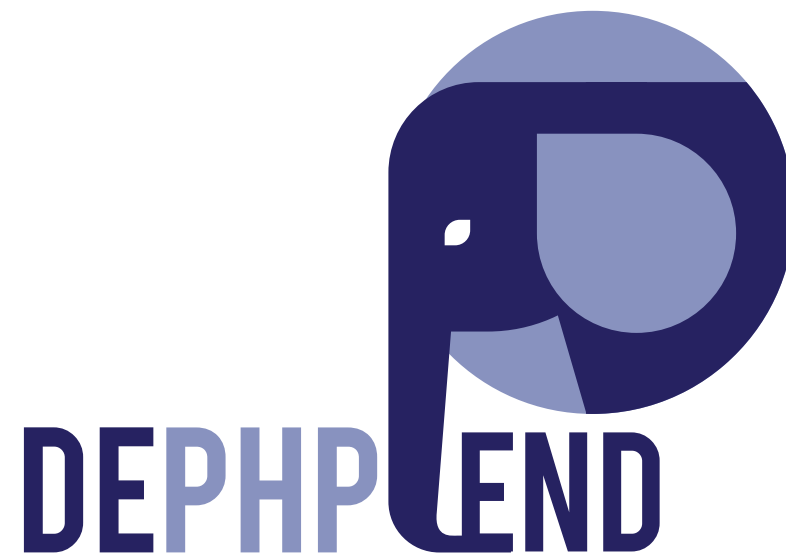




```
<?php

$cmd = shell_exec('dephend 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

TNG  TECHNOLOGY

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





# Metrics

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



# pmd and PhpInspectionseA to detect other architectural smells

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



@michaelhaeu

<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

Use a proper IDE!

# Refactoring Browser by Qafoo



The screenshot shows a web browser window displaying the project page for 'PHP Refactoring Browser'. The page has a dark header with the title 'PHP Refactoring Browser' in white. Below the header, there's a section with the title 'PHP Refactoring Browser' and a description: 'A command line refactoring tool for PHP'. There are three buttons: 'Download PHAR', 'View On GitHub', and 'Built by Qafoo'. A note states: '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.' A build status bar shows 'build passing' in green. The main content area describes the tool as '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.' It also mentions 'The library is standing on the shoulder of giants, using multiple existing libraries:' and lists 'PHP Parser by Nikic' as one of the dependencies.

## PHP Refactoring Browser

A command line refactoring tool for PHP

[View the Project on GitHub](#)  
Qafoo Labs / php-refactoring-browser

[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.

build **passing**

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.

The library is standing on the shoulder of giants, using multiple existing libraries:

- [PHP Parser](#) by Nikic



@michaelhaeu



# 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**





# 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?

# Or that?

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

- not much better
- most questions remain unanswered

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



# 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);
        }
    }
}
```

