Spickzettel zu testgetriebener Entwicklung mit PHPUnit

Oliver Klee | typo3-coding@oliverklee.de | @oliklee
https://github.com/oliverklee/tdd-reader

Version 2.0.2, 5. Mai 2017, für PHP >= 5.6 und TYPO3 CMS 7.6

Lizenz

[1] https://github.com/oliverklee/tdd-reader
[2] http://creativecommons.org/licenses/by-sa/4.0/

Inhaltsverzeichnis

1 Benennung von Dateien und Klassen

1.1 Dateinamen

Dateiname des Produktionscodes	Name der Testdatei
`Classes/Domain/Model/Shoe.php`	`Tests/Unit/Domain/Model/ShoeTest.php`
`Classes/Service/BaristaService.php`	`Tests/Unit/Service/BaristaServiceTest.php`

1.2 Klassennamen

Name der Klasse im Produktionscode	Name der Testklasse
`Shoes\Shop\Domain\Model\Shoe`	`Shoes\Shop\Tests\Unit\Domain\Model\ShoeTest`
`Shoes\Shop\Service\BaristaService`	`Shoes\Shop\Tests\Unit\Service\BaristaServiceTest`

2 Struktur von Testklassen

2.1 Extbase-Extensions

Es gibt auf GitHub dazu auch ein Beispielprojekt (das *Tea-Example*):
`https://github.com/oliverklee/ext_tea`

```php
namespace \OliverKlee\Shop\Tests\Unit\Domain\Model;

use OliverKlee\Shop\Domain\Model\Article;

class ArticleTest extends \TYPO3\CMS\Core\Tests\UnitTestCase {
    /**
     * @var Article;
     */
    protected $subject = null;

    protected function setUp()
    {
        $this->subject = new Article;
        $this->subject->initializeObject();
    }

    /**
     * @test
     */
    public function getNameInitiallyReturnsEmptyString()
    {
        self::assertSame('', $this->subject->getName());
    }

    /**
     * @test
     */
    public function setNameSetsName()
    {
        $name = 'foo bar';

        $this->subject->setName($name);

        self::assertSame($name, $this->subject->getName());
    }

    // ...
}
```

2.2 Nicht-Extbase-Extensions

```php
class AttachmentTest extends \Tx_Phpunit_TestCase {
    /**
     * @var \Tx_Oelib_Attachment
     */
```

```
5       protected $subject = null;
6
7       protected function setUp()
8       {
9           $this->subject = new \Tx_Oelib_Attachment();
10      }
11
12      /**
13       * @test
14       */
15      public function getFileNameInitiallyReturnsAnEmptyString()
16      {
17          self::assertSame('', $this->subject->getFileName());
18      }
19
20      /**
21       * @test
22       */
23      public function getFileNameWithFileNameSetReturnsFileName()
24      {
25          $fileName = 'test.txt';
26
27          $this->subject->setFileName($fileName);
28
29          self::assertSame($fileName, $this->subject->getFileName());
30      }
31
32      /**
33       * @test
34       * @expectedException \UnexpectedValueException
35       */
36      public function setFileNameWithEmptyFileNameThrowsException()
37      {
38          $this->subject->setFileName('');
39      }
40
41      // ...
42  }
```

2.3 Nicht-TYPO3-PHP-Project mit Composer

Es gibt auf GitHub dazu auch ein leeres Startprojekt:
`https://github.com/oliverklee/tdd-seed`

2.3.1 composer.json

Diese composer.json installiert PHPUnit und vfsStream für **PHP bis 5.6**:

```
1   {
2       "require-dev": {
3           "phpunit/phpunit": "^5.7.19",
4           "mikey179/vfsStream": "^1.6.4"
```

```
    },
    "autoload": {
        "psr-4": {
            "..."
        }
    },
    "autoload-dev": {
        "psr-4": {
            "..."
        }
    }
}
```

Diese composer.json installiert PHPUnit und vfsStream für **PHP ab 6**:

```
{
    "require-dev": {
        "phpunit/phpunit": "^6.0.13",
        "mikey179/vfsStream": "^1.6.4"
    },
    "autoload": {
        "psr-4": {
            "..."
        }
    },
    "autoload-dev": {
        "psr-4": {
            "..."
        }
    }
}
```

2.3.2 Testcase

```
namespace OliverKlee\Books\Tests\Unit\Domain\Model;

use OliverKlee\Books\Domain\Model;

class BookTest extends \PHPUnit_Framework_TestCase {
    /**
     * @var Book
     */
    protected $subject = null;

    protected function setUp()
    {
        $this->subject = new Book();
    }

    /**
     * @test
```

```
18          */
19      public function getTitleInitiallyReturnsEmptyString()
20      {
21          self::assertSame('', $this->subject->getTitle());
22      }
23
24      /**
25       * @test
26       */
27      public function setTitleSetsTitle()
28      {
29          $title = 'foo bar';
30
31          $this->subject->setTitle($title);
32
33          self::assertSame($title, $this->subject->getTitle());
34      }
35  }
```

