

Spickzettel zu testgetriebener Entwicklung mit TYPO3 CMS

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

Version 1.5.0, 20. Oktober 2015

Lizenz

Dieser Reader ist unter einer *Creative-Commons*-Lizenz lizenziert, und zwar 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	4
2.1	Extbase-Extensions	4
2.2	Nicht-Extbase-Extensions	5
2.3	Nicht-TYPO3-PHP-Project	6
3	Auf Exceptions testen	6
3.1	Nur auf die Klasse testen (empfohlen)	6
3.2	Auf Klasse, Nachricht und (optional) Code testen (empfohlen)	7
3.3	Try/catch (nicht empfohlen)	7
4	Abstrakte Klassen testen	7
4.1	Den PHPUnit-Mock-Builder benutzen (empfohlen)	7
4.2	Eine konkrete Unterklasse erstellen (empfohlen)	8
4.3	Using eval (nicht empfohlen)	8
5	Das Test-Framework der phpunit-TYPO3-Extension benutzen	9
6	Gemockte Dateisystem mit vfsStream benutzen	9
6.1	Einrichten	9
6.2	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
Tx_Life_Domain_Model_Shoe	Tx_Life_Domain_Model_ShoeTest
Tx_Life_Service_BaristaService	Tx_Life_Service_BaristaServiceTest
tx_frubble_pi1	tx_frubble_pi1Test

2 Struktur von Testklassen

2.1 Extbase-Extensions

```
1 class Tx_Articlebase_Domain_Model_ArticleTest extends \Tx_Extbase_Tests_Unit_BaseTestCase {
2     /**
3      * @var \Tx_Articlebase_Domain_Model_Article
4      */
5     protected $subject = NULL;
6
7     public function setUp() {
8         $this->subject = new Tx_Articlebase_Domain_Model_Article();
9         $this->subject->initializeObject();
10    }
11
12    public function tearDown() {
13        unset($this->subject);
14    }
15
16    /**
17     * @test
18     */
19    public function getNameInitiallyReturnsEmptyString() {
20        $this->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        $this->assertSame(
33            'foo bar',
34            $this->subject->getName()
35        );
36    }
37
38    // ...
39 }
```

2.2 Nicht-Extbase-Extensions

```
1  // Hier die zu testende Klasse einbinden, wenn eure Extension keine
2  // ext_autoload.php benutzt.
3  require_once(t3lib_extMgm::extPath('oelib') . 'class.tx_oelib_Attachment.php');
4
5  class tx_oelib_AttachmentTest extends \Tx_Phunit_TestCase {
6      /**
7       * @var \tx_oelib_Attachment
8       */
9      protected $subject = NULL;
10
11     public function setUp() {
12         $this->subject = new tx_oelib_Attachment();
13     }
14
15     public function tearDown() {
16         unset($this->subject);
17     }
18
19     /**
20      * @test
21      */
22     public function getFileNameInitiallyReturnsAnEmptyString() {
23         $this->assertSame(
24             '',
25             $this->subject->getFileName()
26         );
27     }
28
29     /**
30      * @test
31      */
32     public function getFileNameWithFileNameSetReturnsFileName() {
33         $this->subject->setFileName('test.txt');
34
35         $this->assertSame(
36             'test.txt',
37             $this->subject->getFileName()
38         );
39     }
40
41     /**
42      * @test
43      */
44     public function setFileNameWithEmptyFileNameThrowsException() {
45         $this->setExpectedException('InvalidArgumentException', '$fileName must not be empty.');
```

```
46
47         $this->subject->setFileName('');
48     }
49
50     // ...
51 }
```

