

Testning och refaktorisering

DA288A – Molnbaserade Webbapplikationer

Dagens agenda

- Testning
 - Varför?
 - När? Vad? Hur?
 - Vilka verktyg?
- Refaktorisering
 - Varför?
 - När? Vad? Hur?

Vad är testning?

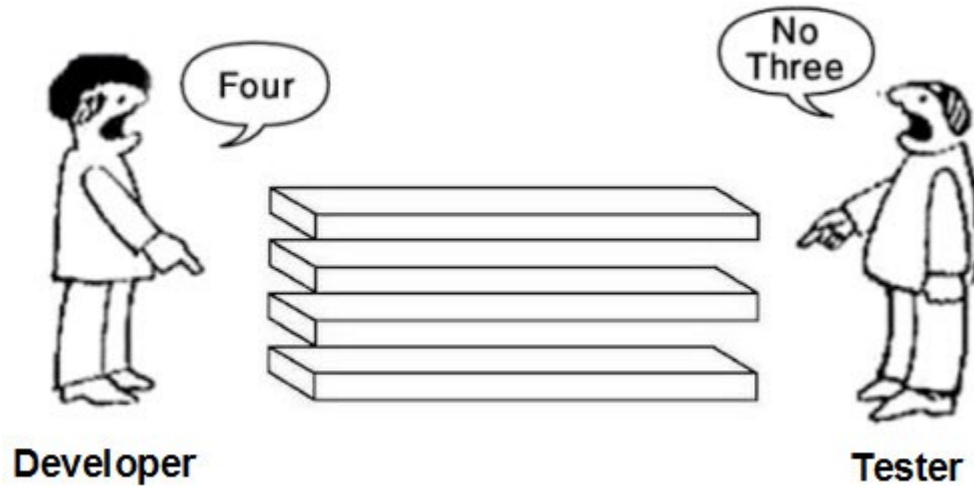
“**Software testing** is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.”

Enl. Wikipedia

- Validering → Bygger vi rätt saker?
- Verifikation → Bygger vi sakerna på rätt sätt?

Varför testar vi våra applikationer?

Old but True Controversy



www.softwaretestinggenius.com

Varför testar vi våra applikationer?

- Tekniska skäl
- Utvecklingsteamets skäl
- Ekonomiska skäl

Tekniska skäl: Säkerställ funktionaliteten

Det här är validering och verifiering!

- Fungerar koden som den är tänkt att göra?
 - Hanteras indatan på rätt sätt?
 - Fungerar koden med felaktig indata?
 - Är koden feltolerant?
- Kommer programmet att fungera i produktion?
 - Matchar vår utvecklingsmiljö produktionsmiljön?
 - Fungerar all kod tillsammans?

My code working well on on my machine

** Deploys **



Utvecklingsteamets skäl: Förtroende

- En bra utvecklare kan visa att hens kod fungerar
- Bra tester säkerställer att koden fungerar
 - Tester är bra att ha under utvecklingen av en funktion
 - De är ännu bättre att ha när koden har levt en tid
- Tester kan användas som bas i diskussioner

Utvecklingsteamets skäl: Historik och nya utvecklare

Tester är dokumentation → förenklar
introduktion av nya utvecklare

- Väl utformade och beskrivna tester visar hur en klass eller metod ska fungera
- BDD (se senare slides) beskriver hur applikationen ska fungera
- Lösta buggar och fel visas med tester

Ekonomiska skäl: Driftstörningar är dyrare än utvecklingstid

- Mjukvarutestning är dyrt
 - Längre utvecklingstid
 - Fler utvecklare/testare
 - Mer infrastruktur
- Fel i mjukvaran är dyrare
 - Nertid (se nästa slide)
 - Dålig PR/goodwill

Ekonomiska skäl: Driftstörningar är dyrare än utvecklingstid

Average Cost of Downtime

Even if your company survives a disaster, the costs are staggering:

- Brokerage \$6M - \$7M / hour
- Banking \$5 - \$6M / hour
- Credit Card \$2M - \$3M / hour
- Pay Per View \$1 - \$2M / hour (up to \$50M for fights)
- Airline Reservations \$1M / hour
- Home Shopping \$100K / hour
- Catalog Sales \$100K / hour
- Tele-ticket \$70K / hour
- Package Shipping \$30K / hour
- ATM Fees \$20K / hour

