

Cheatsheet for test-driven development with TYPO3 CMS

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

Version 2.0.0, October 24, 2015, for TYPO3 CMS 6.2

License

This handout is licensed under a *Creative Commons* license, in this case under an *Attribution-ShareAlike 4.0 (CC BY-SA 4.0)*. This means that you can use, edit and distribute this handout (even commercially) under the following conditions:

Attribution. You need to give credit to the author (me) by listing my name (Oliver Klee). If you also list the source¹, that would be nice. And if you want to make me happy, please drop me an e-mail if you use this document.

ShareAlike. If you edit or change this document or use it as a basis for some other document, you must use the same license for the resulting document.

Name the license. If you distribute this document, you'll need to mention or enclose the license.

You can find a more comprehensive version of this license online.²

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

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

Contents

1	File and class naming	3
1.1	File names	3
1.2	Class names	3
2	Test class structure	3
2.1	Extbase extensions	3
2.2	Non-extbase extensions	5
2.3	Non-TYPO3 PHP projects with Composer	6
2.3.1	composer.json	6
2.3.2	Test case	7
3	Testing for Exceptions	7
3.1	Test for the Exception class only	7
3.2	Test for the exception class, message and the code	8
4	Testing abstract classes	8
4.1	Using the PHPUnit mock builder	8
4.2	Creating a concrete subclass	8
5	Using the testing framework of the PHPUnit TYPO3 extension	9
5.1	Executable examples	9
6	Using mock file systems with vfsStream	10
6.1	Setting it all up	10
6.2	Using the files	10
7	PHPUnit assertions	11

1 File and class naming

1.1 File names

Production code file name	Test file name
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 Class names

Production code class name	Test class name
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 Test class structure

2.1 Extbase extensions

There's an example project (the tea example) for this on GitHub: 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 Non-extbase extensions

```
1 // You need to require_once the class to test only if your extension
2 // does not make use of ext_autoload.php.
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 Non-TYPO3 PHP projects with Composer

2.3.1 composer.json

This setup installs PHPUnit and vfsStream:

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

2.3.2 Test case

```
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 Testing for Exceptions

3.1 Test for the Exception class only

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

3.2 Test for the exception class, message and the code

```
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 Testing abstract classes

4.1 Using the PHPUnit mock builder

This will create an instance of the abstract class with all abstract methods mocked.

```
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 Creating a concrete subclass

This is recommended if you need to provide your subclass with some additional or specific behavior.

In Tests/Unit/Domain/Model/Fixtures/, create a subclass of the abstract class:

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

Then you can use and instantiate the concrete subclass in your unit tests:


```

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 Using the testing framework of the PHPUnit TYPO3 extension

```

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     }

```

5.1 Executable examples

The functional tests for the FileUtility class in the tea example show what tests with vfsStream can look like.

6 Using mock file systems with vfsStream

6.1 Setting it all up

```
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 }
```

6.2 Using the files

```
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

This list is current for 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()
```