Das macht man heute so

Sebastian Bergmann

Sebastian Bergmann

sharing experience

thePHP.cc

Wie hat sich die Art, wie wir programmieren, geändert?

```
init:
    move #$ac,d7
    move #1,d6
mainloop:
wframe:
    cmp.b #$ff,$dff006
    bne wframe
    add d6,d7
    cmp #$f0,d7
    blo ok1
    neg d6
ok1:
    cmp.b #$40,d7
    bhi ok2
    neg d6
ok2:
waitras1:
    cmp.b $dff006,d7
    bne waitras1
    move.w #$fff,$dff180
waitras2:
    cmp.b $dff006,d7
    beq waitras2
    move.w #$116,$dff180
    btst #6,$bfe001
    bne mainloop
    rts
```

Wir überspringen ein paar Jahre …

```php
<?php
require __DIR__ . '/includes/config.php';
require __DIR__ . '/includes/db.php';

$result = $DB->query('
    SELECT *
      FROM auftrag, kunde
     WHERE auftrag.kunden_id = kunde.kunden_id
       AND auftrag.datum BETWEEN "' . $_GET['jahr'] . '-01-01" AND "' .
                                   $_GET['jahr'] . '-12-31";'
);

$auftraege = [];

foreach ($result as $row) {
    $auftraege[] = [
        'Auftragsnummer' => $row['auftrag_id'],
        'Datum' => (new DateTimeImmutable($row['datum']))->format('d.m.Y'),
        'Auftraggeber' => $row['name'] . ', ' . $row['anschrift']
    ];
}

header('Content-Type: application/json; charset=utf-8');
print json_encode($auftraege) . PHP_EOL;
```

- Prozedurales Skript, das von "oben nach unten" ausgeführt wird

- Prozedurales Skript, das von "oben nach unten" ausgeführt wird
- Konfiguration, Datenbankverbindung, alle Variablen im globalen Scope

- Prozedurales Skript, das von "oben nach unten" ausgeführt wird
- Konfiguration, Datenbankverbindung, alle Variablen im globalen Scope
- Request-Verarbeitung, Persistenz, Geschäftslogik und Darstellung nicht von einander getrennt

Wir refaktorieren ein paar Stunden …

```php
<?php declare(strict_types=1);
require __DIR__ . '/../src/autoload.php';

$factory = new Factory;

$request = Request::fromSuperglobals();

$action = $factory->getOrderListAction();

$action->execute($request)->send();
```

https://thephp.cc/schulungen/alten-code-angstfrei-aendern

Wie hat sich die Art, wie wir programmieren, geändert?

Wie hat sich die Art, wie wir programmieren, geändert?

- Prozedural → Objekt-Orientiert

Wie hat sich die Art, wie wir programmieren, geändert?

- Prozedural → Objekt-Orientiert
- Global State → Object State

Wie hat sich die Art, wie wir programmieren, geändert?

- Prozedural → Objekt-Orientiert
- Global State → Object State
- Favour immutable state over mutable state

Was hat sich bei PHP die letzten Jahre getan?

Was hat sich bei PHP die letzten Jahre getan?

- Schneller!

Was hat sich bei PHP die letzten Jahre getan?

- Schneller!
- Weniger Strom

Was hat sich bei PHP die letzten Jahre getan?

- Schneller!
- Weniger Strom
- (Mehr) Typsicherheit

Typsystem

Wer erinnert sich noch an PHP 4?

```php
class Money
{
    var $amount;

    function Money($amount)
    {
        $this->amount = $amount;
    }

    function amount()
    {
        return $this->amount;
    }

    function add($other)
    {
        return new Money($this->amount + $other->amount());
    }
}
```

```php
class Money
{
    var $amount;

    function Money($amount)
    {
        if (!is_int($amount)) {
            trigger_error(
                'Passed argument is not an integer',
                E_USER_ERROR
            );
        }

        $this->amount = $amount;
    }

    function amount()
    {
        return $this->amount;
    }

    function add($other)
    {
        return new Money($this->amount + $other->amount());
    }
}
```

```php
class Money
{
    var $amount;

    function Money($amount)
    {
        $this->amount = $amount;
    }

    function amount()
    {
        return $this->amount;
    }

    function add($other)
    {
        if (!is_a($other, 'Money')) {
            trigger_error(
                'Passed argument is not a Money object',
                E_USER_ERROR
            );
        }

        return new Money($this->amount + $other->amount());
    }
}
```

