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

[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`

```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() {
    $this->subject->setName('foo bar');

    self::assertSame(
      'foo bar',
      $this->subject->getName()
    );
  }

  // ...
}
```

2.2 Nicht-Extbase-Extensions

```php
// Hier die zu testende Klasse nur dann einbinden, wenn eure Extension keine
// ext_autoload.php benutzt.
// require_once(t3lib_extMgm::extPath('oelib') . 'Classes/Attachment.php');

class Tx_Oelib_Tests_Unit_AttachmentTest extends \Tx_Phpunit_TestCase {
  /**
   * @var \Tx_Oelib_Attachment
   */
  protected $subject = NULL;

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

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

  /**
   * @test
   */
  public function getFileNameWithFileNameSetReturnsFileName() {
    $this->subject->setFileName('test.txt');

    self::assertSame(
      'test.txt',
      $this->subject->getFileName()
    );
  }

  /**
   * @test
   * @expectedException \InvalidArgumentException
   */
  public function setFileNameWithEmptyFileNameThrowsException() {
    $this->subject->setFileName('');
  }

  // ...
}
```

2.3 Nicht-TYPO3-PHP-Project mit Composer

2.3.1 composer.json

Diese composer.json installiert PHPUnit und vfsStream:

```
{
    "require-dev": {
        "phpunit/phpunit": "*",
        "mikey179/vfsStream": "*"
    },
    "autoload": {
        "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
   */
  public function getTitleInitiallyReturnsEmptyString() {
    self::assertSame(
      '',
      $this->subject->getTitle()
    );
  }

  /**
   * @test
   */
  public function setTitleSetsTitle() {
    $this->subject->setTitle('foo bar');

    self::assertSame(
      'foo bar',
      $this->subject->getTitle()
    );
  }
}
```

3 Auf Exceptions testen

3.1 Nur auf die Klasse testen

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

3.2 Auf Klasse, Nachricht und Code testen

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

4 Abstrakte Klassen testen

4.1 Den PHPUnit-Mock-Builder benutzen

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

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

use OliverKlee\Coffee\Domain\Model\AbstractBeverage;

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

 protected function setUp() {
   $this->subject = $this->getMockForAbstractClass(
     'OliverKlee\\Coffee\\Domain\\Model\\AbstractBeverage'
   );
 }
```

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:

```
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.

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

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

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

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

```
class tx_oelib_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() {
    $uid = $this->testingFramework->createRecord(
      'tx_oelib_test', array('title' => 'foo')
    );

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

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

```
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 ...
}
```

6.3 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, array($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);

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

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

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()
```