Cheatsheet for test-driven development with TYPO3 CMS

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

Version 2.0.1, May 7, 2016, 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 source[1], 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

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`

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`

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

```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 Non-extbase extensions

```php
1   class AttachmentTest extends \Tx_Phpunit_TestCase {
2       /**
3        * @var \Tx_Oelib_Attachment
4        */
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 \InvalidArgumentException
35       */
36      public function setFileNameWithEmptyFileNameThrowsException()
37      {
38          $this->subject->setFileName('');
39      }
40
41      // ...
42  }
```

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": "4.8.*",
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      {
13          $this->subject = new Book();
14      }
15
16      /**
17       * @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('foo bar', $this->subject->getTitle());
34      }
35  }
```

3 Testing for Exceptions

3.1 Test for the Exception class only

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

3.2 Test for the exception class, message and the code

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

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.

```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
        );
    }
}
```

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:

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

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

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

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

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

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

5 Using the testing framework of the PHPUnit TYPO3 extension

```php
1  class 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     {
11         $this->testingFramework = new \Tx_Phpunit_Framework('tx_oelib');
12
13         $this->subject = new ...;
14     }
15
16     protected function tearDown()
17     {
18         $this->testingFramework->cleanUp();
19     }
20
21     /**
22      * @test
23      */
24     public function findWithUidOfExistingRecordReturnsModelDataFromDatabase()
25     {
26         $title = 'foo';
27         $uid = $this->testingFramework->createRecord(
28             'tx_oelib_test', ['title' => $title]
29         );
30
31         self::assertSame($title, $this->subject->find($uid)->getTitle());
32     }
```

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

```php
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 ...
}
```

6.2 Using the files

```php
/**
 * @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);

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

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

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()
```