PHP 5

```php
final class Money
{
    private $amount;

    public function __construct($amount)
    {
        $this->amount = $amount;
    }

    public function amount()
    {
        return $this->amount;
    }

    public function add(self $other)
    {
        return new Money($this->amount + $other->amount());
    }
}
```

```php
final class Money
{
    private $amount;

    public function __construct($amount)
    {
        if (!is_int($amount)) {
            throw new InvalidArgumentException(
                'Passed argument is not an integer'
            );
        }

        $this->amount = $amount;
    }

    public function amount()
    {
        return $this->amount;
    }

    public function add(self $other)
    {
        return new Money($this->amount + $other->amount());
    }
}
```

PHP 7

```php
final class Money
{
    private $amount;

    public function __construct(int $amount)
    {
        $this->amount = $amount;
    }

    public function amount(): int
    {
        return $this->amount;
    }

    public function add(self $other): self
    {
        return new Money($this->amount + $other->amount());
    }
}
```

PHP 7.4

```php
final class Money
{
    private int $amount;

    public function __construct(int $amount)
    {
        $this->amount = $amount;
    }

    public function amount(): int
    {
        return $this->amount;
    }

    public function add(self $other): self
    {
        return new Money($this->amount + $other->amount());
    }
}
```

Was fehlt noch?

Typ-Deklaration für lokale Variablen

Typisierte Arrays

Typisierte Arrays

wären schön, Collections tun es aber auch

```php
final class MoneyCollection implements \IteratorAggregate
{
    /** @var Money[] */
    private $items = [];

    public static function __construct(Money ...$items): self
    {
        $this->items = $items;
    }

    public function add(Money $item): void
    {
        $this->items[] = $item;
    }

    /** @return Money[] */
    public function asArray(): array
    {
        return $this->items;
    }

    public function getIterator(): MoneyCollectionIterator
    {
        return new MoneyCollectionIterator($this);
    }
}
```

```php
final class MoneyCollectionIterator implements \Iterator {
    private $items;
    private $position;

    public function __construct(MoneyCollection $collection) {
        $this->items = $collection->asArray();
    }

    public function rewind(): void {
        $this->position = 0;
    }

    public function valid(): bool {
        return $this->position < \count($this->items);
    }

    public function key(): int {
        return $this->position;
    }

    public function current(): Money {
        return $this->items[$this->position];
    }

    public function next(): void {
        $this->position++;
    }
}
```

sebastianbergmann/shaku

Was fehlt noch?

Generics

Union Types

In den PHP 7-Beispielen habe ich `declare(strict_types=1);` aus Platzgründen weggelassen.

Es gibt keinen Grund, in neuem Code auf die strikte Interpretation skalarer Typdeklarationen zu verzichten.