Average mean time to repair or recover: 4.0 hours



Morpheus, *How to manage app uptime like a boss* -

<https://www.morpheusdata.com/blog/2016-04-06-how-to-manage-app-uptime-like-a-boss>

Räkneexempel: ROI för testning

I exemplet nedan kostar testningen 100 000 kr

Utvecklingsfas	Pris för att fixa fel i denna fas
Kravfasen	100,00 kr
Designfasen	200,00 kr
Implementationsfasen	1 000,00 kr
Efter leverans	10 000,00 kr

Räkneexempel: ROI för testning

Fas	Fel som hittas utan test	Fel som hittas med test
Kravfasen	160	250
Designfasen	230	300
Implementationsfasen	270	430
Efter leverans	340	20

Räkneexempel: ROI för testning

Pris för felkorrigering	Utan testning	Med testning
Kravfasen	$160 * 100 \text{ kr} = 16\,000 \text{ kr}$	$250 * 100 \text{ kr} = 25\,000 \text{ kr}$
Designfasen	$230 * 200 \text{ kr} = 46\,000 \text{ kr}$	$300 * 200 \text{ kr} = 60\,000 \text{ kr}$
Implementationsfasen	$270 * 1\,000 \text{ kr} = 270\,000 \text{ kr}$	$430 * 1\,000 \text{ kr} = 430\,000 \text{ kr}$
Efter leverans	$340 * 10\,000 \text{ kr} = 3\,400\,000 \text{ kr}$	$20 * 10\,000 \text{ kr} = 200\,000 \text{ kr}$
Summa	3 732 000 kr	715 000 kr

Räkneexempel: ROI för testning

$$ROI = \frac{\text{nytta} - \text{kostnad}}{\text{kostnad}}$$

$$\text{nytta} = \text{Pris utan testning} - \text{Pris med testning} = 2\,917\,000$$

$$\text{kostnad} = 100\,000$$

$$ROI = \frac{2\,917\,000 - 100\,000}{100\,000} = 30,17$$

När testar vi?

- Före utvecklingsfasen
- Under utvecklingsfasen
- Efter utvecklingsfasen

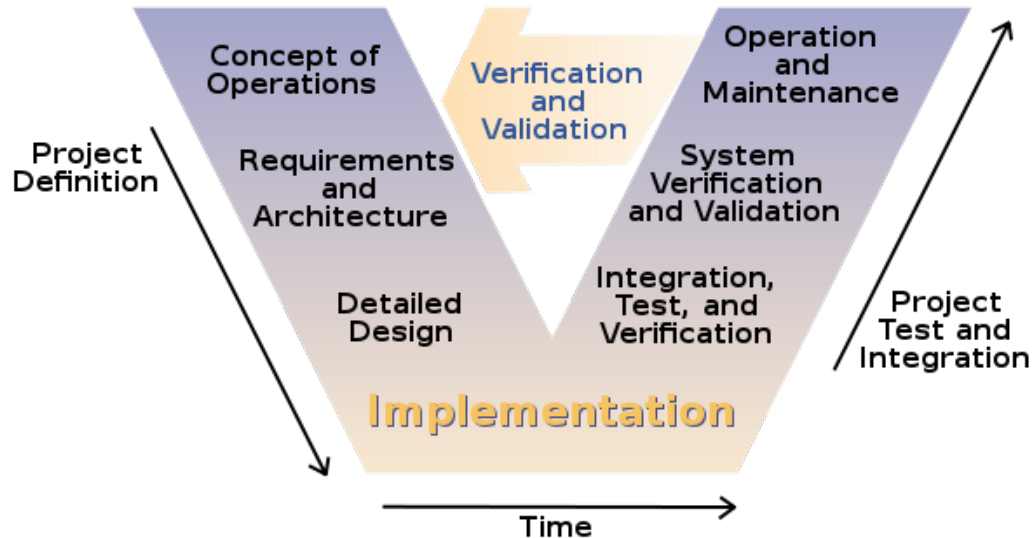
Före utvecklingsfasen

- Bestäm er för en gemensam teststrategi
 - Vad ska testas?
 - Hur ofta körs testerna?
 - Vem ansvarar för testningen?
- Lägg tid på att sätta upp en ordentlig testmiljö
 - Besluta om en gemensam test- och utvecklingsmiljö inom teamet
 - Exempel: PHPUnit + Xdebug + CI

