

Spickzettel zu testgetriebener Entwicklung mit TYPO3 CMS

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

Version 2.0.0, 24. Oktober 2015, für TYPO3 CMS 6.2

Lizenz

Dieser Reader ist unter einer *Creative-Commons*-Lizenz lizenziert, und zwar konkret unter der *Namensnennung-Weitergabe unter gleichen Bedingungen 4.0 (CC BY-SA 4.0)*. Das bedeutet, dass ihr den Reader unter diesen Bedingungen für euch kostenlos verbreiten, bearbeiten und nutzen könnt (auch kommerziell):

Namensnennung. Ihr müsst den Namen des Autors (Oliver Klee) nennen. Wenn ihr dabei zusätzlich auch noch die Quelle¹ nennt, wäre das nett. Und wenn ihr mir zusätzlich eine Freude machen möchtet, sagt mir per E-Mail Bescheid.

Weitergabe unter gleichen Bedingungen. Wenn ihr diesen Inhalt bearbeitet oder in anderer Weise umgestaltet, verändert oder als Grundlage für einen anderen Inhalt verwendet, dann dürft ihr den neu entstandenen Inhalt nur unter Verwendung identischer Lizenzbedingungen weitergeben.

Lizenz nennen. Wenn ihr den Reader weiter verbreitet, müsst ihr dabei auch die Lizenzbedingungen nennen oder beifügen.

Die ausführliche Version dieser Lizenz findet ihr online.²

¹<https://github.com/oliverklee/tdd-reader>

²<http://creativecommons.org/licenses/by-sa/4.0/>

Inhaltsverzeichnis

1	Benennung von Dateien und Klassen	3
1.1	Dateinamen	3
1.2	Klassennamen	3
2	Struktur von Testklassen	3
2.1	Extbase-Extensions	3
2.2	Nicht-Extbase-Extensions	5
2.3	Nicht-TYPO3-PHP-Project mit Composer	6
2.3.1	composer.json	6
2.3.2	Testcase	7
3	Auf Exceptions testen	7
3.1	Nur auf die Klasse testen	7
3.2	Auf Klasse, Nachricht und Code testen	8
4	Abstrakte Klassen testen	8
4.1	Den PHPUnit-Mock-Builder benutzen	8
4.2	Eine konkrete Unterklasse erstellen	8
5	Das Test-Framework der PHPUnit-TYPO3-Extension benutzen	9
6	Gemockte Dateisystem mit vfsStream benutzen	10
6.1	Lauffähige Beispiele	10
6.2	Einrichten	10
6.3	Die Dateien benutzen	10
7	PHPUnit-Assertions	10

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
pi1/class.tx_frubble_pi1.php	Tests/Unit/pi1/pi1Test.php

1.2 Klassennamen

Name der Klasse im Produktionscode	Name der Testklasse
OliverKlee\Shop\Domain\Model\Shoe	OliverKlee\Shop\Tests\Unit\Domain\Model\ShoeTest
OliverKlee\Shop\Service\BaristaService	OliverKlee\Shop\Tests\Unit\Service\BaristaServiceTest
tx_frubble_pi1	tx_frubble_Tests_Unit_pi1_pi1Test

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

```

1 namespace \OliverKlee\Shop\Tests\Unit\Domain\Model;
2
3 use OliverKlee\Shop\Domain\Model\Article;
4
5 class ArticleTest extends \TYPO3\CMS\Core\Tests\UnitTestCase {
6     /**
7      * @var Article;
8      */
9     protected $subject = null;
10
11     protected function setUp() {
12         $this->subject = new Article;
13         $this->subject->initializeObject();
14     }
15
16     /**
17      * @test
18      */
19     public function getNameInitiallyReturnsEmptyString() {
20         self::assertSame(
21             '',
22             $this->subject->getName()
23         );
24     }
25
26     /**
27      * @test
28      */
29     public function setNameSetsName() {
30         $this->subject->setName('foo bar');
31
32         self::assertSame(
33             'foo bar',
34             $this->subject->getName()
35         );
36     }
37
38     // ...
39 }

