Clean Code

Symptoms, Treatment, and Vaccination of Bad Code

Markus Poerschke – #t3cvie – 04. - 05.09.2020

Clean Code

Symptoms, Treatment, and Vaccination of Bad Code

Markus Poerschke – #t3cvie – 04. - 05.09.2020

Agenda

What we will talk about…

- Symptoms
- Code Examples (PHP and TYPO3)
  - Object Orientated Programming (OOP)
  - Functions
  - Code Smell

NOT Agenda

What we will NOT talk about…

- Tools
- Organisational Changes
- Naming Things (there is enough to say for an own session)
- Code Style (use PHP CS Fixer)

About the Speaker

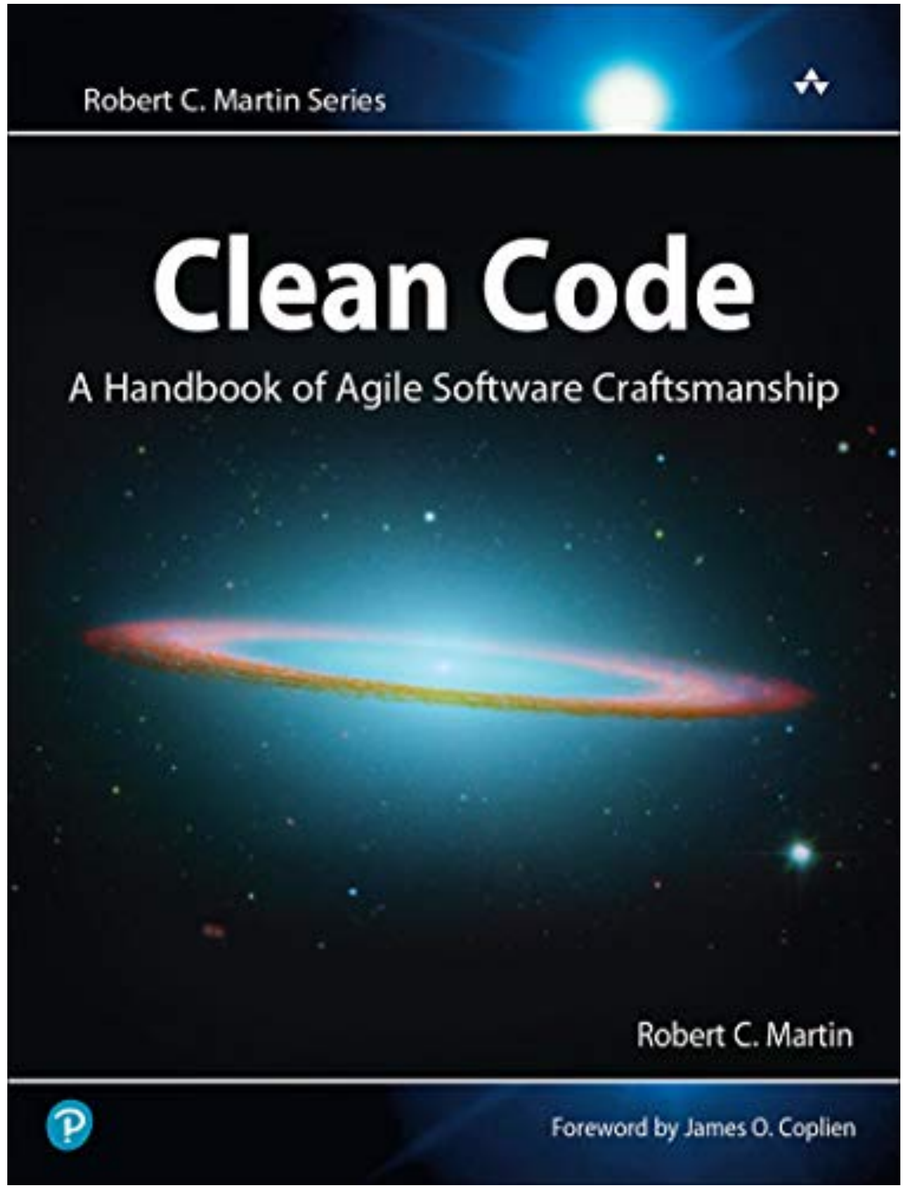
Markus Poerschke

- Working as a Software Engineer since 2010
- Working with mostly PHP, Symfony and TYPO3
- https://markus.poerschke.nrw

Clean Code

Robert C. Martin

The quotes in this presentation are from the book “Clean Code” by Robert C. Martin.