3 Tests ausführen

3.1 Nicht-TYPO3-Projekte

3.1.1 Auf der Kommandozeile

```
vendor/bin/phpunit Test/
```

3.1.2 In PhpStorm

1. Settings > Languages & Frameworks > PHP > PHPUnit
2. PHPUnit library > Use Composer autoloader
3. PHPUnit library > Path to script: `vendor/autoload`
4. OK
5. auf den Order `Tests/` rechtsklicken (oder einen anderen Ordner oder eine Testdatei)
6. Run 'Tests'

3.2 TYPO3-Extensions

3.2.1 In PhpStorm

Für eine existierende TYPO3-Installation im Composer-Modus: Bei diesem Ansatz werden alle installierten Extensions geladen, sodass ihr auch die Features der PHPUnit-Extension nutzen könnt.

1. Settings > Languages & Frameworks > PHP > PHPUnit
2. PHPUnit library > Use Composer autoloader
3. PHPUnit library > Path to script: `vendor/autoload` aus dem Document-Root der TYPO3-Installation
4. Test runner > Default configuration file: `typo3/sysext/core/Build/UnitTests.xml` innerhalb des Document-Roots der TYPO3-Installation
5. OK
6. Run > Edit Configurations
7. Defaults > PHPUnit
8. Command Line > Environment variables
9. zwei Variablen hinzufügen:
   - `TYPO3_CONTENT = Development`
   - `TYPO3_PATH_WEB =` der absolute Pfad zum Document-Root der TYPO3-Installation (ohne den Slash am Ende)
10. auf den Order `Tests/` rechtsklicken (oder einen anderen Ordner oder eine Testdatei)
11. Run 'Tests'

Für eine existierende TYPO3-Installation im Klassik-Modus (Nicht-Composer-Modus): In diesem Fall werdet ihr keine Klassen aus anderen Extensions autoloaden können, d. h., ihr werdet auch keine Features der PHPUnit-Extension nutzen können (und auch nicht von anderen Extension-Abhängigkeiten).

1. Wenn ihr den TYPO3-Source per git statt als TAR-Paket heruntergeladen habt, braucht ihr ein `composer install` im TYPO3-Source-Verzeichnis.
2. Settings > Languages & Frameworks > PHP > PHPUnit
3. PHPUnit library > Use Composer autoloader
4. PHPUnit library > Path to script: `vendor/autoload` innerhalb des TYPO3-Source
5. Test runner > Default configuration file: `typo3/sysext/core/Build/UnitTests.xml` innerhalb des TYPO3-Document-Roots
6. OK
7. Run > Edit Configurations
8. Defaults > PHPUnit
9. Command Line > Environment variables
10. zwei Variablen hinzufügen:
    - `TYPO3_CONTENT = Development`
    - `TYPO3_PATH_WEB` = der absolute Pfad zum Document-Root der TYPO3-Installation (ohne den Slash am Ende)
11. auf den Order `Tests/` rechtsklicken (oder einen anderen Ordner oder eine Testdatei)
12. Run 'Tests'

Ohne eine existierende TYPO3-Installation Dieser Ansatz benutzt das TYPO3-Extension-Skelett[3] von Helmut Hummel und Nicole Cordes.

Fügt die folgenden Abschnitte zur `composer.json` eurer Extension hinzu:

```
"require": {
  "typo3/cms": "~7.6.0"
},
"require-dev": {
  "namelesscoder/typo3-repository-client": "^1.2",
  "nimut/testing-framework": "^1.0",
  "mikey179/vfsStream": "^1.4",
  "phpunit/phpunit": "^4.7 || ^5.0"
},
"config": {
  "vendor-dir": ".Build/vendor",
  "bin-dir": ".Build/bin"
},
"scripts": {
  "post-autoload-dump": [
    "mkdir -p .Build/Web/typo3conf/ext/",
    "[ -L .Build/Web/typo3conf/ext/tea ] || ln -snvf ../../../../. .Build/Web/typo3conf/ext/tea"
  ]
```

[3] https://github.com/helhum/ext_scaffold

```
19  },
20  "extra": {
21    "typo3/cms": {
22      "cms-package-dir": "{$vendor-dir}/typo3/cms",
23      "web-dir": ".Build/Web"
24    }
25  }
```

Ersetzt dabei `tea` in Zeile 18 durch den Schlüssel eurer Extension.