```

2.2 Nicht-Extbase-Extensions

```
1 // Hier die zu testende Klasse nur dann einbinden, wenn eure Extension keine
2 // ext_autoload.php benutzt.
3 // require_once(t3lib_extMgm::extPath('oelib') . 'Classes/Attachment.php');
4
5 class Tx_Oelib_Tests_Unit_AttachmentTest extends \Tx_Phunit_TestCase {
6     /**
7      * @var \Tx_Oelib_Attachment
8      */
9     protected $subject = NULL;
10
11     protected function setUp() {
12         $this->subject = new \Tx_Oelib_Attachment();
13     }
14
15     /**
16      * @test
17      */
18     public function getFileNameInitiallyReturnsAnEmptyString() {
19         self::assertSame(
20             '',
21             $this->subject->getFileName()
22         );
23     }
24
25     /**
26      * @test
27      */
28     public function getFileNameWithFileNameSetReturnsFileName() {
29         $this->subject->setFileName('test.txt');
30
31         self::assertSame(
32             'test.txt',
33             $this->subject->getFileName()
34         );
35     }
36
37     /**
38      * @test
39      * @expectedException \InvalidArgumentException
40      */
41     public function setFileNameWithEmptyFileNameThrowsException() {
42         $this->subject->setFileName('');
43     }
44
45     // ...
46 }
```

2.3 Nicht-TYPO3-PHP-Project mit Composer

2.3.1 composer.json

Diese composer.json installiert PHPUnit und vfsStream:

```
1 {
2     "require-dev": {
3         "phpunit/phpunit": "*",
4         "mikey179/vfsStream": "*"
5     },
6     "autoload": {
7         "psr-4": {
8             "...": ""
9         }
10    }
11 }
```

2.3.2 Testcase

```
1 namespace OliverKlee\Books\Tests\Unit\Domain\Model;
2
3 use OliverKlee\Books\Domain\Model;
4
5 class BookTest extends \PHPUnit_Framework_TestCase {
6     /**
7      * @var Book
8      */
9     protected $subject = NULL;
10
11     protected function setUp() {
12         $this->subject = new Book();
13     }
14
15     /**
16      * @test
17      */
18     public function getTitleInitiallyReturnsEmptyString() {
19         self::assertSame(
20             '',
21             $this->subject->getTitle()
22         );
23     }
24
25     /**
26      * @test
27      */
28     public function setTitleSetsTitle() {
29         $this->subject->setTitle('foo bar');
30
31         self::assertSame(
32             'foo bar',
33             $this->subject->getTitle()
34         );
35     }
36 }
```

3 Auf Exceptions testen

3.1 Nur auf die Klasse testen

```
1 /**
2  * @test
3  * @expectedException InvalidArgumentException
4  */
5 public function createBreadWithNegativeSizeThrowsException() {
6     $this->subject->createBread(-1);
7 }
```

3.2 Auf Klasse, Nachricht und Code testen

```
1  /**
2   * @test
3   * @expectedException \InvalidArgumentException
4   * @expectedExceptionMessage size must be > 0.
5   * @expectedExceptionCode 1323700434
6   */
7  public function createBreadWithNegativeSizeThrowsException() {
8      $this->subject->createBread(-1);
9  }
```

4 Abstrakte Klassen testen

4.1 Den PHPUnit-Mock-Builder benutzen

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

```
1  namespace OliverKlee\Coffee\Tests\Unit\Domain\Model;
2
3  use OliverKlee\Coffee\Domain\Model\AbstractBeverage;
4
5  class Tx_Coffee_Domain_Model_AbstractBeverageTest {
6      /**
7       * @var AbstractBeverage|\PHPUnit_Framework_MockObject_MockObject
8       *
9       protected $subject = null;
10
11     protected function setUp() {
12         $this->subject = $this->getMockForAbstractClass(
13             'OliverKlee\\Coffee\\Domain\\Model\\AbstractBeverage'
14         );
15     }
```

4.2 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:

```
1  namespace OliverKlee\Coffee\Tests\Unit\Domain\Model\Fixtures;
2
3  class TestingBeverage extends \OliverKlee\Coffee\Domain\Model\AbstractBeverage {
4      // ...
5  }
```

