

php|works - Atlanta, US  
Derick Rethans - dr@ez.no  
<http://derickrethans.nl/talks.php>

- Dutchman living in Norway
- eZ Systems A.S.
- eZ Components project lead
- PHP development
- mcrypt, input\_filter, date/time support, unicode
- QA

# It's OK to write code that does not work



# About Testing

### Front-end

- Acceptance Tests and System Tests that run in the browser
- Testing of Web Services with Unit Tests
- Compatibility Testing for Browser/OS/etc. combinations
- Performance Testing
- Security Testing

### Back-end

- Functional Testing of business logic with Unit Tests
- Reusable Components, but they often come with their own tests

## Unit Testing

Tests small parts of an application or library (units) for correctly working code. Tools: PHPUnit, SimpleTest

## System Testing

The testing of a whole integrated system against the specified requirements. Tools: Selenium

## Non-functional Testing

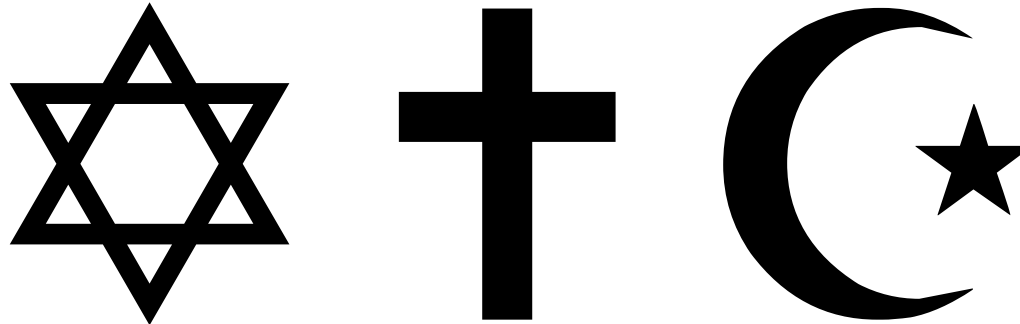
Testing for performance, load, stress, reliability, availability, security. Tools: ab, siege, httpperf, chorizo

# Test-Driven Development

a feature without a test is not a feature

**It is not just a method of testing  
software.**

**It is a religion.**



### Requirements Specification

Define what the software is supposed to do.

### Design

Define how the software is supposed to be implemented.

### Implementation

The implementation of the software itself.

### Testing

The implemented software is tested.

Sometimes.



# TESTING

I FIND YOUR LACK OF TESTS DISTURBING.

## Tests drive the development

- Tests are written before the code
- There is no code without tests

## Test Suites

- Contain tests that check whether the code does what it is supposed to do
- Also cover things that should fail

### Requirements Specification

Define what the software is supposed to do.

### Design

Define how the software is supposed to be implemented.

### Implementation $\equiv$ Testing

The implemented software is tested.

The implementation of the software itself.

**Present the Idea**

**Write the Requirements Document**

**Design the Component**

**Implementation**

- Write API stubs with parameter documentation and descriptions
- Write test cases
- Initial implementation
- Initial implementation review
- Updating implementation according to review
- Implementation review

**Pre-release Testing**

### Write Test Case

Write a test case to test the correct behavior of the method, and verify that the test case fails