Wenn ihr außerdem noch andere, im TER verfügbarer Extensions in den Tests nutzen möchtet (z. B. die PHPUnit-Extension), fügt noch diese Abschnitte zu eurer `composer.json` hinzu (bzw. mergt sie):

```
1  "repositories": [
2  {
3    "type": "composer",
4    "url": "https://composer.typo3.org/"
5  }
6  ],
7  "require-dev": {
8    "typo3-ter/phpunit": "*"
9  },
```

Führt danach diese Schritte in PhpStorm aus:

1. Settings > Languages & Frameworks > PHP > PHPUnit
2. PHPUnit library > Use Composer autoloader
3. PHPUnit library > Path to script:
   - auf den Button *Show hidden files and directories* klicken `.Build/vendor/autoload.php` im Verzeichnis der Extension
4. Test runner > Default configuration file: `typo3/sysext/core/Build/UnitTests.xml` im TYPO3-Source-Verzeichnis
5. OK
6. Run > Edit Configurations
7. Defaults > PHPUnit
8. Command Line > Environment variables
9. zwei Variablen hinzufügen:
   - `TYPO3_CONTENT = Development`
   - `TYPO3_PATH_WEB` = der absolute Pfad zu `.Build/Web` im Verzeichnis der Extension (ohne den Slash am Ende)
10. auf den Order `Tests/` rechtsklicken (oder einen anderen Ordner oder eine Testdatei)
11. Run 'Tests'

3.2.2 Mit dem Backend-Modul der PHPUnit-Extension

Dieser Ansatz funktioniert sowohl für TYPO3-Installationen im Composer-Modus als auch im Klassik-Modus.

Dabei werden alle installierten Extensions geladen (inklusive der PHPUnit-Extension), sodass ihr die Features der PHPUnit-Extension nutzen könnt.

Allerdings werden die Tests dann im Kontext der aktuellen Backend-Session ausgeführt, was die Tests sehr zehrbrechlich macht. Für die meisten Unit-Tests von Extensions funktioniert das, für funktionale Tests wird es hingegen meistens nicht git funktionieren.

1. Admin > PHPUnit

4 Mocks

4.1 Warum mocken?

- um eine Methode „auszuschalten“ (damit sie nicht in die DB schreibt, kein Cruise-Missile abschießt etc.) und `null` zurückzugeben
- um einer Methode einen bestimmten Rückgabewert zu geben oder sie eine Exception werfen zu lassen
- um zu testen, dass eine Methode auf eine bestimmte Art und Weise aufgerufen wird

4.2 Tools für Mocks

Prophecy: Das empfohlene, einfach benutzbare, aktuelle Mocking-Framework. Allerdings kann es keine partiellen Mocks erzeugen.[4]

PHPUnit mocks: Die alte Art, Mocks zu erzeugen. Mocking ist damit etwas unhandlich, aber dafür kann es auch partielle Mocks erzeugen.[5]

Mockery: Auch sehr elegant.[6]

[4] Prophecy-Cheatsheet:
`https://github.com/oliverklee/tdd-reader/blob/master/AdditionalDocuments/prophecy-cheatsheet.pdf`
[5] PHPUnit-Mocking-Cheatsheet:
`https://github.com/oliverklee/tdd-reader/blob/master/AdditionalDocuments/mocking-cheatsheet.pdf`
[6] `https://github.com/mockery/mockery`

5 Auf Exceptions testen

5.1 Nur auf die Klasse testen

```
/**
 * @test
 * @expectedException \UnexpectedValueException
 */
public function createBreadWithNegativeSizeThrowsException()
{
    $this->subject->createBread(-1);
}
```

5.2 Auf Klasse, Nachricht und Code testen

```
/**
 * @test
 * @expectedException \UnexpectedValueException
 * @expectedExceptionMessage size must be > 0.
 * @expectedExceptionCode 1323700434
 */
public function createBreadWithNegativeSizeThrowsException()
{
    $this->subject->createBread(-1);
}
```

6 Abstrakte Klassen testen

6.1 Den PHPUnit-Mock-Builder benutzen

Dies erzeugt eine Instanz der abstrakten Klassen, wobei alle abstrakten Methoden gemockt werden.

```php
namespace OliverKlee\Coffee\Tests\Unit\Domain\Model;

use OliverKlee\Coffee\Domain\Model\AbstractBeverage;

class AbstractBeverageTest {
    /**
     * @var AbstractBeverage|\PHPUnit_Framework_MockObject_MockObject
     */
    protected $subject = null;

    protected function setUp()
    {
        $this->subject = $this->getMockForAbstractClass(
            AbstractBeverage::class
        );
    }
```

6.2 Eine konkrete Unterklasse erstellen

Dies wird empfohlen, wenn die Subklasse zusätzliches oder spezifisches Verhalten haben soll.

