

PHPUnit in PHP

Key concepts

unit test, SUT, arrange, act, assert, code coverage, dependency injection, inversion of control, constructor injection, setter injection, mock

Alternatieve bronnen

<https://phpunit.de/getting-started.html>

<https://phpunit.de/manual/current/en/test-doubles.html>

<https://www.toptal.com/qa/how-to-write-testable-code-and-why-it-matters>

<https://php-and-symfony.matthiasnoback.nl/2014/07/test-doubles/>

<https://codeutopia.net/blog/2009/06/26/unit-testing-4-mock-objects-and-testing-code-which-uses-the-database/>

1. PHPUnit inleiding

In een unit test wordt een klein deel van een programma onafhankelijk van alle andere delen getest. Voor een object-georiënteerde programmeertaal wordt meestal één methode van één klasse per keer getest. De geteste klasse wordt ook de System Under Test (SUT) genoemd.

Unit tests zijn belangrijk omdat ze een eenvoudig antwoord aanbieden op de vraag of een wijziging in een klasse de werking van de rest van de code van het programma niet verstoort. Via de unit tests weet een programmeur dat de wijziging die hij aanbrengt in een klasse de werking van het programma niet in het gedrang brengt.

Er zijn drie delen in een unit test: arrange, act en assert. Tijdens de arrange-fase wordt de SUT geïnitieerd. Tijdens act wordt er met de SUT geïnterageerd door een methode op de SUT uit te voeren. In assert wordt gecontroleerd of het gedrag van de SUT overeenkomt met het verwachte gedrag.

In deze tekst wordt gewerkt met PHPUnit. De installatie van PHPUnit wordt beschreven in onderstaande link:

<https://phpunit.de/manual/current/en/installation.html>

We geven de voorkeur er aan om PHPUnit lokaal in het project onder de map vendor te installeren:

```
composer require --dev phpunit/phpunit ^6
```

Phpunit wordt dan aangeroepen (eventueel in vagrant) als

```
vendor/bin/phpunit SomeTest.php
```

2. Unit-test van een klasse zonder dependencies

In onderstaand voorbeeld is de SUT de klasse Date.

src/Util/Date.php

```
<?php namespace Util;

class Date
{
    const NUMBER_OF_DAYS_PER_MONTH =
        [31,28,31,30,31,30,31,31,30,31,30,31];
    private $day, $month, $year;

    public static function of ($day, $month, $year)
    {
        if(self::validateDate($day, $month, $year)){
            return new self($day,$month,$year);
        } else {
            throw new \Exception("Invalid date");
        }
    }

    private function __construct($day, $month, $year)
    {
        $this->year=$year;
        $this->month=$month;
        $this->day=$day;
    }

    private static function validateYear($year)
    {
        return is_int($year);
    }

    private static function validateMonth($month)
    {
        return is_int($month) && $month>=1 && $month<=12;
    }
}
```

```

private static function validateDate($day, $month, $year)
{
    if ( is_int($day) && self::validateMonth($month) &&
        self::validateYear($year) ){
        if($month==2 && self::validateLeapYear($year)){
            return $day >=1 && $day <= 29;
        } else {
            return $day >=1 &&
                $day <= self::NUMBER_OF_DAYS_PER_MONTH[$month-1];
        }
    } else{
        return false;
    }
}

private static function validateLeapYear($year)
{
    return $year % 4 == 0 &&
        ($year % 100 != 0 || $year % 400 == 0) ;
}

public function isLeapYear()
{
    return self::validateLeapYear($this->year);
}

function __toString()
{
    return sprintf ( "%d/%d/%d",
                    $this->day,
                    $this->month,$this->year );
}
}

```

Het gewone gebruik van de klasse wordt in app.php geïllustreerd.

app.php

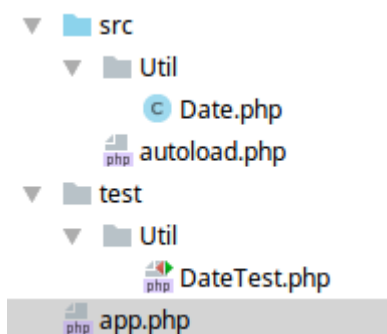
```
<?php

require 'src/autoload.php';
use Util\Date;

try{
    $date=Date::of(1,1,2000);
    print($date);
} catch (Exception $exception){
    print($exception);
}
```