### Fix the Issue

Fix the implementation

### Verify Fix with Test Case

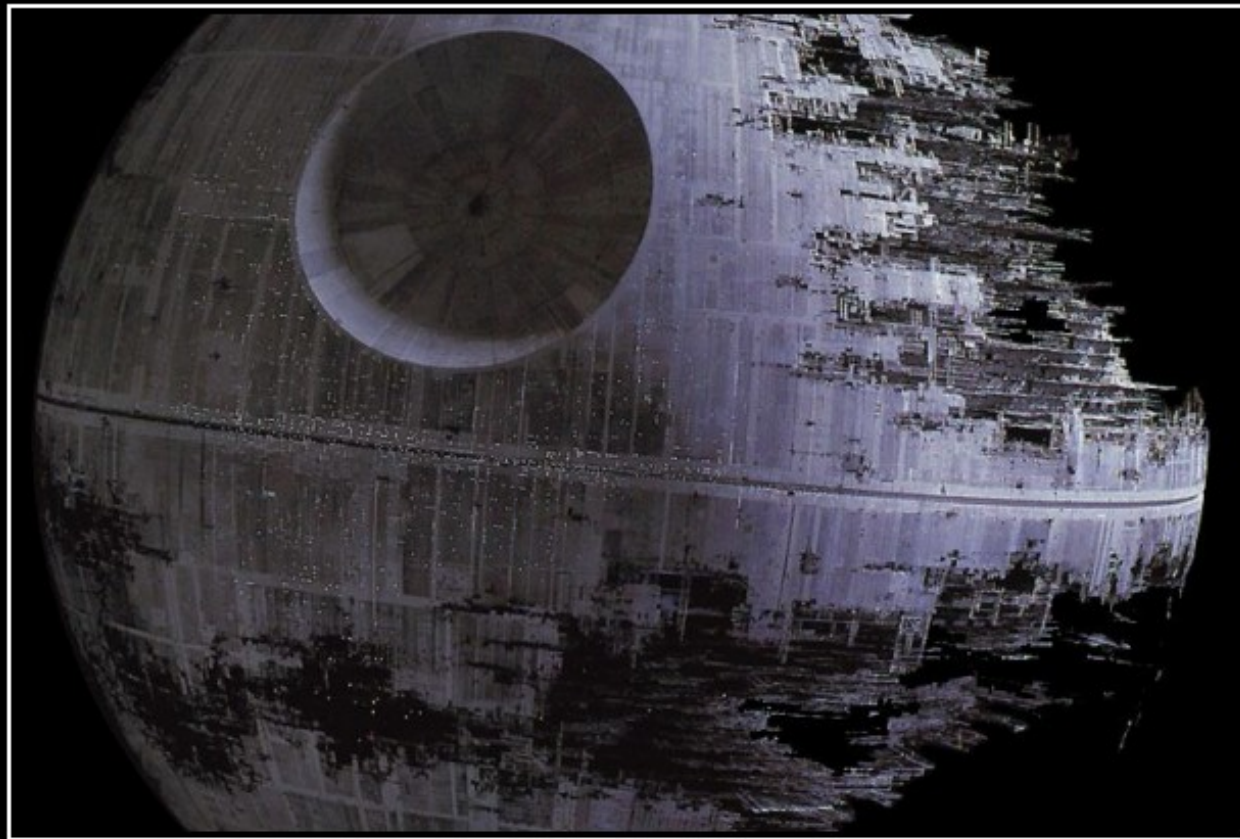
Make sure that the test cases pass with the new implementation

### Check Other Test Cases

Verify that the fix for this defect did not break any other test case

- If you write code, write tests.
- Don't get stuck on unit testing dogma.
- Embrace unit testing karma.
- Think of code and test as one.
- The test is more important than the unit.
- The best time to test is when the code is fresh.
- Tests not run waste away.
- An imperfect test today is better than a perfect test someday.
- An ugly test is better than no test.
- Sometimes, the test justifies the means.
- Only fools use no tools.
- Good tests fail.

<http://www.artima.com/weblogs/viewpost.jsp?thread=203994>



# TESTING

DON'T BE TOO PROUD OF THIS TECHNOLOGICAL TERROR YOU'VE  
CONSTRUCTED. THE ABILITY TO DESTROY A PLANET IS INSIGNIFICANT NEXT  
TO THE POWER OF TESTING.

# PHP Unit

# PHPUnit

## Unit Testing Framework for PHP

```
derick@kossu: ~/dev/ezcomponents/trunk
derick@kossu:~/dev/ezcomponents/trunk$ php UnitTest/src/runtests.php Configuration
ezcUnitTest uses the PHPUnit 3.3.0RC1 framework from Sebastian Bergmann.

[Preparing tests]:
eZ Components:
  Configuration:
    ezcConfigurationManagerDelayedInitTest: .
    ezcConfigurationTest: .....
    .....
    ezcConfigurationManagerTest: .....
    ezcConfigurationIniReaderTest: .....
    ezcConfigurationIniParserTest: ...
    ezcConfigurationIniWriterTest: .....
    ezcConfigurationArrayWriterTest: .....

Time: 0 seconds

OK (161 tests, 252 assertions)
derick@kossu:~/dev/ezcomponents/trunk$
```


































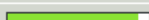
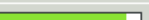
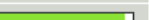















# PHPUnit

## Code Coverage: No Code Without Test

### eZ Components

Current directory: [/home/derick/dev/ezcomponents/trunk/Graph/src](#)

