



MALMÖ HÖGSKOLA

Testning och refaktorisering

Dagens agenda

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

Testning

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

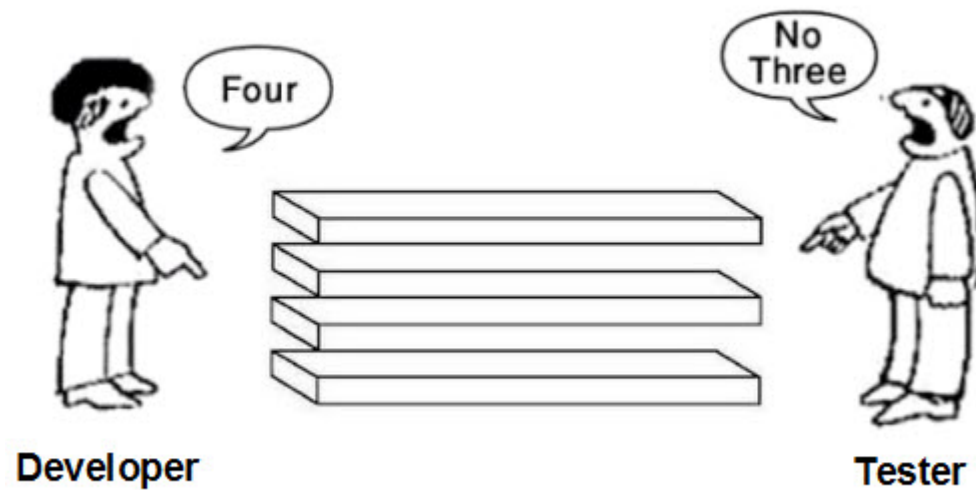
- Validering → Bygger vi rätt saker?
- Verifikation → Bygger vi sakerna på rätt sätt?

Lockart, *Modern PHP*. Ch 10

Wikipedia, *Software testing* - https://en.wikipedia.org/wiki/Software_testing

Varför testar vi våra applikationer?

Old but True Controversy



www.softwaretestinggenius.com

Varför testar vi våra applikationer?

- Tekniska skäl
- Utvecklingsteamets skäl
- Ekonomiska skäl

Lockart, *Modern PHP*. Ch 10

Tekniska skäl: Säkerställ funktionaliteten

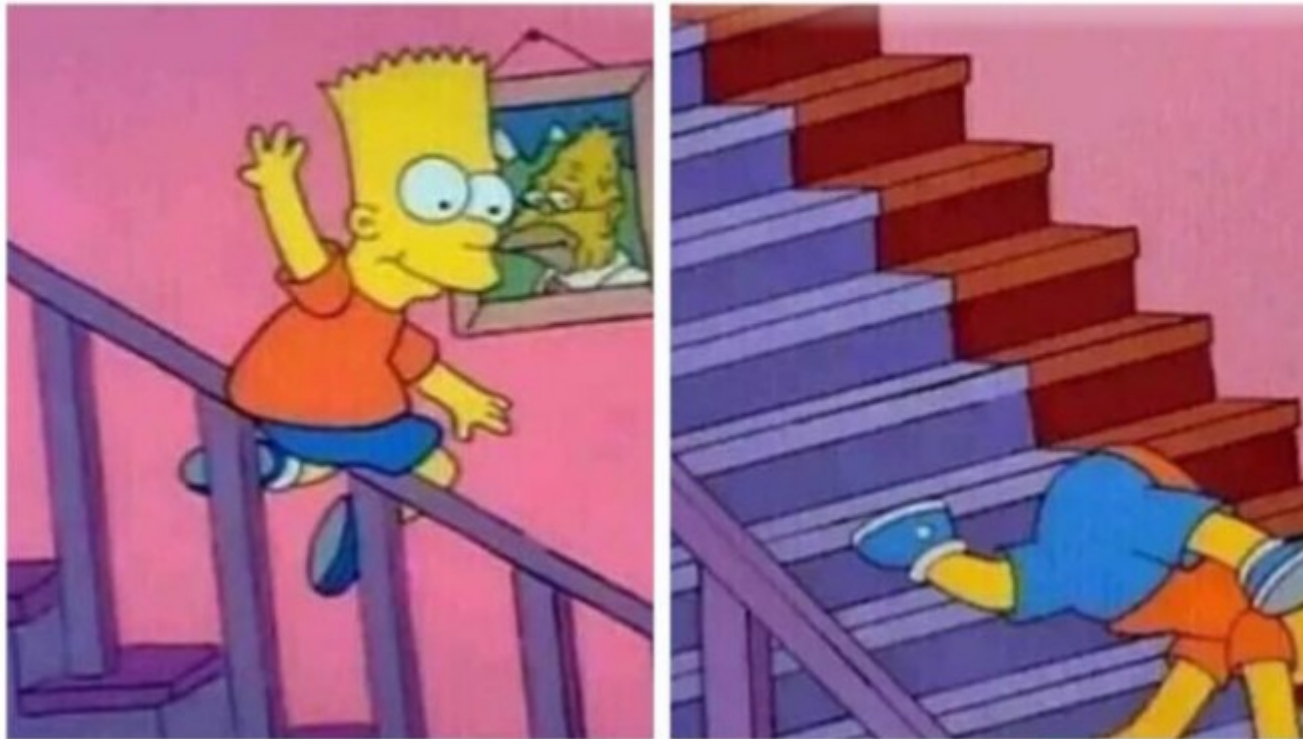
Det här är validering och verifiering!