De unit-test DateTest wordt in de map test/Util geplaatst. De naam van de unit-test moet altijd eindigen op Test. Een courante conventies is om dezelfde mappenstructuur te gebruiken, als de mappenstructuur van de SUT. De naam van unit-test is gelijk aan de naam van de SUT gevolgd door Test.

src/Util/Date.php wordt getest in test/Util/DateTest.php



Een eerste versie van de unit-test wordt hieronder getoond. Er worden twee tests

gespecificeerd. Let hierbij op de conventie van de naamgeving van de test:

test_naamVanTeTestenMethode_toestand_verwachtResultaat

In beide tests wordt één datum aangemaakt (arrange). De methode isLeapYear wordt uitgevoerd (act) en via assertTrue of assertFalse wordt gecontroleerd of het resultaat klopt (assert).

test/Util/DateTest.php (versie 1)

```
<?php
use Util\Date;
use PHPUnit\Framework\TestCase;
class DateTest extends TestCase
{
    public function testIsLeapYear_leapYear_true()
    {
        $date=Date::of(1,1,2000);
        $this->assertTrue($date->isLeapYear());
    }

    public function testIsLeapYear_notALeapYear_false()
    {
        $date=Date::of(1,1,2001);
        $this->assertFalse($date->isLeapYear());
    }
}
```

De test wordt uitgevoerd via de command-line:

vendor/bin/phpunit --bootstrap src/autoload.php test/Util/DateTest.php

De uitvoer wordt hieronder getoond. (--bootstrap zorgt ervoor dat autoload.php uitgevoerd wordt).

```
vagrant@local:/var/www/html$ vendor/bin/phpunit --bootstrap vendor/autoload.php test/DateTest.php
PHPUnit 6.3.1 by Sebastian Bergmann and contributors.

..                                                    2 / 2 (100%)
Time: 57 ms, Memory: 4.00MB
OK (2 tests, 2 assertions)
```

Er zijn verschillende assert-methodes:

<code>assertEquals(expected, actual)</code>	kijk of de bekomen waarde (actual) gelijk is aan een verwachte waarde (expected)
<code>assertTrue(actual)</code>	kijk of de bekomen waarde True is
<code>assertFalse(actual)</code>	kijk of de bekomen waarde False is
<code>assertLessThan(expected, actual)</code>	kijk of de bekomen waarde kleiner is dan de verwachte waarde
<code>assertNull(expected)</code>	kijk of de bekomen waarde gelijk is aan null
<code>assertRegExp(pattern, actual)</code>	kijk of de bekomen waarde voldoet aan een patroon
<code>assertInstanceOf(expected, object)</code>	kijk object een instantie is van een klasse

Een meer uitgebreid overzicht wordt op onderstaande link gegeven:

<https://phpunit.de/manual/6.0/en/appendixes.assertions.html>

Via een dataProvider kunnen meerdere gegevens gestuurd worden naar dezelfde test methode. In onderstaande methode wordt in de methode `providerLeapYears` een array met daarin telkens een waarde voor `$day`, `$month` en `$year` (in een array) teruggegeven.

test/Util/DateTest.php (versie 2)

```
<?php
use Util\Date;
use PHPUnit\Framework\TestCase;
class DateTest extends TestCase
{
    /**
     * @dataProvider providerLeapYears
     */
    public function testIsLeapYear_leapYear_true($day,$month,$year)
    {
        $date=Date::of($day,$month,$year);
        $this->assertTrue($date->isLeapYear());
    }

    public function providerLeapYears()
    {
        return array(
            array(1,1,1904),
            array(1,1,1908),
            array(1,1,1912),
            array(1,1,2000),
            array(1,1,2400),
            array(1,1,2800),
        );
    }
}
```

```
vagrant@local:/var/www/html$ vendor/bin/phpunit --bootstrap vendor/autoload.php test/DateTest.php
PHPUnit 6.3.1 by Sebastian Bergmann and contributors.

.....                                     6 / 6 (100%)

Time: 59 ms, Memory: 4.00MB
OK (6 tests, 6 assertions)
```

Via de annotatie `@expectedException` wordt gecontroleerd of een Exception opgeworpen wordt. Er wordt hieronder gecontroleerd of de methode of ofwel een instantie van de klasse `Date` teruggeeft of dat er een Exception opgeworpen wordt.

