

Testgetriebene Entwicklung mit PHPUnit

- PHP Professional Training
- 22th of November 2007

About me

- **Kore Nordmann**
 - Studying computer science at the University Dortmund
 - Working for eZ systems on eZ components
 - Maintainer and / or Developer in multiple open source projects: Image_3D, KaForkl, ezcGraph, Torii, Business, PHPUnit, WCV, ...
 - Talks about this, because:
 - Maintainer of MockObjects in PHPUnit
 - Develops in test driven developed projects

•Start

- Ever written bugfree code?

•Agenda

- Why you should (unit)test
- Writing tests with PHPUnit
- MockObjects
- Selenium
- Continuous Integration

•Goal of testing

- Write software with less bugs
 - More customer satisfaction
 - Spend less time on software maintenance
- Test complete public API
 - Ensure BC (backwards compability)

• "Old school" development

- Write requirements specification
- Design software
- Implementation phase
- Testing phase

•Test driven development

- Write requirements specification
- Write tests
 - The tests specify the public API
 - The tests specify the internal APIs
- Make tests pass

•Testing helps!

- Microsoft Case Study
 - TDD project has twice the code quality
 - Writing tests requires 15% more time
- IBM Case Study
 - 40% fewer defects
 - No impact on the team's productivity
- John Deere / Ericsson Case Study
 - TDD produces higher quality code
 - Impact of 16% on the team's productivity

•Example ezcGraph

- Really complex project
 - 81k lines of code (including comments)
 - 66 bugs (including documentation bugs) in 1,5 years of development.
- Warm and fuzzy feeling about BC, stability, ... ;)
 - Test output helps debugging strange platform issues

•What should be tested? (1)

- Everything :)
 - Test successful execution
 - Test for failures
 - Test for edgecases

•What should be tested? (2)

- Backend
 - Business logic
 - Components / Libraries
- Frontend
 - User interaction (forms)
 - Content integration (templates)
 - Rich interfaces (AJAX, ...)
 - Webservices

•Types of tests (1)

- Backend
 - Functional testing (UnitTests)
- Frontend
 - System tests (through the browser, UnitTests)
 - Compability tests (different OSs / browsers)
 - Performance tests / security tests / acceptance tests

•Types of tests (2)

- Unit tests
 - Test code fragments (units) of the software
 - Separates your code into fine grained testable units
- System tests
 - Test the behaviour of a complete system
 - Test the response of a webservice
 - Test the end result of a library
 - Use browsers to test a web application

•Types of tests (3)

- Acceptance tests
 - Ensure the applications meets the customers expectations
- Performance tests
 - Test for: performance, load, stress, reliability
 - Tools: ab, httpperf, siege
- Security tests
 - Automatic scanning for common vulnerabilities
 - For example: Chorizo
 - Manual audits (SectionEins GmbH)

•Agenda

- Why you should (unit)test
- Writing tests with PHPUnit
- MockObjects
- Selenium
- Continuous Integration



•PHPUnit

- Installation

- With pear

- pear channel-discover pear.phpunit.de

- pear install phpunit/PHPUnit

- Details:

- http://www.phpunit.de/pocket_guide/3.2/en/installation.html

- Documentation

- http://www.phpunit.de/pocket_guide/3.2/en/index.html

•What will we test?

- A very simple vector class.
 - Does not really matter ;)
- Test Setup
 - component/
 - VectorTest.php
 - Vector.php
 - Tests/
 - Create.php
 - Add.php

•1: The test suite

■ Groups your test classes

■ `// Require tests classes - you may just use your autoload mechanism here...
require_once dirname(__FILE__) . '/Vector.php';`

`// Include tests
require_once dirname(__FILE__) . '/Tests/Create.php';`

```
class VectorTest extends PHPUnit_Framework_TestSuite
{
    public function __construct()
    {
        parent::__construct();
        $this->setName( 'Vector Tests' );

        $this->addTest( VectorCreateTest::suite() );
    }

    public static function suite()
    {
        return new VectorTest( __CLASS__ );
    }
}
```

•1: The test

- Contains the actual tests

- ```
class VectorCreateTest extends PHPUnit_Framework_TestCase
{
 public static function suite()
 {
 return new PHPUnit_Framework_TestSuite(__CLASS__);
 }
}
```

