Cheatsheet for test-driven development with PHPUnit

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

Version 2.0.2, May 5, 2017, for TYPO3 PHP >= 5.6 and CMS 7.6

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

1.2 Class names

Production code class name	Test class name
Shoes\Shop\Domain\Model\Shoe	Shoes\Shop\Tests\Unit\Domain\Model\ShoeTest
Shoes\Shop\Service\BaristaService	Shoes\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
class 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()
    {
        $fileName = 'test.txt';

        $this->subject->setFileName($fileName);

        self::assertSame($fileName, $this->subject->getFileName());
    }

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

    // ...
}
```

2.3 Non-TYPO3 PHP projects with Composer

There is an empty starter project for this on GitHub:
`https://github.com/oliverklee/tdd-seed`

2.3.1 composer.json

This setup installs PHPUnit and vfsStream **for PHP up to 5.6**:

```
{
    "require-dev": {
        "phpunit/phpunit": "^5.7.19",
        "mikey179/vfsStream": "^1.6.4"
```

```
    },
    "autoload": {
        "psr-4": {
            "..."
        }
    },
    "autoload-dev": {
        "psr-4": {
            "..."
        }
    }
}
```

This setup installs PHPUnit and vfsStream **for PHP 6**:

```
{
    "require-dev": {
        "phpunit/phpunit": "^6.0.13",
        "mikey179/vfsStream": "^1.6.4"
    },
    "autoload": {
        "psr-4": {
            "..."
        }
    },
    "autoload-dev": {
        "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
```

```
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 Executing the tests

3.1 Non-TYPO3 projects

3.1.1 On the command line

```
vendor/bin/phpunit Test/
```

3.1.2 Within PhpStorm

1. Settings > Languages & Frameworks > PHP > PHPUnit
2. PHPUnit library > Use Composer autoloader
3. PHPUnit library > Path to script: `vendor/autoload`
4. OK
5. right-click on the `Tests/` folder (or any test file or folder)
6. Run 'Tests'

3.2 TYPO3 extensions

3.2.1 Within PhpStorm

For an existing TYPO3 installation in Composer mode: This will also load all existing extensions (including the PHPUnit extension), making it possible to use the features of the PHPUnit extension.

1. Settings > Languages & Frameworks > PHP > PHPUnit
2. PHPUnit library > Use Composer autoloader
3. PHPUnit library > Path to script: `vendor/autoload` within the TYPO3 document root
4. Test runner > Default configuration file: `typo3/sysext/core/Build/UnitTests.xml` within the TYPO3 document root
5. OK
6. Run > Edit Configurations
7. Defaults > PHPUnit
8. Command Line > Environment variables
9. add two variables:
   - `TYPO3_CONTENT = Development`
   - `TYPO3_PATH_WEB` = the absolute path to the TYPO3 document root (without the trailing slash)
10. right-click on the `Tests/` folder (or any test file or folder)
11. Run 'Tests'

For an existing TYPO3 installation in classic mode (non-Composer mode): In this case, you will not be able to autoload any classe from other extensions, i. e., you will not be able to use any features of the PHPUnit extension (or of any other extension dependencies).

1. If you have downloaded the TYPO3 source via git instead of as a TAR package, you'll need to do a `composer install` in the TYPO3 source directory.
2. Settings > Languages & Frameworks > PHP > PHPUnit
3. PHPUnit library > Use Composer autoloader
4. PHPUnit library > Path to script: `vendor/autoload` within the TYPO3 source
5. Test runner > Default configuration file: `typo3/sysext/core/Build/UnitTests.xml` within the TYPO3 document root
6. OK
7. Run > Edit Configurations
8. Defaults > PHPUnit
9. Command Line > Environment variables
10. add two variables:
    - `TYPO3_CONTENT = Development`
    - `TYPO3_PATH_WEB` = the absolute path to the TYPO3 document root (without the trailing slash)
11. right-click on the `Tests/` folder (or any test file or folder)
12. Run 'Tests'

Without using an existing TYPO3 installation This is the approach used in the TYPO3 extension skeleton[3] by Helmut Hummel and Nicole Cordes.

Add the following sections the `composer.json` of your extension:

```
"require": {
  "typo3/cms": "~7.6.0"
},
"require-dev": {
  "namelesscoder/typo3-repository-client": "^1.2",
  "nimut/testing-framework": "^1.0",
  "mikey179/vfsStream": "^1.4",
  "phpunit/phpunit": "^4.7 || ^5.0"
},
"config": {
  "vendor-dir": ".Build/vendor",
  "bin-dir": ".Build/bin"
},
"scripts": {
  "post-autoload-dump": [
    "mkdir -p .Build/Web/typo3conf/ext/",
    "[ -L .Build/Web/typo3conf/ext/tea ] || ln -snvf ../../../../. .Build/Web/typo3conf/ext/tea"
  ]
},
```

[3] `https://github.com/helhum/ext_scaffold`

```
20 "extra": {
21   "typo3/cms": {
22     "cms-package-dir": "{$vendor-dir}/typo3/cms",
23     "web-dir": ".Build/Web"
24   }
25 }
```

You'll need to replace `tea` in line 18 with the key of your extension.