test/Util/DateTest.php (versie 3)

```
<?php
use Util\Date;
use PHPUnit\Framework\TestCase;
class DateTest extends TestCase{
    /**
     * @dataProvider providerValidDates
     */
    public function testOf_validDayMonthYear_dateObject($day,$month,$year)
    {
        $date=Date::of($day,$month,$year);
        $this->assertInstanceOf(Date::class, $date);
    }

    public function providerValidDates()
    {
        return array(
            array(1,1,2000),
            array(31,1,2000),
            array(29,2,2000)
        );
    }

    /**
     * @dataProvider providerInvalidDates
     * @expectedException Exception
     */
    public function testOf_inValidDate_exception($day,$month,$year){
        $date=Date::of($day,$month,$year);
    }

    public function providerInvalidDates()
    {
        return array(
            array("",1,2000),
            array(1,"",2000),
            array(1,1,""),
            array(null,true,false),
            array(-1,1,2000),
            array(1,-1,2000),
            array(32,1,2000),
            array(30,2,2000),
            array(29,2,2001),
            array(1,-1,2000),
            array(1,13,2000),
        );
    }
}
```

Na installatie van xdebug <https://xdebug.org/docs/install>¹ kan ook een code-coverage report gemaakt worden². Het bestand phpunit.xml moet in het project geplaatst worden en het commando wordt

```
vendor/bin/phpunit --coverage-html="coverage_html" test/Util/DateTest.php
```

phpunit.xml























```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="https://schema.phpunit.de/6.0/phpunit.xsd"
    bootstrap="src/autoload.php"
    backupGlobals="false"
    verbose="true">

    <filter>
        <whitelist processUncoveredFilesFromWhitelist="true">
            <directory suffix=".php">src</directory>
        </whitelist>
    </filter>
</phpunit>
```

In de map coverage_html wordt een overzicht van de code coverage getoond:

/home/jan/PhpstormProjects/example/src / Util / Date.php

	Code Coverage								
	Classes and Traits			Functions and Methods				Lines	
Total		100.00%	1 / 1		100.00%	8 / 8	CRAP		100.00% 20 / 20
Date		100.00%	1 / 1		100.00%	8 / 8	20		100.00% 20 / 20
of					100.00%	1 / 1	2		100.00% 3 / 3
__construct					100.00%	1 / 1	1		100.00% 4 / 4
validateYear					100.00%	1 / 1	1		100.00% 1 / 1
validateMonth					100.00%	1 / 1	3		100.00% 1 / 1
validateDate					100.00%	1 / 1	8		100.00% 7 / 7
validateLeapYear					100.00%	1 / 1	3		100.00% 2 / 2
isLeapYear					100.00%	1 / 1	1		100.00% 1 / 1
toString					100.00%	1 / 1	1		100.00% 1 / 1

1sudo apt-get install php-xdebug voor Debian-based OS :-)

2De volledige code van de test kan je op BB vinden.

3. Unit-test van een klasse met dependencies, ontestbare code

Een klasse met dependencies maakt gebruik van objecten van een andere klasse. Als voorbeeld wordt een klasse User besproken die afhangt van de (voorgedefinieerde) klasse LocalDateTime.

src/Util/User.php

```
<?php namespace Util;

class User
{
    private $id;
    private $name;
    private $loginDate;

    public function __construct($id, $name)
    {
        $this->id = $id;
        $this->name = $name;
        $this->loginDate =
            \DateTime::createFromFormat('U.u', microtime(true));
    }

    public function __toString()
    {
        return sprintf("%d, %s, %s",
            $this->id,
            $this->name,
            $this->loginDate->format("d-m-Y H:i:s[u]"));
    }
}
```

De methode `__toString` is ontestbaar. De klasse `User` staat zelf in voor de creatie van `loginDate`. Dit gebeurt in de constructor van `User`. Tijdens de instantiatie van `User` wordt de tijd uitgelezen en bewaard in `loginDate`. Bij het uitvoeren van de test kan geprobeerd worden om de tijd opnieuw uit te lezen maar dit leidt altijd tot een tijdsverschil van enkele ms. De test faalt zoals getoond in onderstaande figuur.