# 1: The result

- Output on CLI

- ```
$ phpunit VectorTest
PHPUnit 3.2.0 by Sebastian Bergmann.
```

```
F
```

```
Time: 0 seconds
```

```
There was 1 failure:
```

```
1) Warning(PHPUnit_Framework_Warning)
No tests found in class "VectorCreateTest".
```

```
FAILURES!
```

```
Tests: 1, Failures: 1.
```



2: Add some tests

- Test vector creation

- ```
public function testCreateDefaultVector()
{
 $vector = new Vector();

 $this->assertSame($vector->x, 0);
 $this->assertSame($vector->y, 0);
}

public function testCreateVector()
{
 $vector = new Vector(1, 2);

 $this->assertSame($vector->x, 1);
 $this->assertSame($vector->y, 2);
}
```

## 2: Also test for potential failures

- Also test for potential failures

- ```
public function testCreateVectorFromStrings()
{
    try
    {
        $vector = new Vector( 'invalid' );
    }
    catch ( Exception $e )
    {
        // Expected
        return;
    }

    $this->fail( 'Expected Exception.' );
}
```

2: Run the tests

```
■ $ phpunit VectorTest
PHPUnit 3.2.0 by Sebastian Bergmann.

EEF

Time: 0 seconds

There were 2 errors:

1) testCreateDefaultVector(VectorCreateTest)
Undefined property: Vector::$x
.../Tests/Create.php:14

2) testCreateVector(VectorCreateTest)
Undefined property: Vector::$x
.../Tests/Create.php:22

--

There was 1 failure:

1) testCreateVectorFromStrings(VectorCreateTest)
Expected Exception.
.../Tests/Create.php:38

FAILURES!
Tests: 3, Failures: 1, Errors: 2.
```

3: Start the implementation

```
class Vector
{
    public $x;
    public $y;

    public function __construct( $x = 0, $y = 0 )
    {
        if ( !is_numeric( $x ) || !is_numeric( $y ) )
        {
            throw new Exception( 'Only numeric values are
allowed as parameters.' );
        }

        $this->x = $x;
        $this->y = $y;
    }
}
```


3: Run the tests

- `$ phpunit VectorTest`
PHPUnit 3.2.0 by Sebastian Bergmann.

...

Time: 0 seconds

- OK (3 tests)



4: Add another test class

- class VectorTest extends PHPUnit_Framework_TestSuite
{
 public function __construct()
 {
 parent::__construct();
 \$this->setName('Vector Tests');

 \$this->addTest(VectorCreateTest::suite());
 \$this->addTest(VectorAddTest::suite());
 }
 // ...
}

4: Add one new test

```
■ /**
 * Test vector sum with data provider
 *
 * @dataProvider vectorDataProvider
 */
public function testSumVectors(
    Vector $sum1, Vector $sum2, Vector $res )
{
    $this->assertEquals(
        $sum1->add( $sum2 ),
        $res
    );
}
```

4: Define test data

```
public static function vectorDataProvider()
{
    return array(
        array(
            new Vector(), new Vector(), new Vector(),
        ),
        array(
            new Vector( 1, 2 ), new Vector( 2, 3 ), new
Vector( 3, 5 ),
        ),
        array(
            new Vector(), new Vector( .3, .74 ), new Vector(
.3, .74 ),
        ),
        array(
            new Vector( 1 ), new Vector( -12, 2 ), new
Vector( -11, 2 ),
        ),
    );
}
```

4: Run the tests

- `$ phpunit VectorTest`
PHPUnit 3.2.0 by Sebastian Bergmann.

```
...PHP Fatal error: Call to undefined method Vector::add()  
in .../Tests/Add.php on line 36
```



5: Implement the addition

- public function add(\$vector)
{
 \$this->x += \$vector->x;
 \$this->y += \$vector->y;

 return \$this;
}

5: Run the tests again

- `$ phpunit VectorTest`
PHPUnit 3.2.0 by Sebastian Bergmann.

.....

Time: 0 seconds

OK (7 tests)



5: Check the coverage report

- `$ phpunit --coverage-html report/ VectorTest`
PHPUnit 3.2.0 by Sebastian Bergmann.

.....

Time: 0 seconds

OK (7 tests)

Generating code coverage report, this may take a moment.