- 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?

Tekniska skäl: Säkerställ funktionaliteten

My code working well on on my machine

** Deploys **



Utvecklingsteamets skäl: Förtroende

- En utvecklare kan visa att hens kod fungerar
- Bra tester säkerställer att koden fungerar
 - Bra att ha under utvecklingen av en funktion
 - 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 mjukvara 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>

När testar vi?

- Före utvecklingen
- Under utvecklingen
- Efter utvecklingen

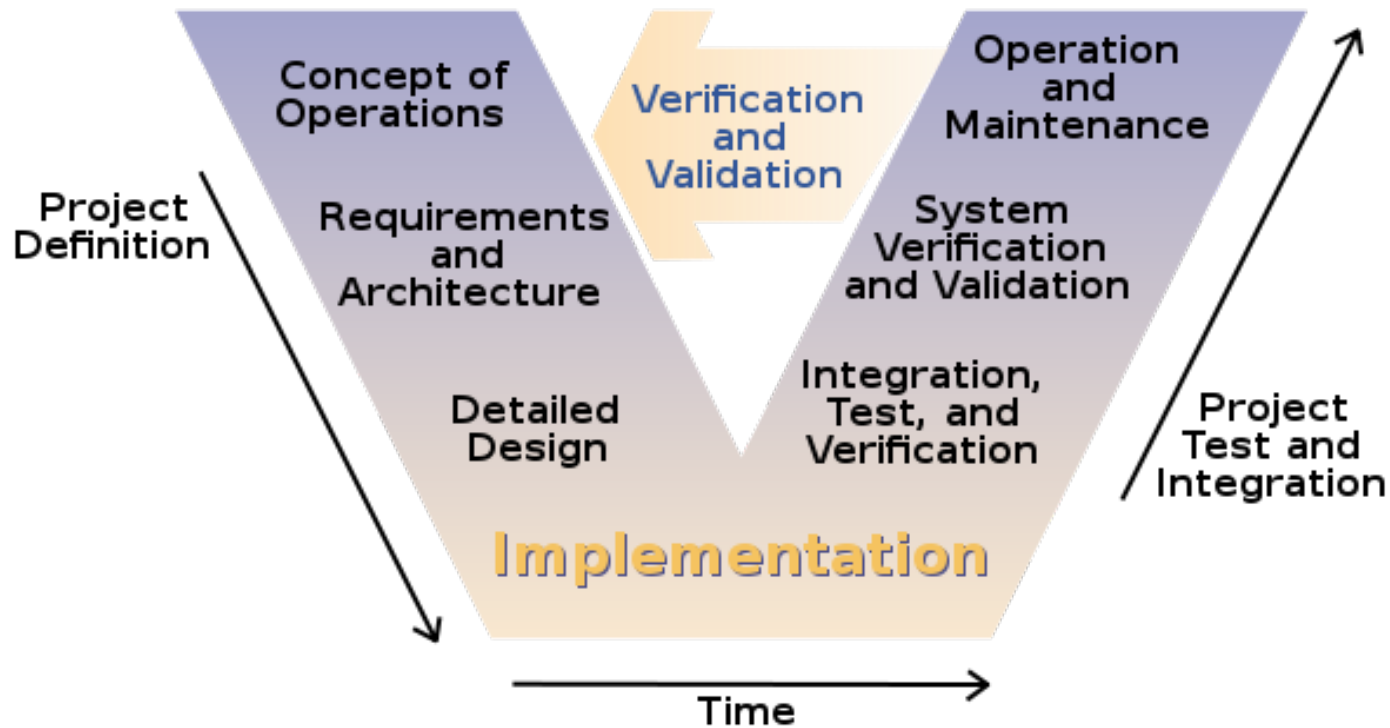
Före utvecklingen