Dann könnt ihr die konkrete Unterklasse in euren Unit-Tests nutzen und instanziiieren.


```

1 use OliverKlee\Coffee\Tests\Unit\Domain\Model\Fixtures\TestingBeverage;
2
3 class Tx_Coffee_Domain_Model_AbstractBeverageTest {
4     /**
5      * @var TestingBeverage
6      *
7      protected $subject = null;
8
9     protected function setUp() {
10         $this->subject = new TestingBeverage();
11     }

```

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

```

1 class tx_oelib_DataMapperTest extends \Tx_Phpunit_TestCase {
2     /**
3      * @var \Tx_Phpunit_Framework
4      */
5     protected $testingFramework = NULL;
6
7     protected $subject = NULL;
8
9     protected function setUp() {
10         $this->testingFramework = new Tx_Phpunit_Framework('tx_oelib');
11
12         $this->subject = new ...;
13     }
14
15     protected function tearDown() {
16         $this->testingFramework->cleanup();
17     }
18
19     /**
20      * @test
21      */
22     public function findWithUidOfExistingRecordReturnsModelDataFromDatabase() {
23         $uid = $this->testingFramework->createRecord(
24             'tx_oelib_test', array('title' => 'foo')
25         );
26
27         self::assertSame(
28             'foo',
29             $this->subject->find($uid)->getTitle()
30         );
31     }

```

6 Gemockte Dateisystem mit vfsStream benutzen

6.1 Lauffähige Beispiele

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

6.2 Einrichten

```
1 use org\bovigo\vfs\vfsStream;
2 use org\bovigo\vfs\vfsStreamDirectory;
3
4 /**
5  * @var \org\bovigo\vfs\vfsStreamFile
6  */
7 protected $moreStuff;
8
9 protected function setUp() {
10     // This is the same as ::register and ::setRoot.
11     $this->root = vfsStream::setup('home');
12     $this->targetFilePath = vfsStream::url('home/target.txt');
13
14     $this->subject = new ...
15 }
```

6.3 Die Dateien benutzen

```
1  /**
2   * @test
3   */
4  public function concatenateWithOneEmptySourceFileCreatesEmptyTargetFile() {
5      // This is one way to create a file with contents, using PHP's file functions.
6      $sourceFileName = vfsStream::url('home/source.txt');
7      // Just calling vfsStream::url does not create the file yet.
8      // We need to write into it to create it.
9      file_put_contents($sourceFileName, '');
10
11     $this->subject->concatenate($this->targetFilePath, array($sourceFileName));
12
13     self::assertSame(
14         '',
15         file_get_contents($this->targetFilePath)
16     );
17 }
18
19 /**
20 * @test
21 */
22 public function concatenateWithOneFileCopiesContentsFromSourceFileToTargetFile() {
23     // This is vfsStream's way of creating a file with contents.
24     $contents = 'Hello world!';
25     $sourceFileName = vfsStream::url('home/source.txt');
26     vfsStream::newFile('source.txt')->at($this->root)->setContent($contents);
27
28     $this->subject->concatenate($this->targetFilePath, array($sourceFileName));
29
30     self::assertSame(
31         $contents,
32         file_get_contents($this->targetFilePath)
33     );
34 }
```

7 PHPUnit-Assertions

Diese Liste ist aktuell für PHPUnit 4.8.x.

```
assertArrayHasKey()
assertClassHasAttribute()
assertArraySubset()
assertClassHasStaticAttribute()
assertContains()
assertContainsOnly()
assertContainsOnlyInstancesOf()
assertCount()
assertEmpty()
assertEqualXMLStructure()
assertEquals()
assertFalse()
assertFileEquals()
assertFileExists()
assertGreaterThan()
```

assertGreaterThanOrEqual()
assertInstanceOf()
assertInternalType()
assertJsonFileEqualsJsonFile()
assertJsonStringEqualsJsonFile()
assertJsonStringEqualsJsonString()
assertLessThan()
assertLessThanOrEqual()
assertNull()
assertObjectHasAttribute()
assertRegExp()
assertStringMatchesFormat()
assertStringMatchesFormatFile()
assertSame()
assertStringEndsWith()
assertStringEqualsFile()
assertStringStartsWith()
assertThat()
assertTrue()
assertXmlFileEqualsXmlFile()
assertXmlStringEqualsXmlFile()
assertXmlStringEqualsXmlString()