5: The coverage

Vector Tests

Current file: /home/kore/devel/personal/documents/textual/dwp_07/ttd/example/stage_05/Vector.php

Legend: executed not executed dead code

	Coverage								
	Classes			Methods			Lines		
Total	 	100.00%	1 / 1	 	100.00%	2 / 2	 	100.00%	9 / 9
Vector	 	100.00%	1 / 1	 	100.00%	2 / 2	 	100.00%	9 / 9
public function __construct(\$x = 0, \$y = 0)	 	100.00%	1 / 1	 	100.00%	1 / 1	 	100.00%	6 / 6
public function add(\$vector)	 	100.00%	1 / 1	 	100.00%	1 / 1	 	100.00%	3 / 3

```

1      : <?php
2      :
3      : class Vector
4      : {
5      :     public $x;
6      :
7      :     public $y;
8      :
9      :     public function __construct( $x = 0, $y = 0 )
10     :     {
11     3 :         if ( !is_numeric( $x ) || !is_numeric( $y ) )
12     3 :         {
13     1 :             throw new Exception( 'Only numeric values are allowed as parameters.' );
14     :         }
15     :
16     2 :         $this->x = $x;
17     2 :         $this->y = $y;
18     2 :     }
19     :
20     :     public function add( $vector )
21     :     {
22     4 :         $this->x += $vector->x;
23     4 :         $this->y += $vector->y;
24     :
25     4 :         return $this;
26     :     }
27     : }
28     :

```

5: Full code coverage

- But there are still missing edge cases!
 - Someone?
- Conclusion:
 - Incomplete coverage = Untested code
 - Complete coverage \neq Fully tested code

6: PHPUnit can handle non fatal errors

```
■ /**
 * @group error
 */
public function testInvalidVectorSum()
{
    $vector = new Vector();

    try
    {
        $vector->add( 2 );
    }
    catch ( PHPUnit_Framework_Error $e )
    {
        $this->assertSame(
            strpos( $e->getMessage(), 'Argument 1 passed to
Vector::add() must be an instance of Vector, integer given' ),
            0,
            'Expected error message: "Argument 1 passed to
Vector::add() must be an instance of Vector, integer given, ..."'
        );
    }
}
```

6: Run test from one group

- `$ phpunit --group error VectorTest`
PHPUnit 3.2.0 by Sebastian Bergmann.

..

Time: 0 seconds

OK (2 tests)



More essential features

- `setUp()` and `tearDown()`
 - Handle information / resources across test cases
- Lots of other different assertions than `assertSame()`
 - `assertEquals()`: Type does not matter, optional delta value
 - May also take `DOMDocuments` as parameters
 - `assertAttributeEquals()`: Check for (private) Object attributes
 - `assertRegExp()`, `assertType()`, `assertTrue()`, ...

Agenda

- Why you should (unit)test
- Writing tests with PHPUnit
- MockObjects
- Selenium
- Continuous Integration



•Use of Mock Objects

- Test object interaction
- Mock unstable, unreliable, untestable, slow, messy backends
 - Stubs for data from slow databases
 - Spys for calls to backends generating binary data

•The basics

- Dummy
 - A random variable
- Fake
 - Faking the expected object (inheritance)
- Stub
 - Provide canned data as return value
- Spy
 - Log requests to object
- Mock
 - Combination of Spy & Stub

Code examples – Stub

```
public function testStub()
{
    // Create, receive and assigne mocked class
    $caller = new exampleCaller();
    $listener = $this->getMock( 'exampleListener', array(
        'call',
        'update',
    ) );
    $caller->setListener( $listener );

    // Set up stubs
    $listener
        ->expects( $this->any() )
        ->method( 'call' )
        ->will( $this->returnValue( true ) );

    $listener
        ->expects( $this->any() )
        ->method( 'update' )
        ->will( $this->returnValue( false ) );

    // Run tested classes, using the stubs
    $caller->run();
}
```

•Code examples – Spy

```
public function testSpy()
{
    // Create, receive and assigne mocked class
    $caller = new exampleCaller();
    $listener = $this->getMock( 'exampleListener', array(
        'call',
    ) );
    $caller->setListener( $listener );

    // Set up expectations
    $listener
        ->expects( $this->once() )
        ->method( 'call' )
        ->with(
            $this->equalTo( 'signal' ),
            $this->equalTo( array( 1, 2, 3 ) )
        );

    // Run tested classes, to match expectations
    $caller->run();
}
```