Symptoms

Rigid

(engl. synonym = inflexible, dt. = starr, steif, unbiegsam)

"Every change forces
a cascade of related changes."

Fragile

“Each change breaks distant
and apparently unrelated things.”

Immobile

“The code is hopelessly entangled;
reuse is impossible.”

Viscous

“Behaving badly is the
most attractive alternative.”

Object Orientated Programming (OOP)

SOLID

Single Responsibility

Open-Close

Liskov Substitution

Interface Segregation

Dependency Inversion

Single Responsibility

“A class should should
serve a single purpose.”

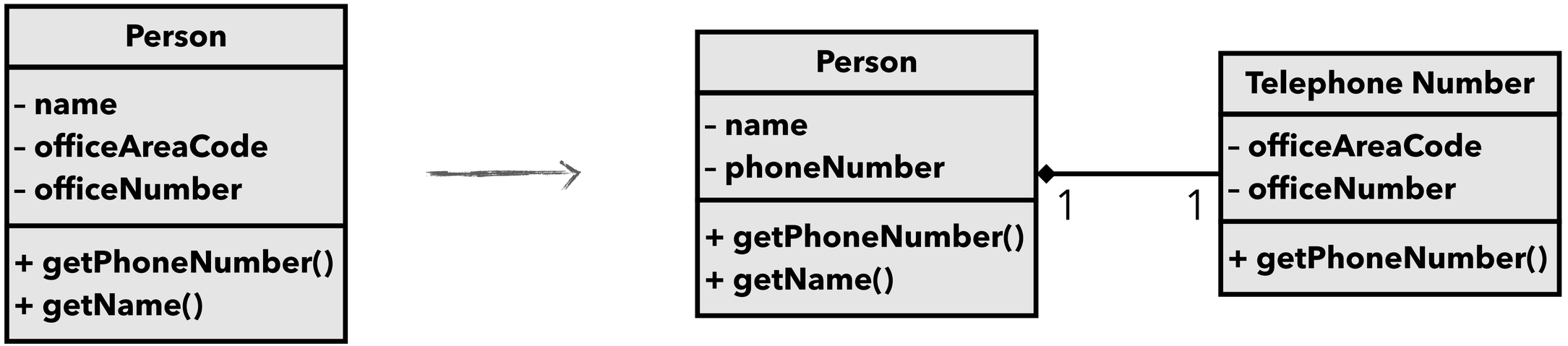
Single Responsibility

Quelle: https://www.amazon.de/Wenger-Schweizer-Offiziersmesser-Messer-Schatulle/dp/BoooRoJDSI

Single Responsibility

GeneralUtility

Single Responsibility



Open-Close Principle

“A module should be open for extension,
but closed for modifiction.”

Open-Close Principle

TYPO3 EXAMPLE

- Avoid XCLASS
- Use
  - Official Extension Points
  - Event-Listeners
  - HTTP-Middleware
  - Symfony DIC (e.g. Decorating Pattern)

Leskov Substitution

"Subclasses should be
substitutable for their base classes."

Interface Segregation

“Many client specific interfaces are better than one general purpose interface.”