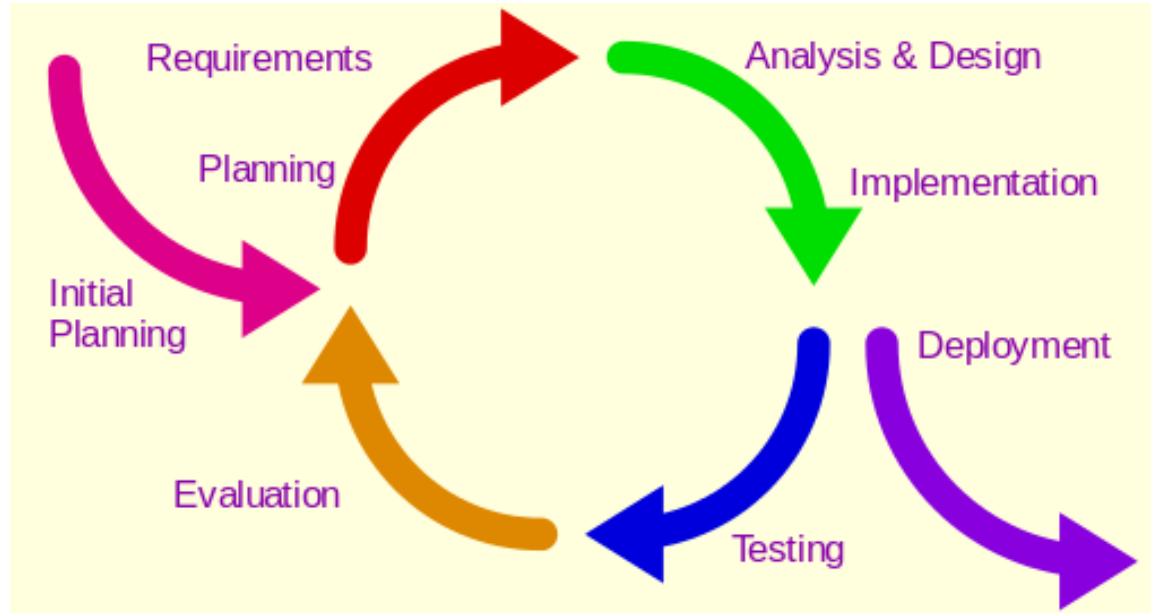
Under utvecklingen

- Kontinuerligt!
 - Ny funktionalitet? Skriv tester direkt
- Testning i två utvecklingsmodeller:
 - V-modellen
 - Agil utveckling

Under utvecklingen



Under utvecklingen



Wikipedia, Iterative and incremental development -
https://en.wikipedia.org/wiki/Iterative_and_incremental_development

Efter utvecklingen

Integrationstestning: Testa applikationen i sin helhet innan den driftsätts i produktion.

Regressionstestning: Säkerställ att “gammal” funktionalitet inte förstörs i nya releaser.

Felsökning: Dokumentera fixade buggar genom att skriva tester som visar att felet är åtgärdat.

Vad testar vi?

- Enskilda klasser och metoder
 - Enhetstester
- Applikationen i sin helhet
 - Integrationstester
 - Systemtester

Hur testar vi?

Vi kommer att titta på:

- BDD-tester med **Behat**
- Enhetstester med **PHPUnit**



Integrationstester med Story BDD

- *Story BDD* står för *Story Behaviour-Driven Design*
- Testar applikationen i sin helhet
- Flera tester som visar hur applikationen ska fungera
 - Testerna skrivs i “berättelseform”:
“As a paying customer, I need to log in”
 - Skriv testerna innan koden – med BDD får du hjälp med att skapa koden du behöver!

Exempel: Story BDD-tester med Behat

- 1) Beskriv applikationens beteenden
- 2) Generera test- och kodskelett
- 3) Implementera testerna och koden
- 4) Kör testerna!