•Code examples – Mock

```
public function testMock()
{
    // Create, receive and assigne mocked class
    $caller = new exampleCaller();
    $listener = $this->getMock( 'exampleListener', array(
        'call',
        'update',
    ) );
    $caller->setListener( $listener );

    // Set up expectations and return values
    $listener
        ->expects( $this->at( 0 ) )
        ->method( 'call' )
        ->with(
            $this->equalTo( 'signal' ),
            $this->equalTo( array( 1, 2, 3 ) )
        )
        ->will( $this->returnValue( true ) );

    $listener
        ->expects( $this->at( 1 ) )
        ->method( 'update' )
        ->will( $this->returnValue( false ) );

    // Run tested classes
    $caller->run();
}
```

- Not possible yet

- Mock final methods
- Mock internally used classes
 - Patch for overloading new available

•Agenda

- Why you should (unit)test
- Writing tests with PHPUnit
- MockObjects
- Selenium
- Continuous Integration



•Selenium (1)

- Test web applications
 - Regular expressions on the result on a local host?
 - Fails for "rich interfaces"
 - Does not respect browser incompatibilities
- We need to control the browsers somehow
 - They are all so f***ing different

•Selenium (2)

- Works around all browser and OS quirks
 - Windows: Internet Explorer 6.0 and 7.0, Firefox 1.5.0.8 and 2.0, Opera 8.5.4 and 9.0.2 (Firefox 0.8 to 2.0, Mozilla Suite 1.6+, 1.7+, Seamonkey 1.0, Opera 8.5+, 9)
 - Mac OS X: Firefox 1.5.0.4 and 2.0 (Safari 1.3+, Firefox 0.8 to 2.0, Camino 1.0a1, Mozilla Suite 1.6+, 1.7+, Seamonkey 1.0)
 - Linux: Firefox 1.5.0.8 and 2.0, Opera 9.0.2, Konqueror 3.5.3 (Firefox 0.8 to 2.0, Mozilla Suite 1.6+, 1.7+, Konqueror 3.5+, Opera 8.5+, 9)

•Selenium (3)

- Test web applications directly in the browser
 - System tests
 - Browser compability testing, by using different browsers
- Full usage of all browser features
 - ECMAScript, XMLHttpRequest, (AJAX), ...

•Selenium IDE

- IDE for Selenium tests
 - Firefox Extension
- Work with test in your browser
 - Record
 - Execute
 - Edit
 - Debug

•Selenium RC

- Automated execution of Selenium tests
- Test may be specified in any language
 - PHP Bindings: PEAR:PHP_Selenium
 - PHPUnit: PHPUnit_Extensions_SeleniumTestCase
- One test may be executed in multiple browser / OS combinations
 - Use and control virtual machines

•Agenda

- Why you should (unit)test
- Writing tests with PHPUnit
- MockObjects
- Selenium
- Continuous Integration