Type Checker

Psalm

```php
<?php
function foo(string $s): void {
    return 'bar';
}

$a = ['hello', 5];
foo($a[1]);
foo();

if (rand(0, 1)) $b = 5;
echo $b;

$c = rand(0, 5);
if ($c) {} elseif ($c) {}
```

```
ERROR: InvalidReturnStatement - 3:12 - No return values are expected for foo
INFO:  UnusedParam - 2:21 - Param $s is never referenced in this method
ERROR: InvalidReturnType - 2:26 - The declared return type 'void' for foo is incorrect, got 'string'
ERROR: InvalidScalarArgument - 7:5 - Argument 1 of foo expects string, int(5) provided
ERROR: TooFewArguments - 8:1 - Too few arguments for method foo - expecting 1 but saw 0
INFO:  PossiblyUndefinedGlobalVariable - 11:6 - Possibly undefined global variable $b,
       first seen on line 10
ERROR: TypeDoesNotContainType - 14:20 - Found a contradiction when evaluating $c and
       trying to reconcile type 'int(0)' to !falsy
```

```php
<?php declare(strict_types=1);
namespace PHPUnit\Util\Annotation;

final class DocBlock
{
    // ...

    /** @var array<string, array<int, string>> */
    private $symbolAnnotations;

    // ...
}
```

```php
<?php declare(strict_types=1);
namespace PHPUnit\Util;

final class Test
{
  // ...
  public static function getMissingRequirements(string $className, string $methodName): array
  {
    // ...
    if (!empty($required['PHP'])) {
      $operator = empty($required['PHP']['operator']) ? '>=' : $required['PHP']['operator'];

      if (!\version_compare(\PHP_VERSION, $required['PHP']['version'], $operator)) {
        $missing[] = \sprintf('PHP %s %s is required.', $operator, $required['PHP']['version']);
        $hint      = $hint ?? 'PHP';
      }
    } elseif (!empty($required['PHP_constraint'])) {
      // ...
  }
  // ...
}
```

```
ERROR: InvalidArgument - src/Util/Test.php:303:78 -
Argument 3 of version_compare expects string(\x3c)|string(lt)|string(\x3c=)|string(le)|
string(\x3e)|string(gt)|string(\x3e=)|string(ge)|string(==)|string(=)|string(eq)|
string(!=)|string(\x3c\x3e)|string(ne), string(>=)|mixed provided
```

```
--- a/src/Util/Test.php
+++ b/src/Util/Test.php
@@ -300,6 +301,8 @@ public static function getMissingRequirements(string $className, string $met
         if (!empty($required['PHP'])) {
             $operator = empty($required['PHP']['operator']) ? '>=' : $required['PHP']['operator

+            self::ensureOperatorIsValid($operator);
+
             if (!\version_compare(\PHP_VERSION, $required['PHP']['version'], $operator)) {
                 $missing[] = \sprintf('PHP %s %s is required.', $operator, $required['PHP']['ve
                 $hint      = $hint ?? 'PHP';
@@ -1279,4 +1286,19 @@ private static function shouldCoversAnnotationBeUsed(array $annotations):

         return true;
     }
+
+    private static function ensureOperatorIsValid(string $operator): void
+    {
+        if (!\in_array($operator, ['<', 'lt', '<=', 'le', '>', 'gt', '>=', 'ge', '==', '=', 'eq
+            throw new Exception(
+                \sprintf(
+                    '"%s" is not a valid version_compare() operator',
+                    $operator
+                )
+            );
+        }
+    }
 }
```

```php
<?php declare(strict_types=1);
/** @psalm-pure */
function f(int $x): int
{
    /** @var int $i */
    static $i = 0;

    $i += $x;

    return $i;
}
```

```
ERROR: ImpurePureStaticVariable - src/test.php:6:5 -
Cannot use a static variable in a mutation-free context
```

```php
<?php declare(strict_types=1);
/** @psalm-pure */
function f(int $x): int
{
    fopen(__FILE__, 'r');

    return $x;
}
```

```
ERROR: ImpureFunctionCall - src/test.php:5:5 -
Cannot call an impure function from a mutation-free context
fopen(__FILE__, 'r');
```

```php
<?php declare(strict_types=1);
/** @psalm-immutable */
final class Money
{
    private int $amount;

    public function __construct(int $amount)
    {
        $this->amount = $amount;
    }

    public function amount(): int
    {
        return $this->amount;
    }

    public function setAmount(int $amount)
    {
        $this->amount = $amount;
    }
}
```

```
ERROR: InaccessibleProperty - src/Money.php:21:9 -
Money::$amount is marked readonly
$this->amount = $amount;
```