Behat, Quick *start* - http://behat.org/en/latest/quick_start.html



Feature: Product basket

In order to buy products

As a customer

I need to be able to put interesting products into a basket

Rules:

- VAT is 20%
- Delivery for basket under £10 is £3
- Delivery for basket over £10 is £2

Scenario: Buying a single product under £10

Given there is a "Sith Lord Lightsaber", which costs £5

When I add the "Sith Lord Lightsaber" to the basket

Then I should have 1 product in the basket

And the overall basket price should be £9

Scenario: Buying a single product over £10

Given there is a "Sith Lord Lightsaber", which costs £15

When I add the "Sith Lord Lightsaber" to the basket

Then I should have 1 product in the basket

And the overall basket price should be £20

Scenario: Buying two products over £10

Given there is a "Sith Lord Lightsaber", which costs £10

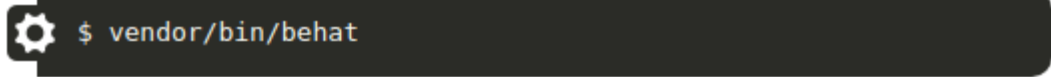
And there is a "Jedi Lightsaber", which costs £5

When I add the "Sith Lord Lightsaber" to the basket

And I add the "Jedi Lightsaber" to the basket

Then I should have 2 products in the basket

And the overall basket price should be £20



```
$ vendor/bin/behav
```



```
// features/bootstrap/FeatureContext.php
```

```
use Behat\Behat\Tester\Exception\PendingException;
use Behat\Behat\Context\SnippetAcceptingContext;
use Behat\Gherkin\Node\PyStringNode;
use Behat\Gherkin\Node\TableNode;
```

```
class FeatureContext implements SnippetAcceptingContext
{
```

```
    /**
```

```
     * @Given there is a :arg1, which costs f:arg2
```

```
     */
```

```
    public function thereIsAWhichCostsPs($arg1, $arg2)
```

```
    {
```

```
        throw new PendingException();
```

```
    }
```

```
    /**
```

```
     * @When I add the :arg1 to the basket
```

```
     */
```

```
    public function iAddTheToTheBasket($arg1)
```

```
    {
```

```
        throw new PendingException();
```

```
    }
```

```
    /**
```

```
     * @Then I should have :arg1 product(s) in the basket
```

```
     */
```

```
    public function iShouldHaveProductInTheBasket($arg1)
```

```
    {
```

```
        throw new PendingException();
```



```
// features/bootstrap/FeatureContext.php
```

```
use Behat\Behat\Tester\Exception\PendingException;
use Behat\Behat\Context\SnippetAcceptingContext;
use Behat\Gherkin\Node\PyStringNode;
use Behat\Gherkin\Node\TableNode;

class FeatureContext implements SnippetAcceptingContext
{
    private $shelf;
    private $basket;

    public function __construct()
    {
        $this->shelf = new Shelf();
        $this->basket = new Basket($this->shelf);
    }

    /**
     * @Given there is a :product, which costs £:price
     */
    public function thereIsAWhichCostsPs($product, $price)
    {
        $this->shelf->setProductPrice($product, floatval($price));
    }

    /**
     * @When I add the :product to the basket
     */
    public function iAddTheToTheBasket($product)
    {
        $this->basket->addProduct($product);
    }
}
```



```
// features/bootstrap/Basket.php
```

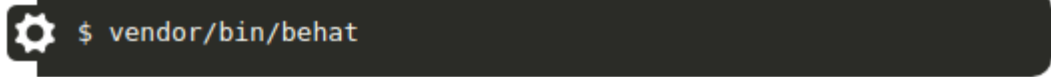
```
final class Basket implements \Countable
{
    private $shelf;
    private $products;
    private $productsPrice = 0.0;

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

    public function addProduct($product)
    {
        $this->products[] = $product;
        $this->productsPrice += $this->shelf->getProductPrice($product);
    }

    public function getTotalPrice()
    {
        return $this->productsPrice
            + ($this->productsPrice * 0.2)
            + ($this->productsPrice > 10 ? 2.0 : 3.0);
    }

    public function count()
    {
        return count($this->products);
    }
}
```



```
$ vendor/bin/behav
```