Legend: Low: 0% to 35% Medium: 35% to 70% High: 70% to 100%

	Coverage					
	Classes		Methods		Lines	
Total		<b>94.39%</b> 101 / 107		<b>86.89%</b> 444 / 511		<b>86.32%</b> 8990 / 10415
<u>axis</u>		<b>100.00%</b> 5 / 5		<b>85.25%</b> 52 / 61		<b>91.02%</b> 669 / 735
<u>charts</u>		<b>100.00%</b> 5 / 5		<b>100.00%</b> 36 / 36		<b>99.56%</b> 683 / 686
<u>colors</u>		<b>100.00%</b> 3 / 3		<b>100.00%</b> 17 / 17		<b>100.00%</b> 216 / 216
<u>data_container</u>		<b>100.00%</b> 2 / 2		<b>100.00%</b> 13 / 13		<b>100.00%</b> 39 / 39
<u>datasets</u>		<b>100.00%</b> 9 / 9		<b>89.80%</b> 44 / 49		<b>93.63%</b> 235 / 251
<u>driver</u>		<b>50.00%</b> 3 / 6		<b>49.47%</b> 47 / 95		<b>54.97%</b> 1339 / 2436
<u>element</u>		<b>100.00%</b> 4 / 4		<b>100.00%</b> 21 / 21		<b>99.35%</b> 458 / 461
<u>exceptions</u>		<b>96.15%</b> 25 / 26		<b>96.00%</b> 24 / 25		<b>95.50%</b> 106 / 111
<u>interfaces</u>		<b>100.00%</b> 7 / 7		<b>97.73%</b> 43 / 44		<b>93.74%</b> 943 / 1006
<u>math</u>		<b>100.00%</b> 7 / 7		<b>100.00%</b> 43 / 43		<b>99.72%</b> 350 / 351
<u>options</u>		<b>92.86%</b> 13 / 14		<b>90.32%</b> 28 / 31		<b>94.11%</b> 815 / 866
<u>palette</u>		<b>83.33%</b> 5 / 6		<b>100.00%</b> 0 / 0		<b>83.33%</b> 5 / 6
<u>renderer</u>		<b>100.00%</b> 8 / 8		<b>100.00%</b> 67 / 67		<b>96.19%</b> 3005 / 3124
<u>structs</u>		<b>100.00%</b> 3 / 3		<b>100.00%</b> 7 / 7		<b>100.00%</b> 35 / 35
<u>graph.php</u>		<b>100.00%</b> 1 / 1		<b>100.00%</b> 0 / 0		<b>100.00%</b> 1 / 1
<u>tools.php</u>		<b>100.00%</b> 1 / 1		<b>100.00%</b> 2 / 2		<b>100.00%</b> 91 / 91

Generated by [PHPUnit 3.3.0RC1](#) and [Xdebug 2.1.0-dev](#) at Mon Sep 8 10:30:06 CEST 2008.

```
550 :  
551 19 :      switch ( $this->interval )  
552 :      {  
553 19 :      case self::MONTH:  
554 4 :          $time = strtotime( '+1 month', $time );  
555 4 :          break;  
556 15 :      case self::YEAR:  
557 4 :          $time = strtotime( '+1 year', $time );  
558 4 :          break;  
559 11 :      case self::DECADE:  
560 0 :          $time = strtotime( '+10 years', $time );  
561 0 :          break;  
562 11 :      default:  
563 11 :          $time += $this->interval;  
564 11 :          break;  
565 19 :      }  
566 19 :      }  
567 :  
568 19 :      return $steps;  
569 :      }  
570 :  
571 :      /**
```

Green: Covered code.

Red: Not covered code.

Grey: Unreachable code.

White: No code.