test/Util/UserTest.php

```
<?php

use Util\User;
use PHPUnit\Framework\TestCase;

class DateTest extends PHPUnit_Framework_TestCase
{
    public function testToString_User_correctString()
    {
        $user=new User(1,"jan");
        $now=\DateTime::createFromFormat('U.u', microtime(true));
        $output=sprintf("%d, %s, %s",
            1, "jan",$now->format("d-m-Y H:i:s[u]"));
        $this->assertEquals($output, $user->__toString());
    }
}
```

```
vagrant@local:/var/www/html$ vendor/bin/phpunit --bootstrap vendor/autoload.php test/UserTest.php
PHPUnit 6.3.1 by Sebastian Bergmann and contributors.

F                                                                    1 / 1 (100%)
Time: 77 ms, Memory: 4.00MB
There was 1 failure:

1) DateTest::testToString_User_correctString
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'1, jan, 04-10-2017 12:44:23[047700]'
+'1, jan, 04-10-2017 12:44:23[047600]'

/var/www/html/test/UserTest.php:15

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

4. Unit-test van een klasse met dependencies, testbare code

Het vorige voorbeeld kan eenvoudig testbaar gemaakt worden. De truc bestaat erin om User de controle over de instantiatie van de loginDate te ontnemen (inversion of control). De klasse User krijgt nu een waarde voor loginDate via de constructor (constructor injection). In de test kan dus één DateTime object gemaakt worden, dit object wordt via de constructor geïnjecteerd in de klasse User.

src/Util/User.php

```
<?php namespace Util;

class User
{
    private $id;
    private $name;
    private $loginDate;

    public function __construct($id, $name, \DateTime $loginDate)
    {
        $this->id = $id;
        $this->name = $name;
        $this->loginDate = $loginDate;
    }

    public function __toString()
    {
        return sprintf("%d, %s, %s",
            $this->id,
            $this->name,
            $this->loginDate->format("d-m-Y H:i:s[u]"));
    }
}
```

test/Util/UserTest.php

```
<?php

use Util\User;
use PHPUnit\Framework\TestCase;

class UserTest extends TestCase
{
    public function testToString_User_correctString()
    {
        $name = 'testuser';
        $now = \DateTime::createFromFormat('U.u',
                                           microtime(true));
        $user = new User(1, $name, $now);
        $expectedOutput = sprintf("%d, %s, %s",
                                   1,
                                   $name,
                                   $now->format("d-m-Y H:i:s[u]")
        );
        $this->assertEquals($expectedOutput,
                           $user->__toString());
    }
}
```

5. Testen van de databank

De interactie met de databank kan getest worden adhv een aparte databank testpersondb.

In het onderstaand voorbeeld is PersoonModel het SUT. Aangezien er getest wordt met een werkende databank en de SUT dus niet onafhankelijk van de buitenwereld getest wordt, gaat het hier eigenlijk niet over een unit-test maar eerder over een functionele of integration test. Let in de code ook op de constructor injection van het PDO-object.

test/model/PDOPersonModelTest.php

```
<?php
use PHPUnit\Framework\TestCase;
use \model\PDOPersonModel;

class PDOPersonModelTest extends TestCase
{
    public function setUp()
    {
        $user = 'root';
        $password = 'root';
        $database = 'testpersondb';
        $server = 'localhost';
        $this->connection = new PDO(
            "mysql:host=$server;dbname=$database",
            $user,
            $password);
        $this->connection->setAttribute(
            PDO::ATTR_ERRMODE,
            PDO::ERRMODE_EXCEPTION
        );
        $this->connection->exec('DROP TABLE IF EXISTS persons');
        $this->connection->exec('CREATE TABLE persons (
            id INT,
            name VARCHAR(255),
            PRIMARY KEY (id)
        )');

        $persons=$this->providerPersons();
        foreach($persons as $person){
            $this->connection->exec(
                "INSERT INTO persons (id, name) VALUES (".
                $person['id'].", '".$person['name']."'");
        }
    }

    public function tearDown()
    {
        $this->connection = null;
    }
}
```

test/model/PDOPersonModelTest.php

```
public function providerPersons()
{
    return [
        ['id'=>'1', 'name'=>'testname1'],
        ['id'=>'2', 'name'=>'testname2'],
        ['id'=>'3', 'name'=>'testname3']
    ];
}

public function
testListPersons_personsInDatabase_ArrayPersons()
{
    $personModel = new PDOPersonModel($this->connection);
    $actualPersons = $personModel->listPersons();
    $expectedPersons=$this->providerPersons();
    $this->assertEquals('array', gettype($actualPersons));
    $this->assertEquals(count($expectedPersons)
                        , count($actualPersons));
    foreach($actualPersons as $actualPerson ){
        $this->assertContains($actualPerson, $expectedPersons);
    }
}