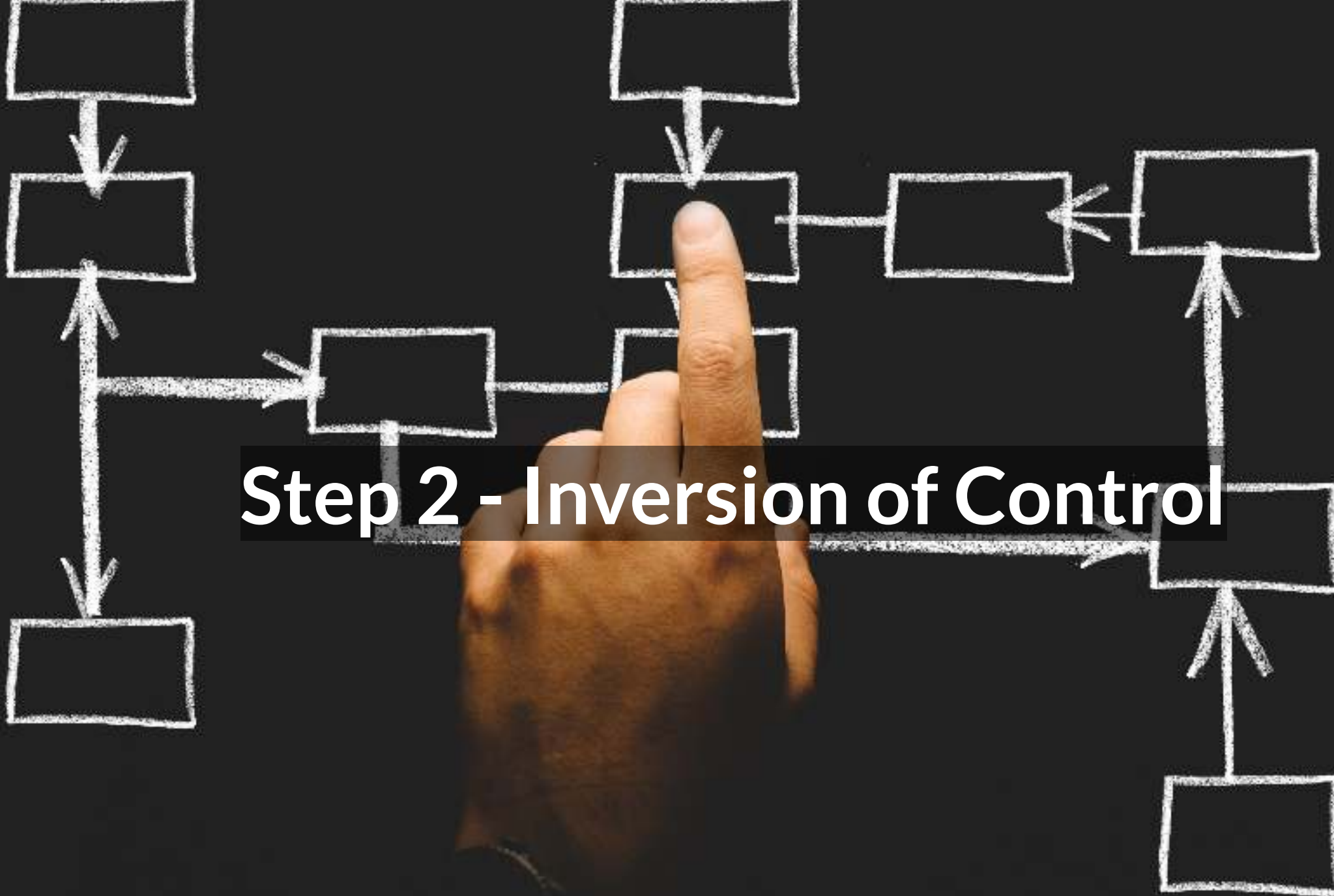




# Summary: Immutable Domain Objects

- 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 (\*)

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



# 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(** ... */);
    }
}
```

Don't make me think about stupid things!





# 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

# 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() { /** ... */ }
}
```

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





A close-up photograph of a spiral-bound sketchbook. In the center, a crumpled ball of blue paper sits on a page filled with various pencil sketches. These sketches include a large circle, several straight lines, a small star-like shape, and a large arrow pointing towards the bottom right. The text "Step 3 - Show Intent" is printed in white on a dark rectangular background, centered over the blue paper ball. A black pen is visible on the right side of the page, and the spiral binding of the notebook is on the left.

## 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)

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

# 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



@michaelhaeu

**TNG** TECHNOLOGY  
CONSULTING



A delivery person wearing a white helmet and dark clothing is riding a motorcycle on a city street. A large, wrapped package is secured to the back of the motorcycle. The background is blurred, showing buildings and a road, suggesting motion. A semi-transparent dark box with white text is overlaid on the center of the image.