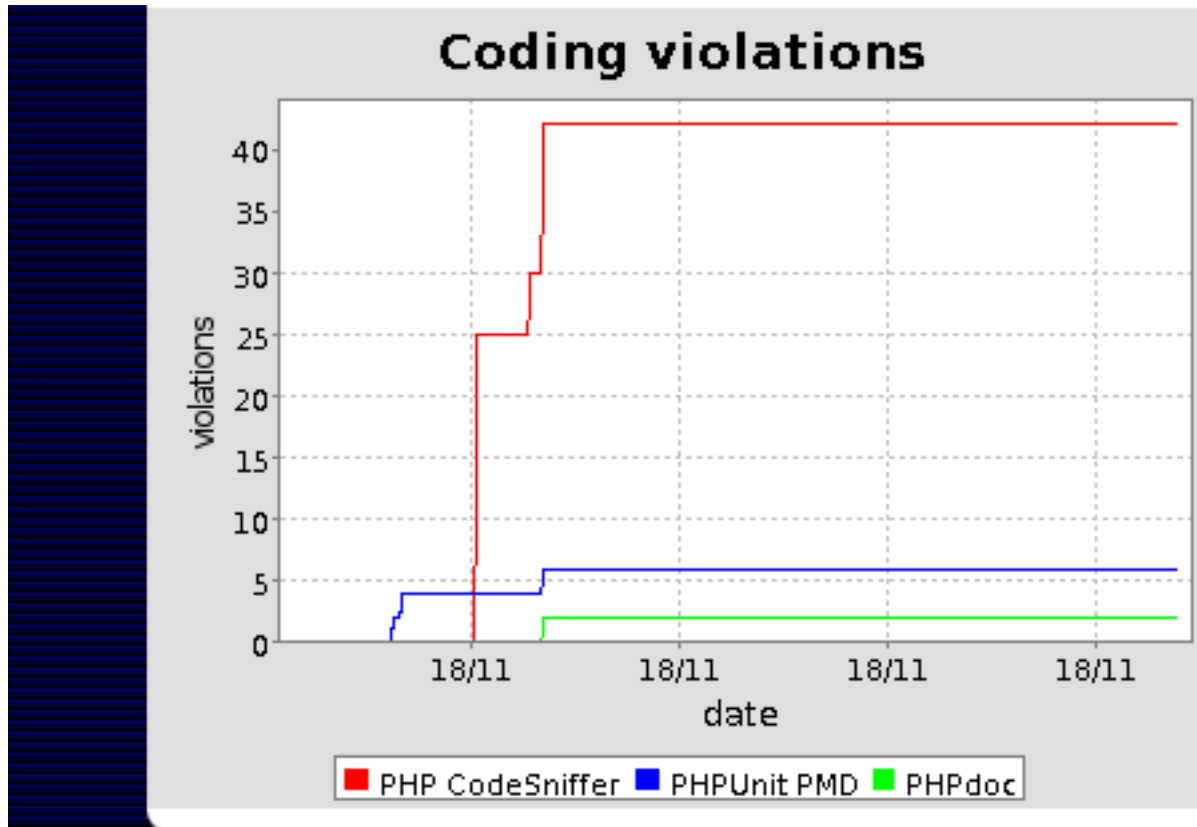
•Continuous Integration

- Continuously runs your tests
 - After each commit to your VCS (version control system)
 - After a configured timespan
- Shows broken builds
 - Sends mails to your project manager ;)
 - Makes conflicting changes easily visible

•CruiseControl

- Java based framework for continuous integration
 - New BSD licensed
- Integration with PHP tools
 - phpUnderControl by Manuel Pichler
 - Integrates PHPUnit, CodeSniffer & CruiseControl

- phpUnderControl – Screenshot (1)



•phpUnderControl – Screenshot (2)

11/18/2007 19:48:23 (build.247)
11/18/2007 19:47:19 (build.246)
11/18/2007 19:46:15 (build.245)
11/18/2007 19:45:11 (build.244)
11/18/2007 19:44:08 (build.243)
11/18/2007 19:43:04 (build.242)
11/18/2007 19:42:01 (build.241)
11/18/2007 19:40:57 (build.240)
11/18/2007 19:39:53 (build.239)
11/18/2007 19:38:50 (build.238)

More builds

RSS

72 The CRAP index is 132. The Change Risk Analysis and Predictions (CRAP) index of a function or method uses cyclomatic complexity and code coverage from automated tests to help estimate the effort and risk associated with maintaining legacy code. A CRAP index over 30 is a good indicator of crappy code.

72 The NPath complexity is 83520. The NPath complexity of a function or method is the number of acyclic execution paths through that method. A threshold of 200 is generally considered the point where measures should be taken to reduce complexity.

72 The code coverage is 0.00 which is considered low.

136 The CRAP index is 132. The Change Risk Analysis and Predictions (CRAP) index of a function or method uses cyclomatic complexity and code coverage from automated tests to help estimate the effort and risk associated with maintaining legacy code. A CRAP index over 30 is a good indicator of crappy code.

136 The NPath complexity is 83520. The NPath complexity of a function or method is the number of acyclic execution paths through that method. A threshold of 200 is generally considered the point where measures should be taken to reduce complexity.