- 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

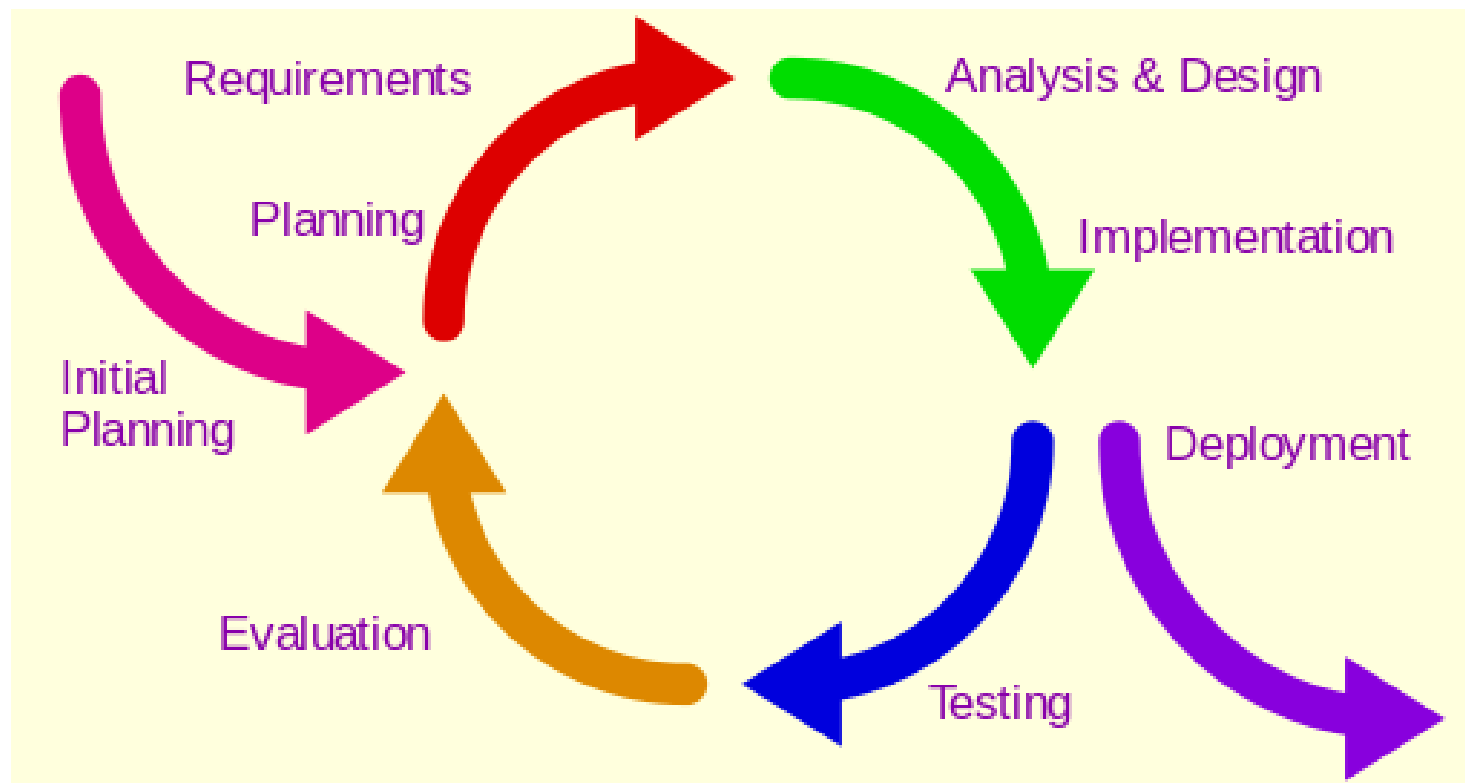
- Kontinuerligt!
 - Ny funktionalitet? Skriv tester direkt
- Testning i två utvecklingsmodeller:
 - V-modellen
 - Agil utveckling

Under utvecklingen



Wikipedia, V-Model - [https://en.wikipedia.org/wiki/V-Model_\(software_development\)](https://en.wikipedia.org/wiki/V-Model_(software_development))

Under utvecklingen



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

Efter utvecklingen

Integrationstestning: Testa applikationen 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.