Dependency Inversion

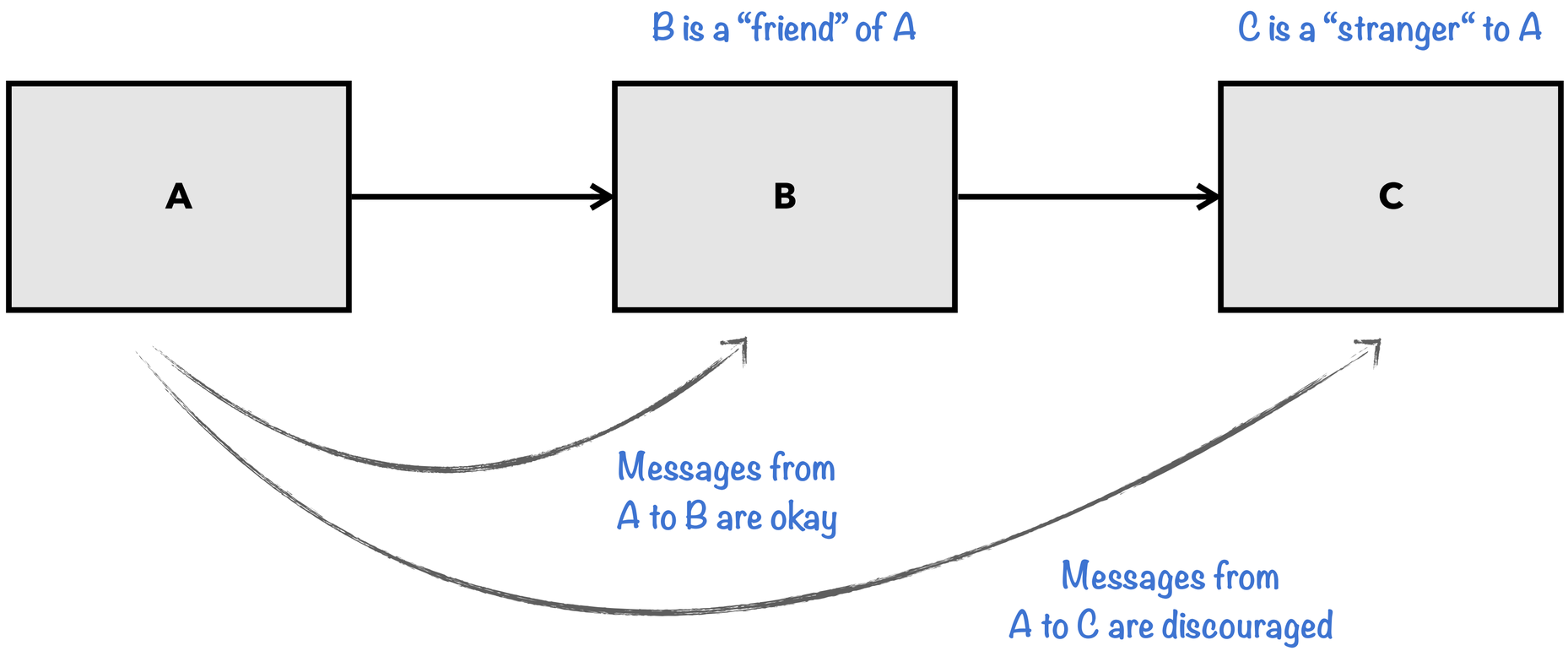
“Depend upon abstractions,
do not depend upon concretions.”

Dependency Inversion

An Extbase controller depends implicitly on a FluidView.

Law of Demeter

Law of Demeter



“If I'm asking you to give me $20, I'm interacting with your public interface. If I'm asking you to get your wallet out of your pocket, open it, pull a $20 note, and put it in my hand, I'm getting way deep into your private implementation. What if you don't have a wallet? Oops.”

Twitter: @abdurrakhimov

“If I'm asking you to give me $20, I'm interacting with your public interface. If I'm asking you to get your wallet out of your pocket, open it, pull a $20 note, and put it in my hand, I'm getting way deep into your private implementation. What if you don't have a wallet? Oops.”

Twitter: @abdurrakhimov

“If I'm asking you to give me $20, I'm interacting with your public interface. If I'm asking you to get your wallet out of your pocket, open it, pull a $20 note, and put it in my hand, I'm getting way deep into your private implementation. What if you don't have a wallet? Oops.”

Twitter: @abdurrakhimov

“If I'm asking you to give me $20, I'm interacting with your public interface. If I'm asking you to get your wallet out of your pocket, open it, pull a $20 note, and put it in my hand, I'm getting way deep into your private implementation. What if you don't have a wallet? Oops.”