**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, ...)





# 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"?





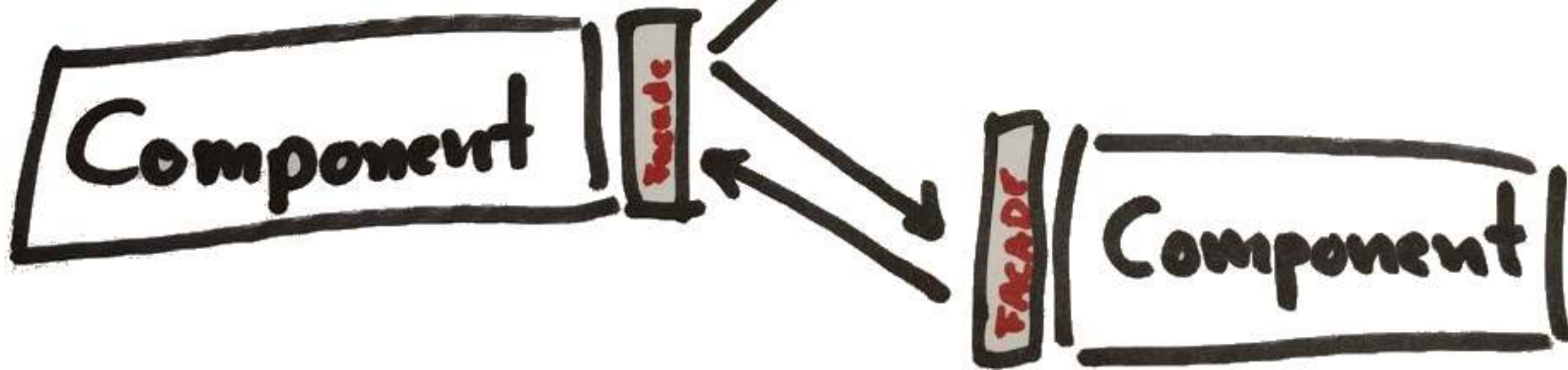
Let the fun begin!