2.3 Nicht-TYPO3-PHP-Project

```
1 namespace Books\Domain\Model;
2
3 $currentDirectory = dirname(__FILE__);
4 require_once($currentDirectory . '/../Classes/Domain/Model/Book.php');
5
6 class BookTest extends \PHPUnit_Framework_TestCase {
7     /**
8      * @var Book
9      */
10    protected $subject = NULL;
11
12    public function setUp() {
13        $this->subject = new Book();
14    }
15
16    public function tearDown() {
17        unset($this->subject);
18    }
19
20    /**
21     * @test
22     */
23    public function getTitleInitiallyReturnsEmptyString() {
24        $this->assertSame(
25            '',
26            $this->subject->getTitle()
27        );
28    }
29
30    /**
31     * @test
32     */
33    public function setTitleSetsTitle() {
34        $this->subject->setTitle('foo bar');
35
36        $this->assertSame(
37            'foo bar',
38            $this->subject->getTitle()
39        );
40    }
41 }
```

3 Auf Exceptions testen

3.1 Nur auf die Klasse testen (empfohlen)

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

3.2 Auf Klasse, Nachricht und (optional) Code testen (empfohlen)

```
1  /**
2   * @test
3   */
4  public function createBreadWithNegativeSizeThrowsException() {
5      $this->setExpectedException(
6          'InvalidArgumentException',
7          '$size must be > 0.',
8          1323700434
9      );
10
11     $this->subject->createBread(-1);
12 }
13
14 /**
15  * @test
16  */
17 public function createBreadWithZeroSizeThrowsException() {
18     $this->setExpectedException(
19         'InvalidArgumentException',
20         '$size must be > 0.'
21     );
22
23     $this->subject->createBread(-1);
24 }
```

3.3 Try/catch (nicht empfohlen)

```
1  /**
2   * @test
3   */
4  public function createBreadWithNegativeSizeThrowsException() {
5      try {
6          $this->subject->createBread(-1);
7          $this->fail('The expected exception has not been thrown.');
```

4 Abstrakte Klassen testen

4.1 Den PHPUnit-Mock-Builder benutzen (empfohlen)

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

```
1  class Tx_Coffee_Domain_Model_AbstractBeverageTest {
2      /**
3       * @var \Tx_Coffee_Domain_Model_AbstractBeverage|\PHPUnit_Framework_MockObject_MockObject
4       */
5      protected $subject = NULL;
6
7      protected function setUp() {
8          $this->subject = $this->getMockForAbstractClass('Tx_Coffee_Domain_Model_AbstractBeverage');
```

4.2 Eine konkrete Unterklasse erstellen (empfohlen)

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

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

```
1 class Tx_Coffee_Domain_Model_TestingBeverage extends \Tx_Coffee_Domain_Model_AbstractBeverage {
2     // ...
3 }
```

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

```
1 require_once(t3lib_extMgm::extPath('coffee') . 'Tests/Unit/Fixtures/TestingBeverage.php');
2
3 class Tx_Coffee_Domain_Model_AbstractBeverageTest {
4     /**
5      * @var \Tx_Coffee_Domain_Model_TestingBeverage
6      *
7      protected $subject = NULL;
8
9     protected function setUp() {
10         $this->subject = new Tx_Coffee_Domain_Model_TestingBeverage();
11     }
```

4.3 Using eval (nicht empfohlen)

Dies ist nicht empfehlenswert, weil ihr damit die Code-Vervollständigung in eurer IDE kaputt-macht.

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     public function setUp() {
10         $this->testingFramework = new Tx_Phpunit_Framework('tx_oelib');
11
12         $this->subject = new ...;
13     }
14
15     public function tearDown() {
16         $this->testingFramework->cleanUp();
17
18         unset($this->subject, $this->testingFramework);
19     }
20
21     /**
22      * @test
23      */
24     public function findWithUIdOfExistingRecordReturnsModelDataFromDatabase() {
25         $uid = $this->testingFramework->createRecord(
26             'tx_oelib_test', array('title' => 'foo')
27         );
28
29         $this->assertSame(
30             'foo',
31             $this->subject->find($uid)->getTitle()
32         );
33     }
34 }
```

6 Gemockte Dateisystem mit vfsStream benutzen

6.1 Einrichten

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

6.2 Die Dateien benutzen

```
1  /**
2   * @test
3   */
4  public function checkFilePathOfExistingNonEmptyFileReturnsTrue() {
5      $file = vfsStream::newFile('test.php')->at($this->moreStuff);
6      $file->withContent('Hello world!');
7
8      $this->assertTrue(
9          $this->subject->checkFile(vfsStream::url('Stuff/moreStuff/test.php'))
10     );
11 }
```

7 PHPUnit-Assertions

Diese Liste ist aktuell für PHPUnit 3.7.x.