Twitter: @abdurrakhimov

BAD

```
$customer->getWallet()->getMoney()->take(20);
```

GOOD

```
$customer->pay(20);
```

Immutable Objects

“An **immutable object** is an object whose state cannot be changed after it is created.“

“This is in contrast to a **mutable object**, which can be modified after it is created.“

BAD

```php
<?php

function tomorrows_day_name(\DateTime $today)
{
    $today->add(new \DateInterval('P1D'));
    return $today->format('l');
}

$today = new \DateTime();
echo  "Tomorrow is ".tomorrows_day_name($today)
     .", today is ".$today->format('l');
```

as a side effect of the function,
the state of $today is changed

Output: Tomorrow is Saturday, today is Saturday

GOOD

```php
<?php

function tomorrows_day_name(\DateTimeImmutable $today)
{
    $tomorrow = $today->add(new \DateInterval('P1D'));
    return $tomorrow->format('l');
}

$today = new \DateTimeImmutable();
echo  "Tomorrow is ".tomorrows_day_name($today)
     .", today is ".$today->format('l');
```

PHP 8

```php
<?php

function tomorrows_day_name(\DateTimeInterface $today)
{
    $today = \DateTimeImmutable::createFromInterface($today);
    $tomorrow = $today->add(new \DateInterval('P1D'));
    return $tomorrow->format('l');
}

$today = new \DateTime();
echo  "Tomorrow is ".tomorrows_day_name($today)
     ., today is ".$today->format('l');
```

Functions

“Functions should do one thing.
They should do it well. They should do it only.“

“A **pure function** is a function where the return value is only determined by its input value, without observable side effects.“

Pure

```
function f(int $a, int $b): int
{
    return $a + $b;
}
```

Impure

```php
function f(int $a): int
{
    static $b = 0;
    $b += $a;

    return $b;
}
```

mutation of a local static variable

Impure

```
function f(string $fileName): string
{
    return (string) file_get_contents($fileName);
}
```

reading from I/O device

Impure

```php
$b = 123;

function f(int $a): string
{
    global $b;
    $b += $a;

    return $b;
}
```

mutating non-local variable

Impure

```php
class Person
{
    private $name = '';

    public function setName(string $name): void
    {
        $this->name = $name;
    }

    public function getName(): string
    {
        return $this->name;
    }
}
```

side effect: object state is changed

mutable reference variable

Impure

```
$contentObject = GeneralUtility::makeInstance(ContentObjectRenderer::class);
$contentObject->typoLink('', []);

return $contentObject->lastTypoLinkUrl;
```

TYPO3 EXAMPLE

Function Arguments

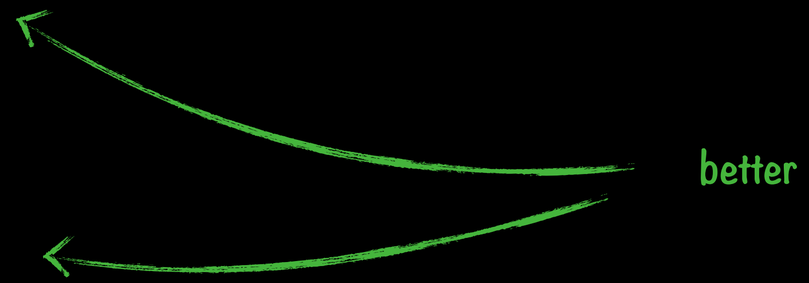
```
function makeCircle(float $x, float $y, float $radius);

function makeCircle(Point $center, float $radius);
```

```
function sum(int $z1, int $z2, int $z3);
sum(1, 2, 3); // max 3 numbers can be summed

function sum(array $summands);
sum([1, 2, 3, 4]);

function sum(int ...$summands);
sum(1, 2, 3, 4, 5, 6);
sum(...[1, 2, 3, 4]);
```



Side Effects

BAD

```
class UserValidator
{
    public function checkPassword(
        string $userName,
        string $password
    ): bool
    {
        $user = $this->findUserByName($userName);
        if ($user !== null) {
            $hashedPassword = $user->getPassword();
            if ($hashedPassword === $this->hashPassword($password)) {
                $_SESSION['LOGGED_IN'] = true;
                return true;
            }
        }

        return false;
    }
}
```

