

open **YOUR** mind



## Quality Assurance in PHP Projects

Sebastian Bergmann

<http://sebastian-bergmann.de/>

July 21<sup>st</sup> 2008

# Who I am



- Sebastian Bergmann
- Involved in the PHP Project since 2000
- Developer of PHPUnit
- Author, Consultant, Coach, Trainer

# Quality Assurance in PHP Projects

## Schedule

- 08:30am – 10:00am Tutorial
- 10:00am – 10:30am Coffee Break
- 10:30am – 12:00pm Tutorial

# Quality Assurance in PHP Projects

## Topics

- PHPUnit
  - Writing and running tests
  - Organizing tests into suites and groups
  - Refactoring, Test-Driven and Behaviour-Driven Development
  - Code Coverage

# Quality Assurance in PHP Projects

## Topics

- Selenium
  - Recording and running tests with Selenium IDE
  - Integrating PHPUnit with Selenium RC
- Continuous Integration
  - phpUnderControl
  - PHP\_CodeSniffer

# “Hot Topics” in PHP's History

From a Keynote by Derick Rethans



- Make it work
- Make it fast
- Make it scale
- Make it secure

Now that we know how to build applications that "just work", are fast and scalable, as well as secure, the next logical step is to implement processes and use techniques that help us assure that the software works correctly throughout the software's lifecycle.

# Why test?

- Companies develop more and more enterprise-critical applications with PHP.
- Tests help to make sure that these applications work correctly.

# What needs testing?

## Web Application

### – Backend

- Business Logic
- Reusable Components

### – Frontend

- Form Processing, Templates, ...
- “Rich Interfaces” with AJAX, JSON, ...
- Feeds, Web Services, ...

# And how do we test it?

## Web Application

### – Backend

- Functional Testing of the business logic with Unit Tests
- Reusable Components
  - External components should have their own unit tests.

### – Frontend

- Acceptance Tests or System Tests that are run “in the browser”
- Testing of feeds, web services, etc. with unit tests
- Compatibility Tests for Operating System / Browser combinations
- Performance Tests and Security Tests

# Testing Software

- Component Tests
  - Executable code fragments, so called *Unit Tests*, test the correctness of parts, *units*, of the software (*system under test*)
- System Tests
  - Conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. (Wikipedia)
- Non-Functional Tests
  - Performance, Stability, Security, ...

# Testing Software

- Developer Tests
  - Ensure that the code works correctly
- Acceptance Tests
  - Ensure that the code does what the customer wants

# Test Tools

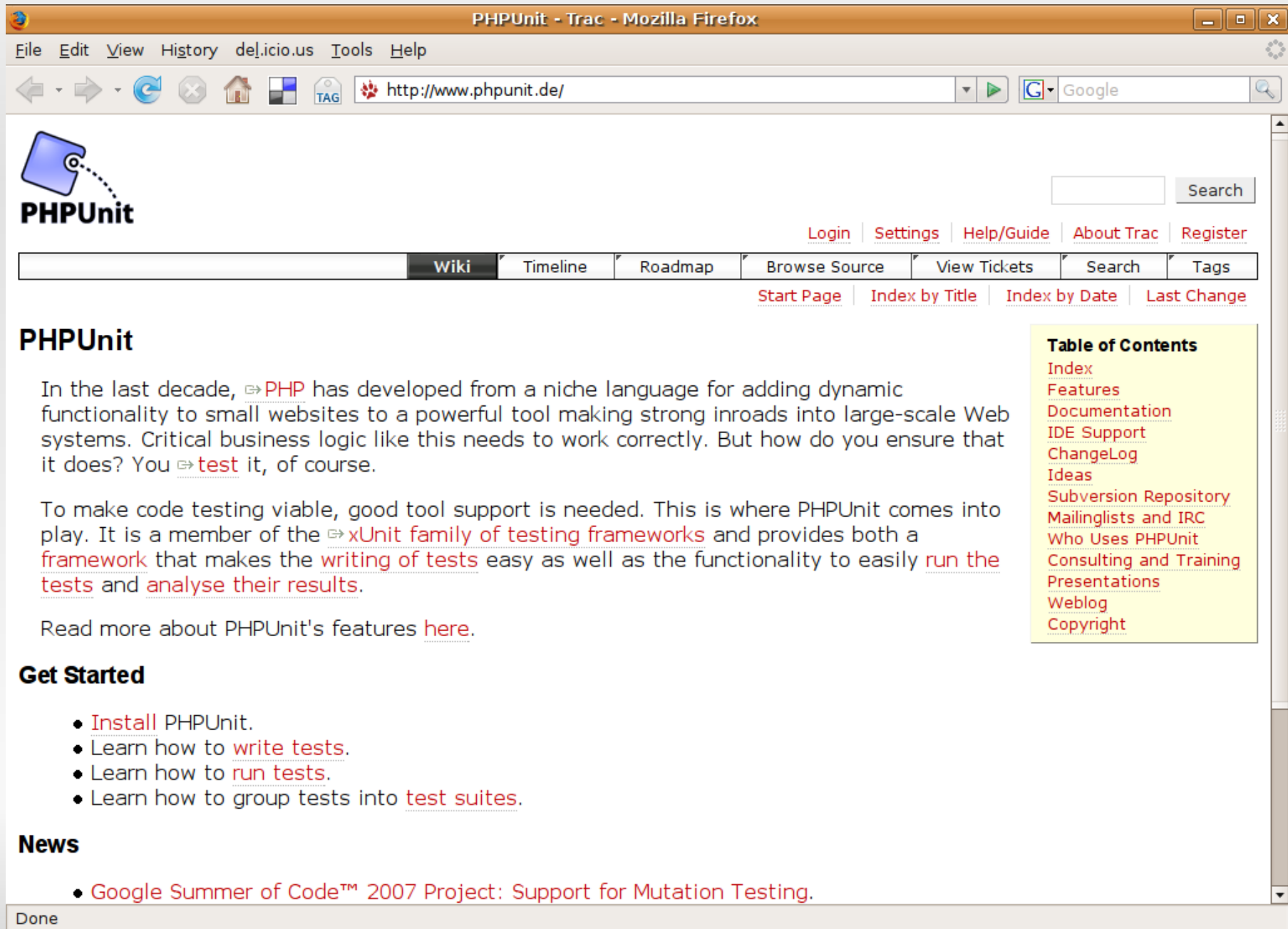
- To make (code) testing viable, good tool support is needed.
- This is where a testing framework such as PHPUnit comes into play.
- Requirements
  - Reusable test environment
  - Strict separation of production code and test code
  - Automatic execution of test code
  - Analysis of the result
  - Easy to learn to use and easy to use