If you'd like to use other extensions that are available from the TER (e. g., the PHPUnit extension), you'll need to add (or merge) these sections to your `composer.json`:

```
"repositories": [
{
  "type": "composer",
  "url": "https://composer.typo3.org/"
}
],
"require-dev": {
  "typo3-ter/phpunit": "*"
},
```

Then do the following in PhpStorm:

1. Settings > Languages & Frameworks > PHP > PHPUnit
2. PHPUnit library > Use Composer autoloader
3. PHPUnit library > Path to script:
   - click on the *Show hidden files and directories* button `.Build/vendor/autoload.php` within the extension directory
4. Test runner > Default configuration file: `typo3/sysext/core/Build/UnitTests.xml` within the TYPO3 source
5. OK
6. Run > Edit Configurations
7. Defaults > PHPUnit
8. Command Line > Environment variables
9. add two variables:
   - `TYPO3_CONTENT = Development`
   - `TYPO3_PATH_WEB` = the absolute path to `.Build/Web` folder in your extension directory (without the trailing slash)
10. right-click on the `Tests/` folder (or any test file or folder)
11. Run 'Tests'

3.2.2 Using the PHPUnit back-end module

This works for TYPO3 installations both in Composer mode and in classic mode.

This will load all installed extensions (including the PHPUnit extension), making it possible to use the features of the PHPUnit extension.

However, this will also execute the tests in the current back-end context, making the tests very brittle. This works fine for most unit tests of extensions, but will not work for functional tests.

1. Admin > PHPUnit

4 Mocks

4.1 Why mock?

- to "disable" a method (to not write to the DB, or to not launch a cruise missile) and return null
- to have a method redurn a particular return value or throw an exception
- to test that a method gets called in a certain way

4.2 Tools for mocking

Prophecy: The recommended, state-of the art, easy-to-use mocking framework. It cannot create partial mocks, though.[4]

PHPUnit mocks: The old way of creating mocks. Creating mocks is a bit unwieldy, but it can create partial mocks.[5]

Mockery: Also very elegant.[6]

[4] Prophecy cheatsheet:
`https://github.com/oliverklee/tdd-reader/blob/master/AdditionalDocuments/prophecy-cheatsheet.pdf`
[5] PHPUnit mocking cheatsheet:
`https://github.com/oliverklee/tdd-reader/blob/master/AdditionalDocuments/mocking-cheatsheet.pdf`
[6] `https://github.com/mockery/mockery`

5 Testing for Exceptions

5.1 Test for the Exception class only

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

5.2 Test for the exception class, message and the code

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

6 Testing abstract classes

6.1 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
        );
    }
```

6.2 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();
    }
```

7 Using the testing framework of the PHPUnit TYPO3 extension

```php
class 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()
    {
        $title = 'foo';
        $uid = $this->testingFramework->createRecord(
            'tx_oelib_test', ['title' => $title]
        );

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

7.1 Executable examples

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

8 Using mock file systems with vfsStream

8.1 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 ...
}
```

8.2 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, [$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));
}
```

9 PHPUnit assertions

This list is current for PHPUnit 5.7.x.

```
assertArrayHasKey()
assertClassHasAttribute()
assertArraySubset()
assertClassHasStaticAttribute()
assertContains()
assertContainsOnly()
assertContainsOnlyInstancesOf()
assertCount()
assertDirectoryExists()
assertDirectoryIsReadable()
assertDirectoryIsWritable()
assertEmpty()
assertEqualXMLStructure()
assertEquals()
assertFalse()
assertFileEquals()
assertFileExists()
assertFileIsReadable()
assertFileIsWritable()
assertGreaterThan()
assertGreaterThanOrEqual()
assertInfinite()
assertInstanceOf()
assertInternalType()
assertIsReadable()
assertIsWritable()
assertJsonFileEqualsJsonFile()
assertJsonStringEqualsJsonFile()
assertJsonStringEqualsJsonString()
assertLessThan()
assertLessThanOrEqual()
assertNan()
assertNull()
assertObjectHasAttribute()
assertRegExp()
assertStringMatchesFormat()
assertStringMatchesFormatFile()
assertSame()
assertStringEndsWith()
assertStringEqualsFile()
assertStringStartsWith()
assertThat()
assertTrue()
assertXmlFileEqualsXmlFile()
assertXmlStringEqualsXmlFile()
assertXmlStringEqualsXmlString()
```