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

[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
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

```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 Non-extbase extensions

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

2.3.1 composer.json

This setup installs PHPUnit and vfsStream:

```
{
    "require-dev": {
        "phpunit/phpunit": "*",
        "mikey179/vfsStream": "*"
    },
    "autoload": {
        "psr-4": {
            "..."
        }
    }
}
```

2.3.2 Test case

```
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 Testing for Exceptions

3.1 Test for the Exception class only

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

3.2 Test for the exception class, message and the code

```
/**
 * @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.

```
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 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:

```
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:

```
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 Using the testing framework of the PHPUnit TYPO3 extension

```
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()
        );
    }
```

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

```
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

```
/**
 * @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

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()
```