...
```


test/model/PDOPersonModelTest.php

```
...

    public function providerInvalidIds()
    {
        return [['0'], ['-1'], ['1.2'], ["aaa"], [12], [1.2]];
    }

    /**
     * @expectedException \InvalidArgumentException
     * @dataProvider providerInvalidIds
     */
    public function
testIdExists_invalidId_InvalidArgumentException($id)
    {
        $personModel = new PDOPersonModel($this->connection);
        $actual=$personModel->idExists($id);
    }

...
```

6. Mock objects

Soms is het nodig om een mock te maken van een dependency van de SUT. In onderstaand voorbeeld is de SUT de klasse `PersonController`. Deze klasse is afhankelijk van de klasse `JSONView` en `PersonModel` (let op de constructor-injection). We willen de klasse `PersonController` testen onafhankelijk van deze dependencies.

src/Controller/PersonController.php

```
<?php
namespace controller;

use model\PersonModel;
use view\View;

class PersonController
{
    private $personModel;
    private $jsonPersonView;
    private $jsonPersonsView;

    public function __construct(PersonModel $personModel,
                                View $jsonPersonView, View $jsonPersonsView)
    {
        $this->personModel = $personModel;
        $this->jsonPersonView = $jsonPersonView;
        $this->jsonPersonsView = $jsonPersonsView;
    }

    public function addPersonByIdAndName($id, $name)
    {
        $statusCode = 201;
        $person = null;
        try {
            if ($this->personModel->idExists($id)) {
                $statusCode = 200;
            }
            $person = $this->personModel
                ->addPersonByIdAndName($id, $name);
        } catch (\InvalidArgumentException $exception) {
            $statusCode = 400;
        } catch (\PDOException $exception) {
            $statusCode = 500;
        }
        $this->jsonPersonView->show(
            ['person' => $person, 'statusCode' => $statusCode]);
    }
}
```

In `PersonControllerTest` wordt een mock van de dependencies gemaakt. In onderstaande test wordt het gedrag van de mocks vastgelegd. `personModel` verwacht dat de methode `addPersonByIdAndName` aangeroepen wordt met `$id` en `$name` als argument. De terugkeerwaarde van `addPersonByIdAndName` wordt ook vastgelegd. `mockJsonView` verwacht dat de methode `show` aangeroepen wordt met `$data` als argument. De mocks worden vervolgens geïnjecteerd in de constructor van `PersonController` en de methode `addPersonByIdAndName` wordt aangeroepen op het `PersonController` object. De test bestaat er dus in om te kijken of in de methode `addPersonByIdAndName` van de klasse `PersonController` dat `addPersonByIdAndName` aangeroepen wordt op de `personModel` met het juiste argument en om te kijken of de methode `show` aangeroepen wordt op de `jsonView` met het juiste argument.

test/controller/PersonControllerTest.php

```
<?php
use PHPUnit\Framework\TestCase;
use \controller\PersonController;

class PersonControllerTest extends TestCase
{
    public function setUp()
    {
        $this->personModel =
            $this->getMockBuilder('\model\PersonModel')
                ->disableOriginalConstructor()
                ->getMock();
        $this->jsonPersonView =
            $this->getMockBuilder('\view\JsonPersonView')
                ->disableOriginalConstructor()
                ->getMock();
        $this->jsonPersonsView =
            $this->getMockBuilder('\view\JsonPersonsView')
                ->disableOriginalConstructor()
                ->getMock();
    }

    public function providerPersons()
    {
        return [
            ['id'=>'1', 'name'=>'testname1'],
            ['id'=>'2', 'name'=>'testname2'],
            ['id'=>'3', 'name'=>'testname3']];
    }
}
```

```

/**
 * @dataProvider providerPersons
 */
public function
testaddPersonById_validPerson_showPersonAndStatus201($id, $name){
    $person=['id'=>$id, 'name'=>$name];
    $this->personModel->expects($this->atLeastOnce())
        ->method('addPersonByIdAndName')
        ->with( $this->equalTo($id), $this->equalTo($name))
        ->will($this->returnValue($person));

    $data=['person' => $person, 'statuscode' => 201];

    $this->jsonPersonView->expects($this->atLeastOnce())
        ->method('show')
        ->with($this->equalTo($data));
    $personController = new PersonController(
        $this->personModel, $this->jsonPersonView,
        $this->jsonPersonsView);
    $personController->addPersonByIdAndName($id,$name);
}
}

```