Lockart, *Modern PHP*. Ch 10

Vad testar vi?

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

Hur testar vi?



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, ex. gränsfallen
 - Skriv testerna i samband med att koden skrivs

Integrationstester med StoryBDD

- 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

Testning: exempel

Enhetstestning 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)
```

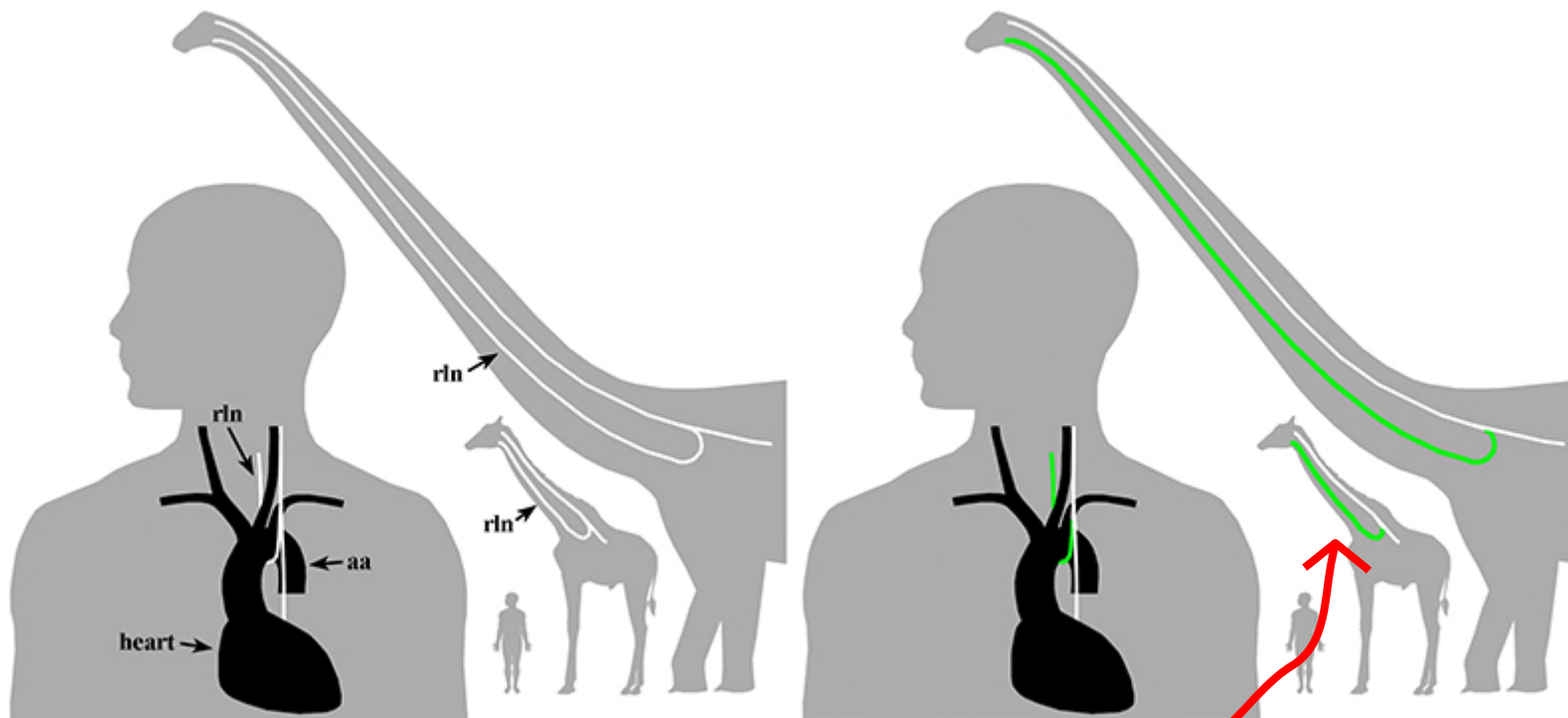
Below you see an alternative output of the test result that is enabled using the `--testdox` option:

```
→ 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



code smell!

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 utbyta 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 ökar förståelse av koden
- Användande av code patterns ökar förståelse av koden
 - 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 funktionaliteten 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

När refaktoriserar vi vår kod?



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 refaktorerar vi vår kod?

Code smell

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

Hur refaktorerar vi vår kod?

Fungerande,
vältestad kod



Välj ut en code smell
som ska åtgärdas



Hur refaktorerar vi vår kod?

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**

Martin Fowler, *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);
    }
}
?>
```