Enhetstester med PHPUnit

- Testar de enskilda beståndsdelarna av en applikation
- Flera tester för varje enskild metod och klass
 - Testa många typer av indata
 - Välj ut relevanta typfall, exempelvis gränfallen
 - Skriv testerna i samband med att koden skrivs

Exempel: Enhetstester med PHPUnit

PHPUnit, *Getting started* - <https://phpunit.de/getting-started.html>

Code

src/Email.php

```
<?php
declare(strict_types=1);

final class Email
{
    private $email;

    private function __construct(string $email)
    {
        $this->ensureIsValidEmail($email);

        $this->email = $email;
    }

    public static function fromString(string $email): self
    {
        return new self($email);
    }

    public function __toString(): string
    {
        return $this->email;
    }

    private function ensureIsValidEmail(string $email)
    {
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            throw new InvalidArgumentException(
                sprintf(
                    '"%s" is not a valid email address',
                    $email
                )
            );
        }
    }
}
```

Test Code

tests/EmailTest.php

```
<?php
declare(strict_types=1);

use PHPUnit\Framework\TestCase;

/**
 * @covers Email
 */
final class EmailTest extends TestCase
{
    public function testCanBeCreatedFromValidEmailAddress()
    {
        $this->assertInstanceOf(
            Email::class,
            Email::fromString('user@example.com')
        );
    }

    public function testCannotBeCreatedFromInvalidEmailAddress()
    {
        $this->expectException(InvalidArgumentException::class);

        Email::fromString('invalid');
    }

    public function testCanBeUsedAsString()
    {
        $this->assertEquals(
            'user@example.com',
            Email::fromString('user@example.com')
        );
    }
}
```

Test Execution

```
→ phpunit --bootstrap src/Email.php tests/EmailTest
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.
```

```
...
```

```
3 / 3 (100%)
```

```
Time: 70 ms, Memory: 10.00MB
```

```
OK (3 tests, 3 assertions)
```

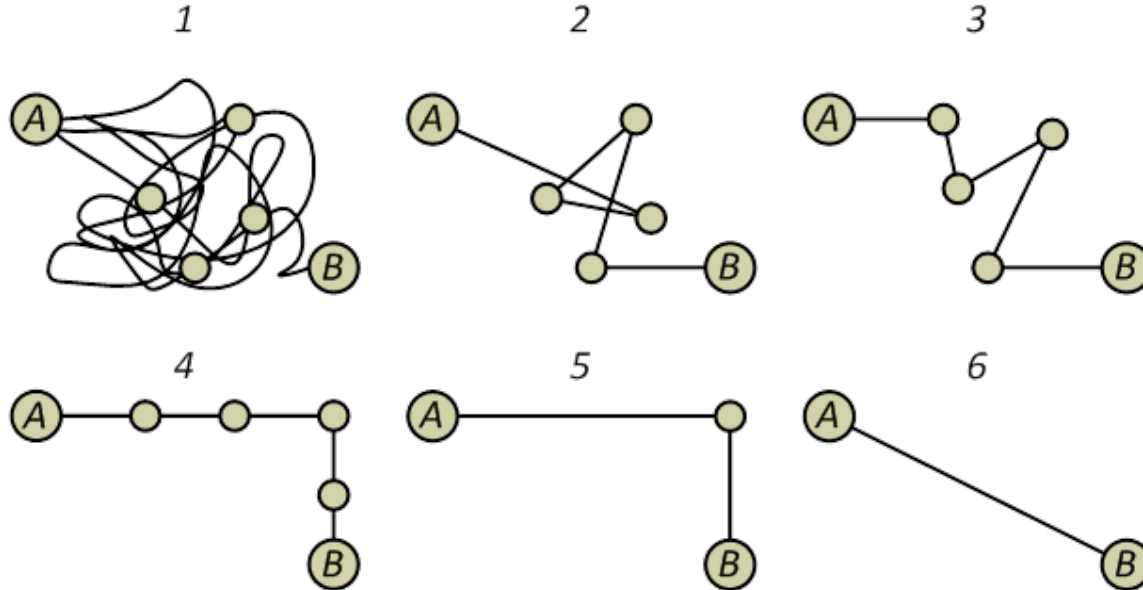
Below you see an alternative output of the test result that is enabled using the `--testdox` option:

```
→ phpunit --bootstrap src/Email.php --testdox tests
PHPUnit 6.1.0 by Sebastian Bergmann and contributors.
```

```
Email
```

```
[x] Can be created from valid email address
[x] Cannot be created from invalid email address
[x] Can be used as string
```

Refaktorisering



Vad är refaktorisering?