```
assertArray[Not]HasKey(mixed $key, array $array[, string $message = ''])
assertClass[Not]HasAttribute(string $attributeName, string $className[, string $message = ''])
assertClass[Not]HasStaticAttribute(string $attributeName, string $className[, string $message = ''])
assert[Not]Contains(mixed $needle, Iterator|array $haystack[, string $message = ''])
assert[Not]ContainsOnly(string $type, Iterator|array $haystack[, boolean $isNativeType = NULL, string $message = ''])
assertContainsOnlyInstancesOf(string $classname, Traversable|array $haystack[, string $message = ''])
assert[Not]Count($expectedCount, $haystack[, string $message = ''])
assert[Not]Empty(mixed $actual[, string $message = ''])
assertEqualXMLStructure(DOMElement $expectedElement, DOMElement $actualElement[, boolean $checkAttributes = FALSE, string $message = ''])
assert[Not]Equals(mixed $expected, mixed $actual[, string $message = ''])
assertFalse(bool $condition[, string $message = ''])
assertFile[Not]Equals(string $expected, string $actual[, string $message = ''])
assertFile[Not]Exists(string $filename[, string $message = ''])
assertGreaterThan(mixed $expected, mixed $actual[, string $message = ''])
assertGreaterThanOrEqual(mixed $expected, mixed $actual[, string $message = ''])
assert[Not]InstanceOf($expected, $actual[, $message = ''])
assert[Not]InternalType($expected, $actual[, $message = ''])
assertJsonFileEqualsJsonFile(mixed $expectedFile, mixed $actualFile[, string $message = ''])
assertJsonStringEqualsJsonFile(mixed $expectedFile, mixed $actualJson[, string $message = ''])
assertJsonStringEqualsJsonString(mixed $expectedJson, mixed $actualJson[, string $message = ''])
assertLessThan(mixed $expected, mixed $actual[, string $message = ''])
assertLessThanOrEqual(mixed $expected, mixed $actual[, string $message = ''])
assert[Not]Null(mixed $variable[, string $message = ''])
assertObject[Not]HasAttribute(string $attributeName, object $object[, string $message = ''])
assert[Not]RegExp(string $pattern, string $string[, string $message = ''])
assertString[Not]MatchesFormat(string $format, string $string[, string $message = ''])
assertString[Not]MatchesFormatFile(string $formatFile, string $string[, string $message = ''])
assert[Not]Same(mixed $expected, mixed $actual[, string $message = ''])
assertSelectCount(array $selector, integer $count, mixed $actual[, string $message = '', boolean $isHtml = TRUE])
assertSelectEquals(array $selector, string $content, integer $count, mixed $actual[, string $message = '', boolean $isHtml = TRUE])
assertSelectRegExp(array $selector, string $pattern, integer $count, mixed $actual[, string $message = '', boolean $isHtml = TRUE])
assertStringEnds[Not]With(string $suffix, string $string[, string $message = ''])
assertString[Not]EqualsFile(string $expectedFile, string $actualString[, string $message = ''])
assertStringStarts[Not]With(string $prefix, string $string[, string $message = ''])
assertTag(array $matcher, string $actual[, string $message = '', boolean $isHtml = TRUE])
assertThat(mixed $value, PHPUnit_Framework_Constraint $constraint[, $message = ''])
assertTrue(bool $condition[, string $message = ''])
assertXmlFile[Not]EqualsXmlFile(string $expectedFile, string $actualFile[, string $message = ''])
assertXmlString[Not]EqualsXmlFile(string $expectedFile, string $actualXml[, string $message = ''])
assertXmlString[Not]EqualsXmlString(string $expectedXml, string $actualXml[, string $message = ''])
```