Erzeugt in `Tests/Unit/Unit/Domain/Model/Fixtures/` eine Unterklasse der abstrakten Klasse:

```php
namespace OliverKlee\Coffee\Tests\Unit\Domain\Model\Fixtures;

class TestingBeverage extends \OliverKlee\Coffee\Domain\Model\AbstractBeverage {
    // ...
}
```

Dann könnt ihr die konkrete Unterklasse in euren Unit-Tests usen und instanziieren.

```php
use OliverKlee\Coffee\Tests\Unit\Domain\Model\Fixtures\TestingBeverage;

class AbstractBeverageTest {
    /**
     * @var TestingBeverage
     *
    protected $subject = null;

    protected function setUp()
    {
        $this->subject = new TestingBeverage();
    }
```

7 Das Test-Framework der PHPUnit-TYPO3-Extension benutzen

```php
class DataMapperTest extends \Tx_Phpunit_TestCase {
    /**
     * @var \Tx_Phpunit_Framework
     */
    protected $testingFramework = null;

    protected $subject = null;

    protected function setUp()
    {
        $this->testingFramework = new \Tx_Phpunit_Framework('tx_oelib');

        $this->subject = new ...;
    }

    protected function tearDown()
    {
        $this->testingFramework->cleanUp();
    }

    /**
     * @test
     */
    public function findWithUidOfExistingRecordReturnsModelDataFromDatabase()
    {
        $title = 'foo';
        $uid = $this->testingFramework->createRecord(
            'tx_oelib_test', ['title' => $title]
        );

        self::assertSame($title, $this->subject->find($uid)->getTitle());
    }
```

8 Gemockte Dateisystem mit vfsStream benutzen

8.1 Lauffähige Beispiele

Die funktionalen Tests zur FileUtility-Klasse im Tea-Beispiel zeigen, wie Tests mit vfsStream aussehen können.

8.2 Einrichten

```
use org\bovigo\vfs\vfsStream;
use org\bovigo\vfs\vfsStreamDirectory;

/**
 * @var \org\bovigo\vfs\vfsStreamFile
 */
protected $moreStuff;

protected function setUp()
{
    // This is the same as ::register and ::setRoot.
    $this->root = vfsStream::setup('home');
    $this->targetFilePath = vfsStream::url('home/target.txt');

    $this->subject = new ...
}
```

8.3 Die Dateien benutzen

```
/**
 * @test
 */
public function concatenateWithOneEmptySourceFileCreatesEmptyTargetFile()
{
    // This is one way to create a file with contents, using PHP's file functions.
    $sourceFileName = vfsStream::url('home/source.txt');
    // Just calling vfsStream::url does not create the file yet.
    // We need to write into it to create it.
    file_put_contents($sourceFileName, '');

    $this->subject->concatenate($this->targetFilePath, [$sourceFileName]);

    self::assertSame('', file_get_contents($this->targetFilePath));
}

/**
 * @test
 */
public function concatenateWithOneFileCopiesContentsFromSourceFileToTargetFile()
{
    // This is vfsStream's way of creating a file with contents.
    $contents = 'Hello world!';
    $sourceFileName = vfsStream::url('home/source.txt');
    vfsStream::newFile('source.txt')->at($this->root)->setContent($contents);

```

```
27      $this->subject->concatenate($this->targetFilePath, [$sourceFileName]);
28
29      self::assertSame($contents, file_get_contents($this->targetFilePath));
30  }
```

9 PHPUnit-Assertions

Diese Liste ist aktuell für PHPUnit 5.7.x.

```
assertArrayHasKey()
assertClassHasAttribute()
assertArraySubset()
assertClassHasStaticAttribute()
assertContains()
assertContainsOnly()
assertContainsOnlyInstancesOf()
assertCount()
assertDirectoryExists()
assertDirectoryIsReadable()
assertDirectoryIsWritable()
assertEmpty()
assertEqualXMLStructure()
assertEquals()
assertFalse()
assertFileEquals()
assertFileExists()
assertFileIsReadable()
assertFileIsWritable()
assertGreaterThan()
assertGreaterThanOrEqual()
assertInfinite()
assertInstanceOf()
assertInternalType()
assertIsReadable()
assertIsWritable()
assertJsonFileEqualsJsonFile()
assertJsonStringEqualsJsonFile()
assertJsonStringEqualsJsonString()
assertLessThan()
assertLessThanOrEqual()
assertNan()
assertNull()
assertObjectHasAttribute()
assertRegExp()
assertStringMatchesFormat()
assertStringMatchesFormatFile()
assertSame()
assertStringEndsWith()
assertStringEqualsFile()
assertStringStartsWith()
assertThat()
assertTrue()
assertXmlFileEqualsXmlFile()
assertXmlStringEqualsXmlFile()
assertXmlStringEqualsXmlString()
```