136 The code coverage is 0.00 which is considered low.

Duplication (Files: 2, Lines: 28, Tokens: 68)

72 /opt/cruisecontrol-bin-2.7/projects/php-under-control/source/src/Math.php

136 /opt/cruisecontrol-bin-2.7/projects/php-under-control/source/src/Math.php

```
public function div($v1, $v2)
{
    $v3 = $v1 / ($v2 + $v1);
    if ($v3 > 14)
    {
        $v4 = 0;
        for ($i = 0; $i < $v3; $i++)
        {
            $v4 += ($v2 * $i);
        }
    }
    $v5 = ($v4 < $v3 ? ($v3 - $v4) : ($v4 - $v3));
    $v6 = ($v1 * $v2 * $v3 * $v4 * $v5);
    $d = array($v1, $v2, $v3, $v4, $v5, $v6);
    $v7 = 1;
    for ($i = 0; $i < $v6; $i++)
    {
        shuffle( $d );
        $v7 = $v7 + $i * end($d);
    }
    $v8 = $v7;
    foreach ( $d as $x )
    {
        $v8 *= $x;
    }
}
```

•phpUnderControl – Screenshot (3)

cruisecontrol
continuous integration toolkit

Project
php-under-control

waiting for next time to build since
11/18/2007 19:07:35

Latest Build
 11/18/2007 19:06:47 (build.209)
 11/18/2007 19:05:43 (build.208)
 11/18/2007 19:04:38 (build.207)
 11/18/2007 19:03:23 (build.206)
 11/18/2007 19:02:20 (build.205)
 11/18/2007 19:01:16 (build.204)
 11/18/2007 19:00:10 (build.203)
 11/18/2007 18:59:06 (build.202)
 11/18/2007 18:58:02 (build.201)
 11/18/2007 18:56:58 (build.200)
 More builds

Build Results	Test Results	XML Log File	Metrics	PHP CodeSniffer	PHPUnit PMD
BUILD COMPLETE - build.209					
Date of build:		11/18/2007 19:06:47			
Time to build:		2 seconds			
Last changed:		11/18/2007 19:05:43			
Last log entry:					
Build Artifacts					
PHPUnit PMD errors/warnings (0 / 7)					
7 warnings					
Errors/Warnings: (1)					
Result: 1					
PHP CodeSniffer errors/warnings (40 / 2)					
/opt/cruisecontrol-bin-2.7/projects/php-under-control/source/tests/MathTest.php		18	PHP version not specified		
/opt/cruisecontrol-bin-2.7/projects/php-under-control/source/tests/MathTest.php		18	Missing @category tag in file comment		
/opt/cruisecontrol-bin-2.7/projects/php-under-control/source/tests/MathTest.php		18	Missing @package tag in file comment		
/opt/cruisecontrol-bin-2.7/projects/php-under-control/source/tests/MathTest.php		18	Missing @author tag in file comment		
/opt/cruisecontrol-bin-2.7/projects/php-under-control/source/tests/MathTest.php		18	Missing @license tag in file comment		
/opt/cruisecontrol-bin-2.7/projects/php-under-control/source/tests/MathTest.php		18	Missing @link tag in file comment		
/opt/cruisecontrol-bin-2.7/projects/php-under-control/source/tests/MathTest.php		47	Missing @return tag in function comment		
/opt/cruisecontrol-bin-2.7/projects/php-under-control/source/tests/MathTest.php		57	Missing @return tag in function comment		
/opt/cruisecontrol-bin-2.7/projects/php-under-control/source/tests/MathTest.php		65	Missing @return tag in function comment		
phpdoc errors/warnings: (0 / 2)					
WARNING in MathTest.php on line 69: File "/opt/cruisecontrol-bin-2.7/projects/php-under-control/source/tests/MathTest.php" has no page-level DocBlock, use @package in the first DocBlock to create one					
WARNING: Class PhpUnderControl_Example_MathTest parent PHPUnit_Framework_TestCase not found					
Unit Tests: (2)					
All Tests Passed					
Modifications since last successful build: (1)					
change	User	force build/force build		11/18/2007 19:05:43	

•Questions?

- Open questions?
- Ressources:
 - <http://phpunit.de>
 - <http://kore-nordmann.de>
 - xUnit Test Patterns (ISBN: 978-0131495050)
- Contact
 - kore@php.net
 - mail@kore-nordmann.de

The end

- Thanks for listening