A GOOD START

```
class UserValidator
{
    public function checkPasswordAndInitializeSession(
        string $userName,
        string $password
    ): bool
    {
        $user = $this->findUserByName($userName);
        if ($user !== null) {
            $hashedPassword = $user->getPassword();
            if ($hashedPassword === $this->hashPassword($password)) {
                $_SESSION['LOGGED_IN'] = true;
                return true;
            }
        }

        return false;
    }
}
```

BETTER

```
if ($userValidator->checkPassword($userName, $password)) {
    $_SESSION['LOGGED_IN'] = true;
}
```

Command Query Separation

“Functions should either do something or answer something, but not both.“

“Either your function should
change the state of an object, or it should
return some information about that object.“

Code Smell

Useless Code I: Commented Code

BAD

```
function indexAction()
{
    // $repo = $this->getRepository();
    // $list = $repository->getList();
    // var_dump($list);

    $list = $this->getList();
    return $this->render(
        'index.html.twig',
        ['list' => $list]
    );
}
```

BETTER

```
function indexAction()
{
    $list = $this->getList();
    return $this->render(
        'index.html.twig',
        ['list' => $list]
    );
}
```

Useless Code II: Dead Code

BAD

```
function indexAction()
{
    $list = $this->getList();
    return $this->render(
        'index.html.twig',
        ['list' => $list]
    );

    $repo = $this->getRepository();
    $list = $repository->getList();
    var_dump($list);
}
```

this code is unreachable because
the function exited with a return already

Useless Code III: Useless Code

BAD

```
function indexAction()
{
    $list = new Array();
    $list = $this->getList();
    return $this->render(
        'index.html.twig',
        ['list' => $list]
    );
}
```

Magic Numbers

BAD

```
function articleAction(BlogArticle $article)
{
    if ($article->getStatus() !== 42) {
        throw new HttpNotFoundException();
    }

    // ...
}
```

what is the meaning of 42?

BETTER

```
function articleAction(BlogArticle $article)
{
    if ($article->getStatus() !== BlogArticle::PUBLISHED) {
        throw new HttpNotFoundException();
    }

    // ...
}
```

BETTER

```
function articleAction(BlogArticle $article)
{
    if ($article->isPublished() === false) {
        throw new HttpNotFoundException();
    }

    // ...
}
```

```
class Article
{
    const STATUS_DRAFT = 0;
    const STATUS_PUBLISHED = 1;

    private $status;

    public function setStatus($status)
    {
        $this->status = $status;
    }
}

$article->setStatus("draft");
```

no error validation

```php
class Article
{
    const STATUS_DRAFT = 0;
    const STATUS_PUBLISHED = 1;

    private $status;

    public function setStatus(int $status)
    {
        $this->status = $status;
    }
}

$article->setStatus("draft");
```

error, because status must be int

```php
class Article
{
    const STATUS_DRAFT = 0;
    const STATUS_PUBLISHED = 1;

    private $status;

    public function setStatus(int $status)
    {
        $this->status = $status;
    }
}

$article->setStatus(404);
```

404 is an invalid value

BETTER

```
class Article
{
    private Status $status;

    public function setStatus(Status $status)
    {
        $this->status = $status;
    }
}

$article->setStatus(Status::DRAFT());
```

```
class Status extends Enum
{
    private const DRAFT = 'draft';
    private const PUBLISHED = 'published';
}

$status = Status::DRAFT();
$status === Status::DRAFT(); // true
$status === Status::PUBLISHED(); // false
```

BETTER

```
function articleAction(BlogArticle $article)
{
    if ($article->getStatus() !== Status::PUBLISHED()) {
        throw new HttpNotFoundException();
    }

    // ...
}
```

Summary

Summary

- Stick to the SOLID principles
- Don’t grab into strangers wallet
- Use Enums or Constants instead of Magic Numbers
- Use immutable objects if possible (avoid mutable objects)
- Avoid impure functions / methods (if possible)
- Delete useless code
- Apply the Command Query Separation principle

Thanks for you attention!
Questions?