“**Code refactoring** is the process of restructuring existing computer code—changing the factoring—without changing its external behavior.”

Enl. Wikipedia

Typer av refaktorisering

Abstraktion: Dölj implementation för att enklare kunna byta ut den vid behov.

Nedbrytning: Bryt ner långa funktioner och klasser i mindre, mer förståeliga enheter.

Namngivning: Vettigare namngivning av klasser, variabler och metoder

Omstrukturering: Mer logisk placering av variabler, metoder och kommentarer.

Varför refaktorerar vi vår kod?

- Tydlighet
- Förtroende
- Prestanda

Varför vi refaktorerar vår kod: tydlighet

- Bra namngivning av variabler och metoder bidrar till ökad förståelse för vad koden gör
- Användande av code patterns bidrar till ökad förståelse för vad koden gör
 - Enklare och billigare att underhålla
 - Enklare att bygga ut



Varför vi refaktorerar vår kod: förtroende

- Om applikationen innehåller kod som inte går att testa
 - Bygg om koden som inte är testbar
 - Testa de utbrutna delarna

Varför vi refaktorerar vår kod: prestanda

- Genom att bygga bort flaskhalsar kan vi få ut mer av hårdvaran:
 - Fler förfrågningar per sekund
 - Kortare svarstider
 - Gladare kunder
 - (och mer pengar att köpa kakor för)

När refaktorerar vi vår kod?