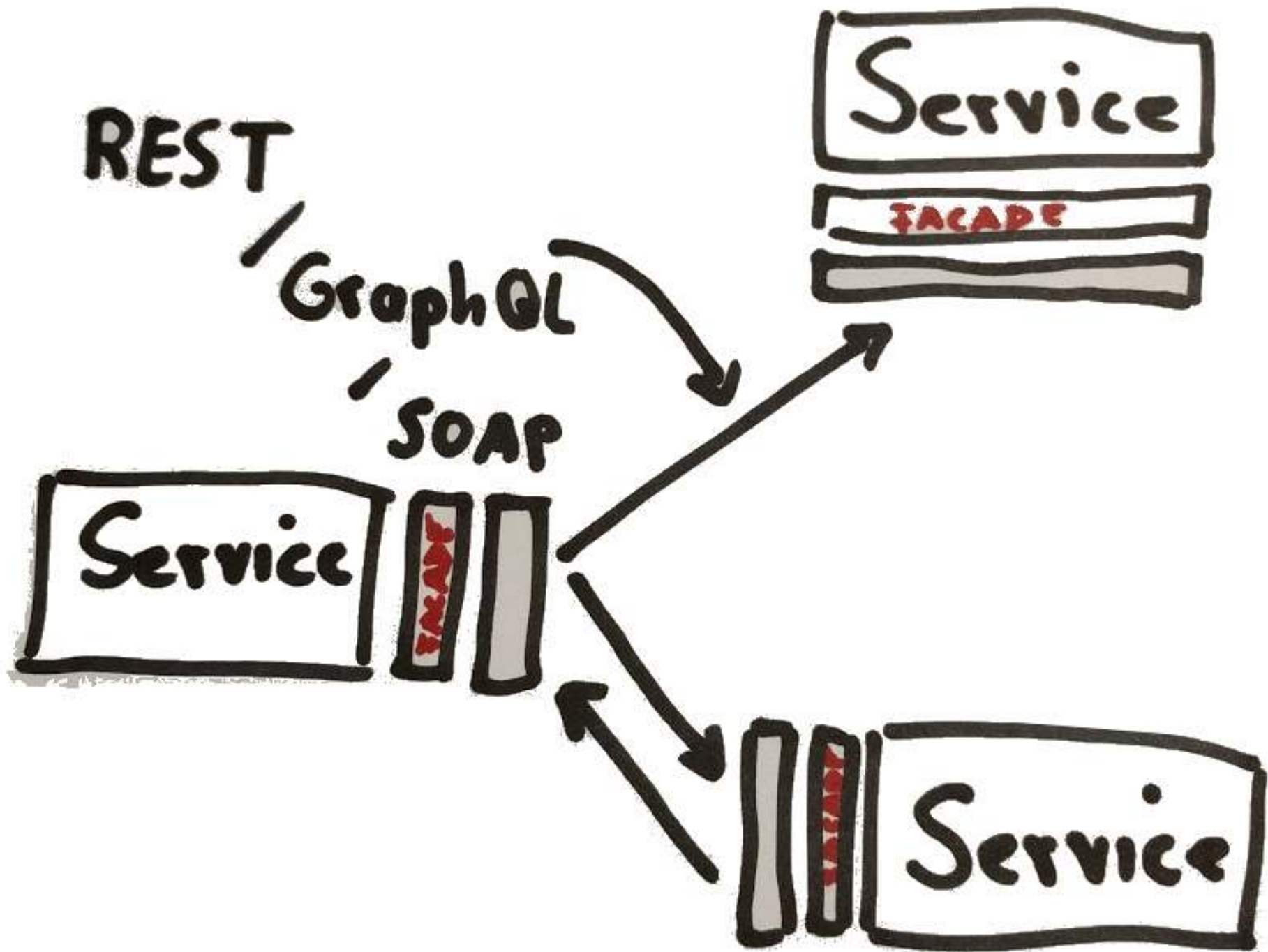
# Microservices

- 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)



"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.



# 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 several Marines in camouflage uniforms standing in a line. The focus is on a Marine in the foreground wearing a camouflage cap, looking towards the right. Other Marines are visible in the background, also in formation. The image is used as a background for a title slide.