Tests

Richtig testen

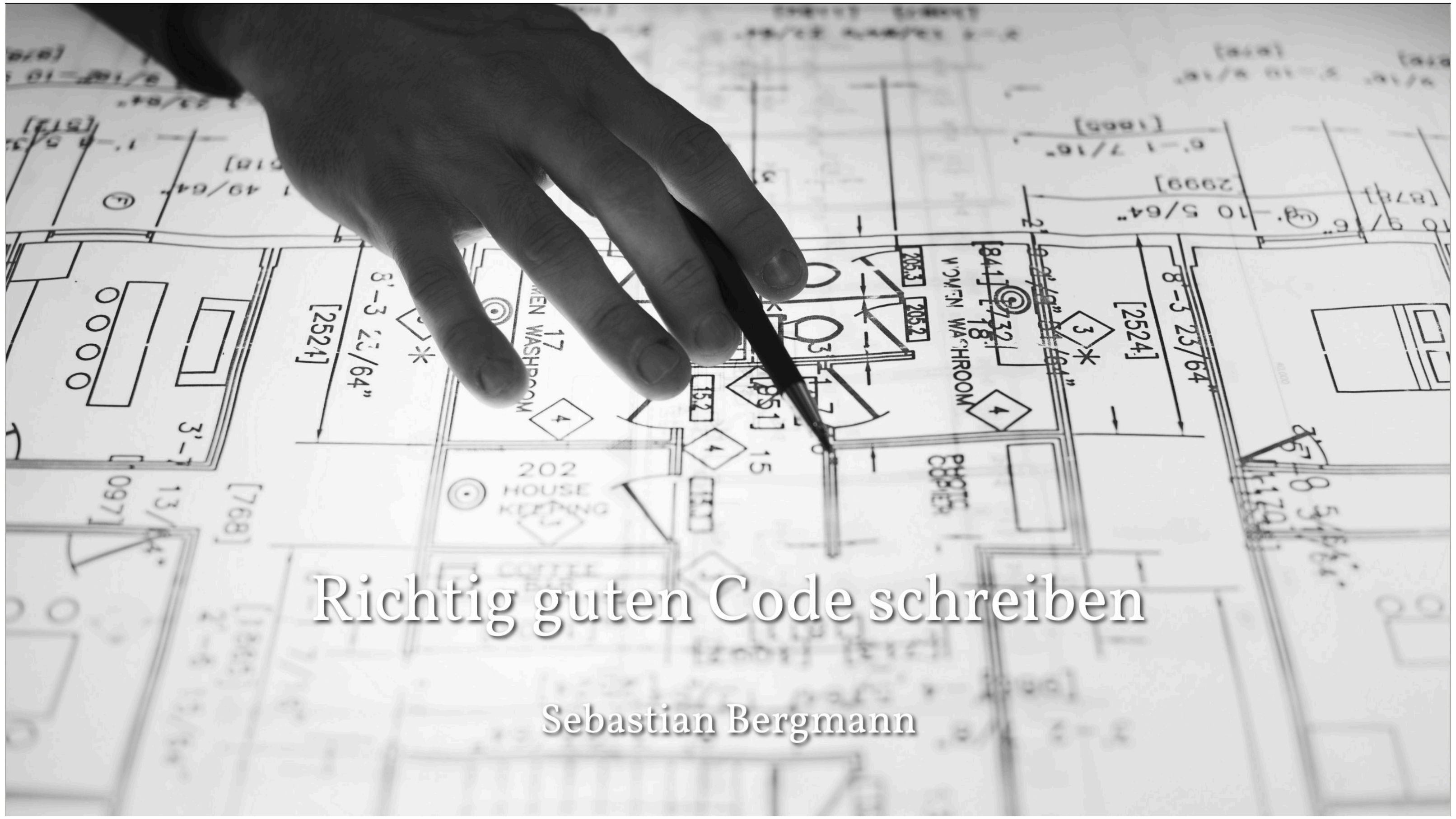
Sebastian Bergmann

27. Juni 2019

https://thephp.cc/termine/2019/06/devtreff-siegburg/richtig-testen

Unit Tests

Test-Driven Development



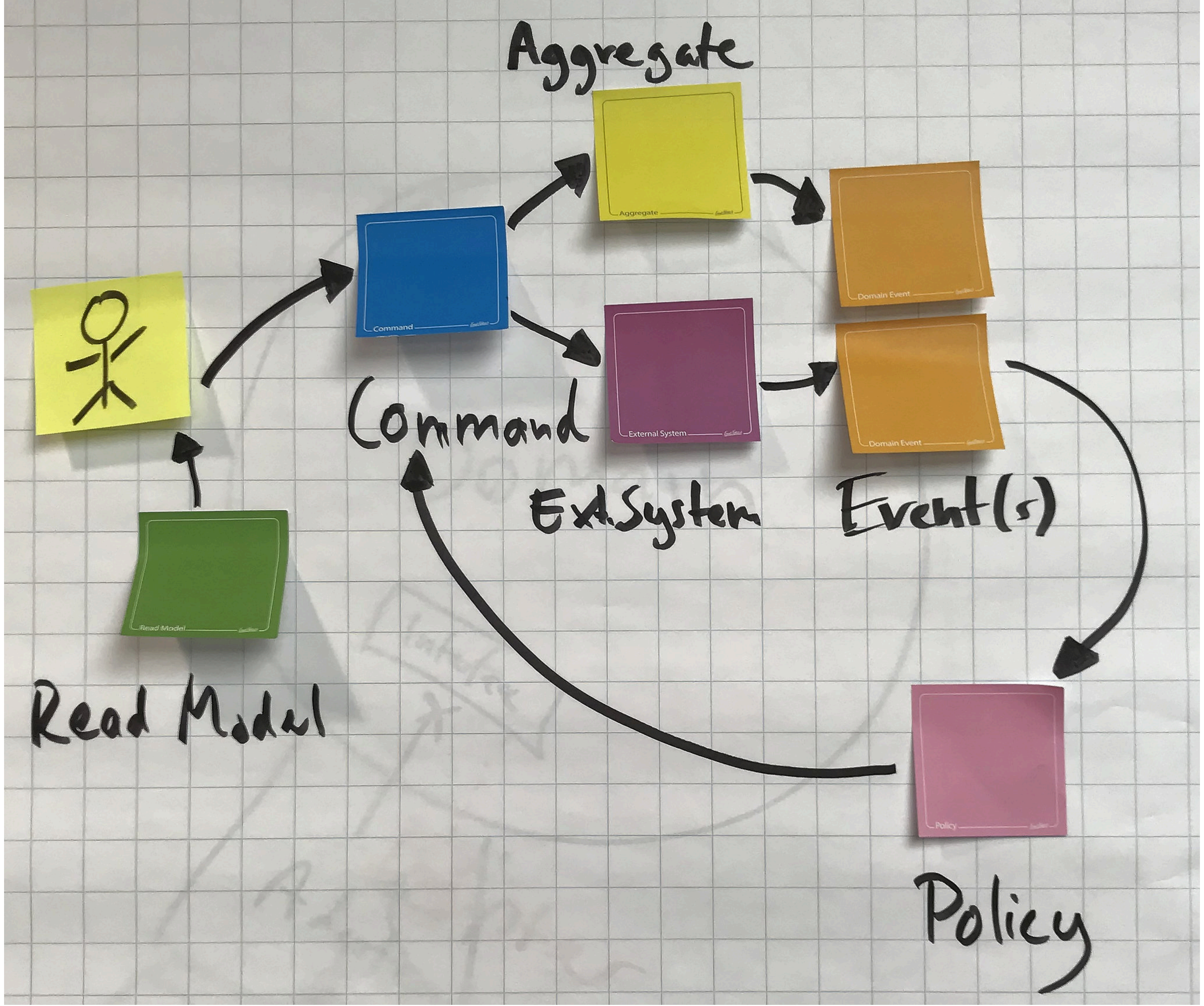
Richtig guten Code schreiben

Sebastian Bergmann

https://thephp.cc/termine/2019/10/contao-konferenz/richtig-guten-code-schreiben

Domain-Driven Design

Event Storming


Aggregate
Aggregate
Command
Command
External System
Ext.System
Domain Event
Domain Event
Event(s)
Read Model
Read Model
Policy
Policy

Contact

https://thephp.cc

https://talks.thephp.cc

sebastian@thephp.cc

@s_bergmann

Image Credits

- "Day of the Tentacle", Copyright 1993 LucasArts sharing experience

thePHP.cc