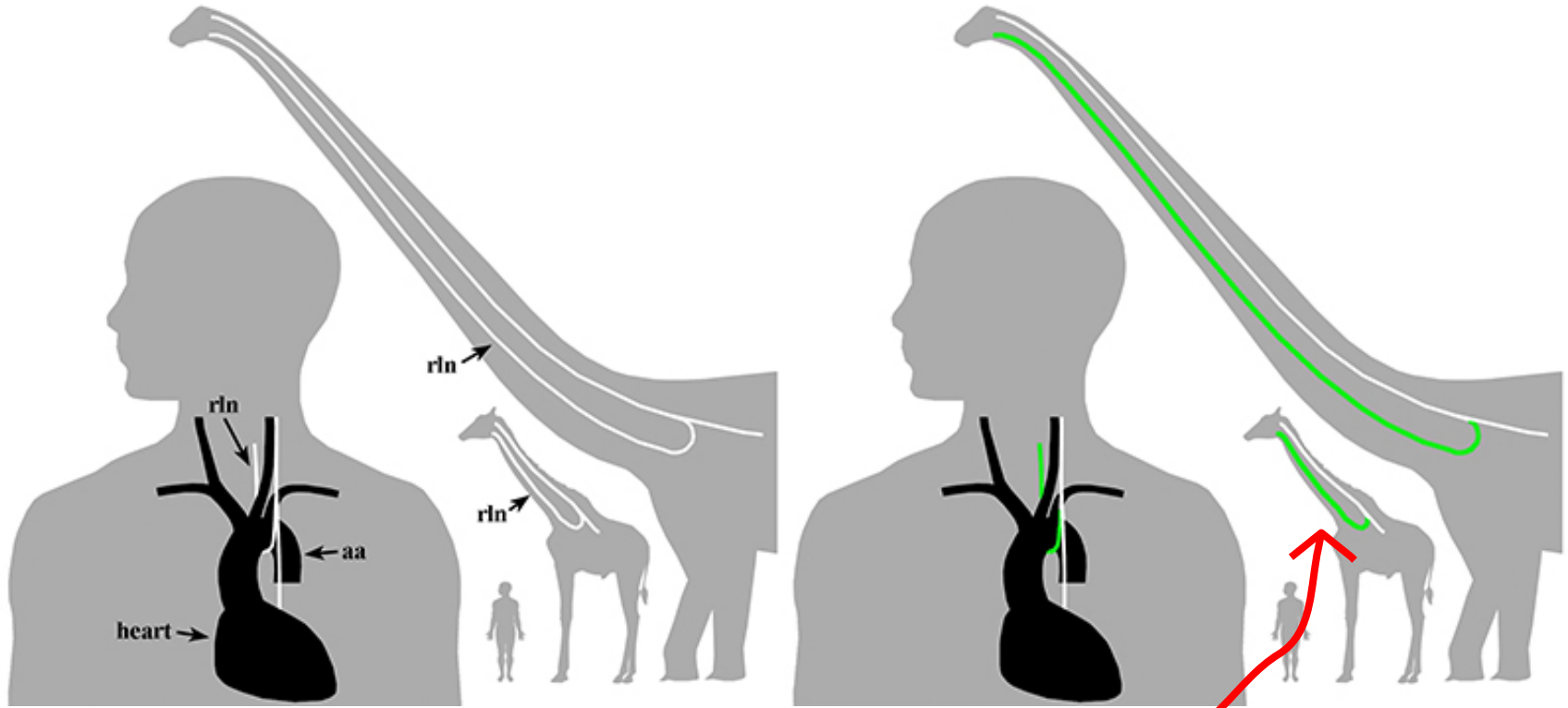
Code smell

- På applikationsnivå
 - Duplicerad kod
 - För hög komplexitet
- På klassnivå
 - För stor klass
 - Klassen gör för mycket
 - Klassen gör för lite
 - Klassen beror för hårt på andra klasser
 - Ihopklumpning av data

När refaktorerisar vi vår kod?

Code smell

- På metodnivå:
 - För många parameterar
 - För lång metod
 - För långt metodnamn
 - För kort metodnamn
 - För mycket returdata



code smell!

Hur refaktorerar vi vår kod?

Fungerande,
vältestad kod



Välj ut en code smell
som ska åtgärdas



Hur refaktorerar vi?

Större aktiviteter med GitFlow

- 1) Se till att koden är vältestad
- 2) Skapa en ny feature-branch
- 3) Refaktorisera koden
- 4) Kör igenom testerna

Matchar inte testerna koden? Skriv om dem?

- 5) Mergea in feature-branchen i develop igen

Klassiska refaktoriseringsmetoder

Catalog of Refactorings



Martin Fowler

10 December 2013

This catalog of refactorings includes those refactorings described in my original book on Refactoring, together with the Ruby Edition.

Using the Catalog ►

Tags

- ☐ associations
- ☐ encapsulation
- ☐ generic types
- ☐ interfaces
- ☐ class extraction
- ☐ GOF Patterns
- ☐ local variables
- ☐ vendor libraries
- ☐ errors
- ☐ type codes

► Add Parameter

A method needs more information from its caller.

Add a parameter for an object that can pass on this information.

[more...](#)

► Change Bidirectional Association to Unidirectional

► Change Reference to Value

► Change Unidirectional Association to Bidirectional

► Pull Up Constructor Body

► Pull Up Field

► Pull Up Method

You have methods with identical results on subclasses.

Move them to the superclass.

[more...](#)

► Push Down Field

► Push Down Method

Fowler, M., *Catalog of Refactorings*. <https://refactoring.com/catalog>

Exempel

Duplicerad
kod!

```
<?php
class Student {
    var $name;

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

    function getName()
    {
        return $this->name;
    }

    function setName($name)
    {
        $this->name = $name;
    }
}
```

```
class Teacher {
    var $name;

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

    function getName()
    {
        return $this->name;
    }

    function setName($name)
    {
        $this->name = $name;
    }
}
?>
```

Generalisering

```
<?php
class Person {
    var $name;

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

    function getName()
    {
        return $this->name;
    }

    function setName($name)
    {
        $this->name = $name;
    }
}

class Student extends Person {
    function __construct($name)
    {
        parent::__construct($name);
    }
}

class Teacher extends Person {
    function __construct($name)
    {
        parent::__construct($name);
    }
}
?>
```