# Optional Step 5 -

## CQRS

Command Query  
Responsibility Segregation

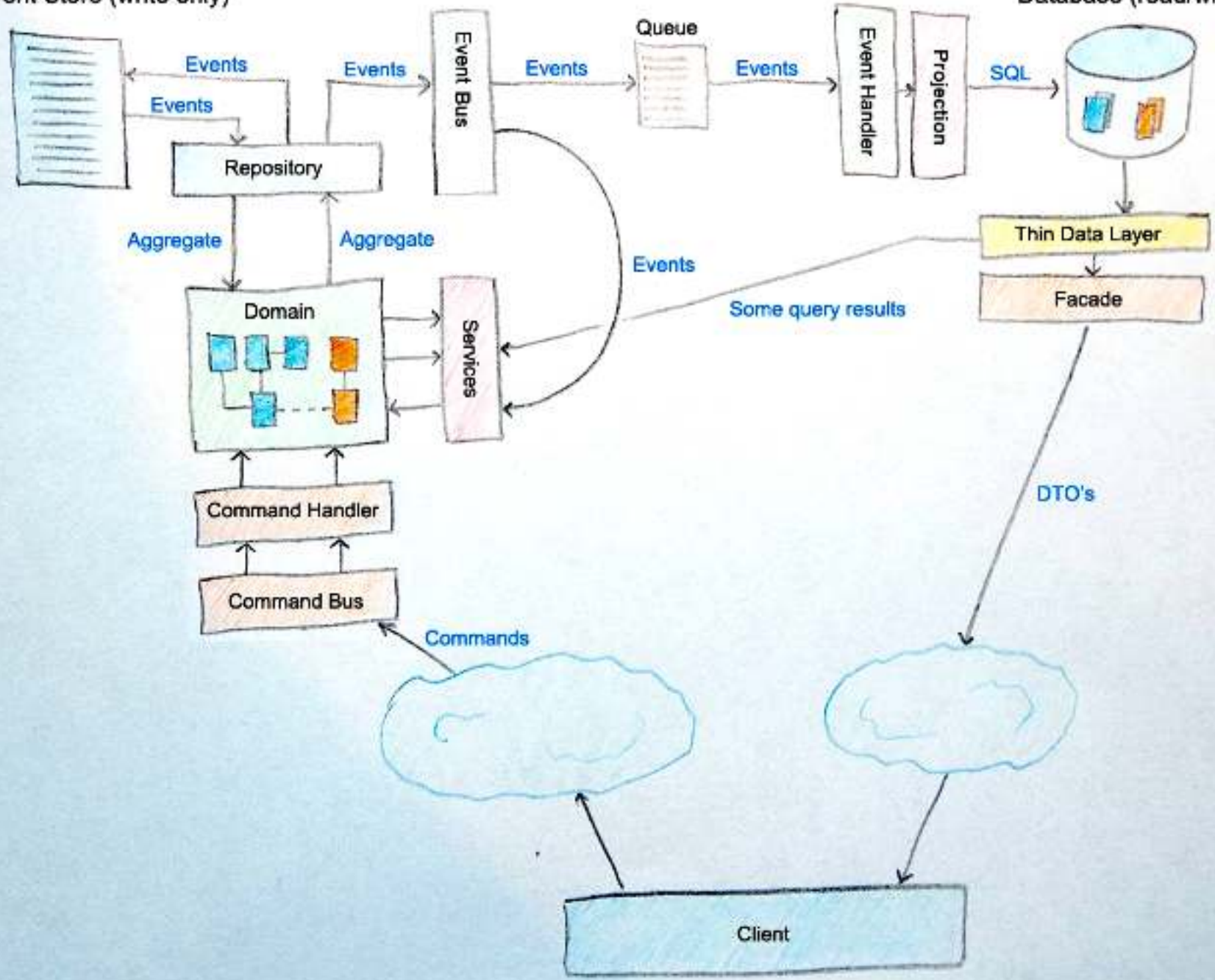
# Event Sourcing

- 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



Event Store (write only)

Database (read/write)



# 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



# Reading Recommendations



Modernizing  
Legacy  
Applications in  
PHP

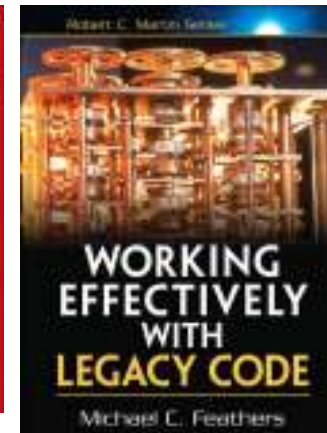
(Paul M. Jones)



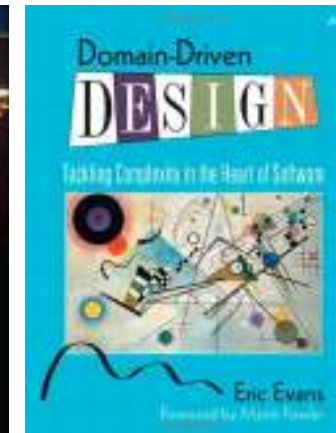
Clean  
Architecture  
(Robert C. Martin)



Patterns of  
Enterprise  
Application  
Architecture  
(Martin Fowler)



Working  
Effectively with  
Legacy Code  
(Michael C. Feathers)



Domain Driven  
Design  
(Eric Evans)



@michaelhaeu

TNGE TECHNOLOGY CONSULTING





# Thank you!

Michael Haeuslmann - @michaelhaeu





# Credits

Stock photos from pexels.com

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

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/coup>