**< 100% Coverage  $\equiv$  Not fully tested code**

**100% Coverage  $\neq$  Fully tested code**

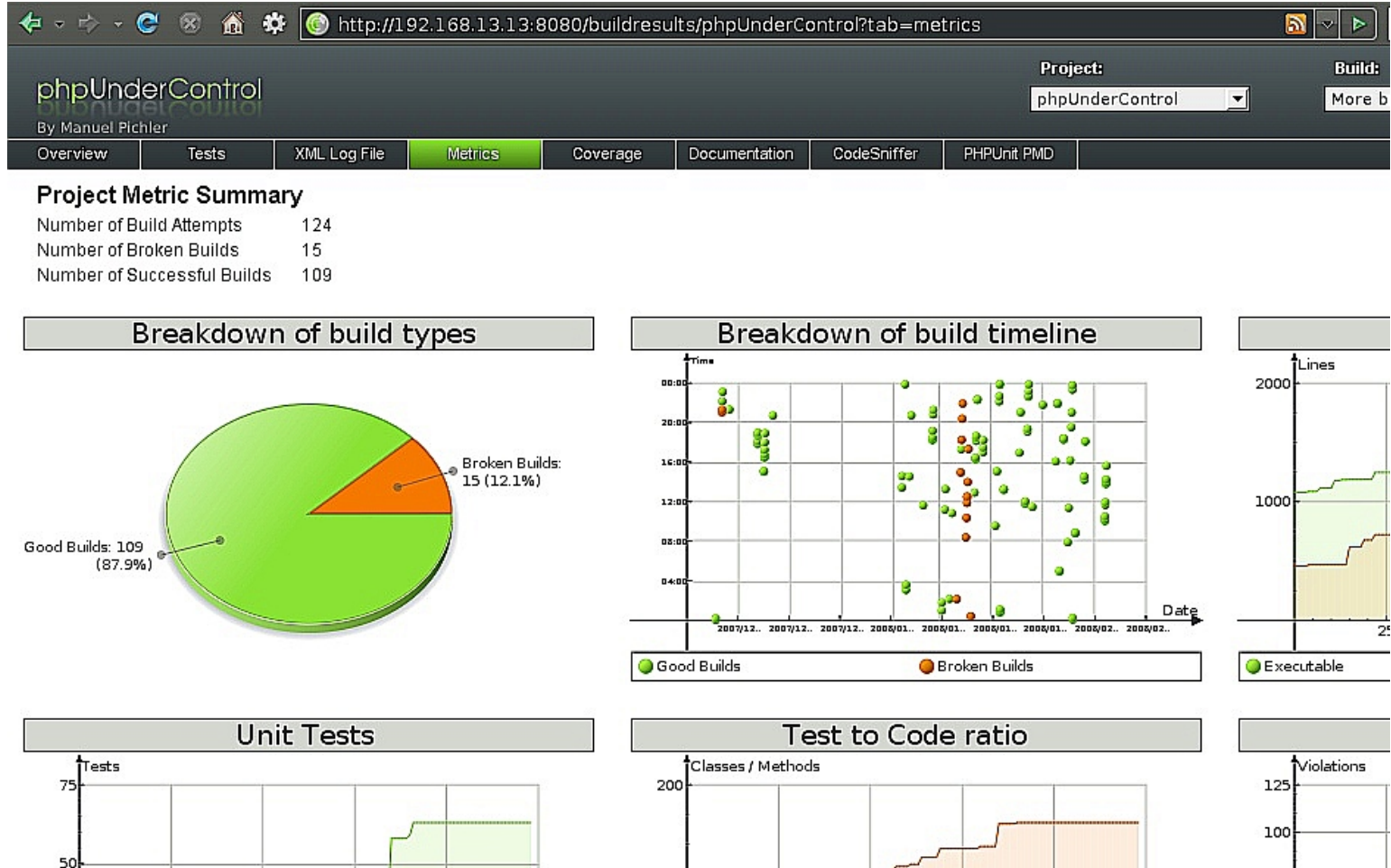
**How much coverage should you aim for?**

- It doesn't matter, just write some tests.
- How much rice do I need to cook?
- 100%, and no less!

<http://www.developertesting.com/archives/month200705/20070504-000425.html>

# phpUnderControl

## Continuous integration with CruiseControl



- Testing singletons more than one time
- System/Operating System dependent tests
- Private methods
- Code that depends on the state of an external resource
- Things that simply should never happen

# Politics

resistance to change

Fear - more work to do

- Introduce TDD concepts gently
- Whenever a problem is found, make and retain a test case for further use
- Start using TDD for new projects

Ignorance - too much time spent on testing

- Out of date with modern processes
- Belief that testing slows the schedule (only if you follow the ship-and-see process)
- You save time later, because you wouldn't have to re-test or re-debug newly written additions or big fixes.



**TESTS**

WHERE ARE YOURS ?!

# Case Studies

## 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

- At first I didn't like that I need to write tests for my code, but now after using it for more than 10 months I can't program without it.
- Helps to come up with better APIs.
- It gives confidence that our software is working well at all times. Even after making major changes and/or changing software we are dependent on.
- Productivity increases - you might lose some when you make the initial tests, but you'll get it back later. The code covered by tests is 'insured' against future changes.
- It works well for libraries, not so well for GUI applications.
- Some things cannot be tested like server errors, unless you use mock-objects.



<http://homepage.mac.com/hey.you/lessons.html>

Testuvius: <http://www.artima.com/weblogs/viewpost.jsp?thread=203994>

PHP Unit: <http://phpunit.de>

phpUnderControl: <http://phpundercontrol.org>

Xdebug: <http://xdebug.org>

These Slides: <http://derickrethans.nl/talks.php>

PHP: <http://www.php.net>