# Test Tools for PHP / Web

- Unit Tests
  - PHPUnit
  - PHPT
  - SimpleTest
- System Tests
  - Selenium
    - PHPUnit + Selenium RC
- Non-Functional Tests
  - Performance, Load, Stress, Availability, ...
    - ab, httpperf, JMeter, Grinder, OpenSTA, ...
  - Security
    - Chorizo

# PHPUnit: Website

<http://www.phpunit.de/>




The screenshot shows a Mozilla Firefox browser window displaying the PHPUnit website. The browser's address bar shows the URL <http://www.phpunit.de/>. The website's header includes the PHPUnit logo, a search bar, and navigation links for [Login](#), [Settings](#), [Help/Guide](#), [About Trac](#), and [Register](#). Below the header is a navigation menu with links for [Wiki](#), [Timeline](#), [Roadmap](#), [Browse Source](#), [View Tickets](#), [Search](#), and [Tags](#). The main content area features a large heading for "PHPUnit" followed by an introductory paragraph. A "Table of Contents" sidebar is visible on the right, listing various sections like [Index](#), [Features](#), [Documentation](#), [IDE Support](#), [ChangeLog](#), [Ideas](#), [Subversion Repository](#), [Mailinglists and IRC](#), [Who Uses PHPUnit](#), [Consulting and Training](#), [Presentations](#), [Weblog](#), and [Copyright](#). Below the main text, there are sections for "Get Started" with a list of links and a "News" section with a single link.

PHPUnit - Trac - Mozilla Firefox

File Edit View History del.icio.us Tools Help

[http://www.phpunit.de/](#) Google

  Search

[Login](#) | [Settings](#) | [Help/Guide](#) | [About Trac](#) | [Register](#)

[Wiki](#) | [Timeline](#) | [Roadmap](#) | [Browse Source](#) | [View Tickets](#) | [Search](#) | [Tags](#)

[Start Page](#) | [Index by Title](#) | [Index by Date](#) | [Last Change](#)

## PHPUnit

In the last decade, [PHP](#) has developed from a niche language for adding dynamic functionality to small websites to a powerful tool making strong inroads into large-scale Web systems. Critical business logic like this needs to work correctly. But how do you ensure that it does? You [test](#) it, of course.

To make code testing viable, good tool support is needed. This is where PHPUnit comes into play. It is a member of the [xUnit family of testing frameworks](#) and provides both a [framework](#) that makes the [writing of tests](#) easy as well as the functionality to easily [run the tests](#) and [analyse their results](#).

Read more about PHPUnit's features [here](#).

### Get Started

- [Install](#) PHPUnit.
- Learn how to [write tests](#).
- Learn how to [run tests](#).
- Learn how to group tests into [test suites](#).

### News

- [Google Summer of Code™ 2007 Project: Support for Mutation Testing](#).

Done

# PHPUnit: Installation

```
sb@vmware ~ % pear channel-discover pear.phpunit.de
Adding Channel "pear.phpunit.de" succeeded
Discovery of channel "pear.phpunit.de" succeeded

sb@vmware ~ % pear install phpunit/PHPUnit
downloading PHPUnit-3.2.11.tgz ...
Starting to download PHPUnit-3.2.11.tgz (201,578 bytes)
.....done: 201,578 bytes
install ok: channel://pear.phpunit.de/PHPUnit-3.2.11
```

# Outline

- PHPUnit
  - Writing tests
  - Running tests
- Selenium
  - Recording tests with the Selenium IDE
  - Running tests with PHPUnit and Selenium RC
- phpUnderControl
  - Continuous Integration
  - Software Metrics

# The Bowling Game Kata

## Introduction: Scoring Bowling

The game consists of 10 frames as shown below.

- In each frame the player has two opportunities to knock down 10 pins.
- The score for the frame is the total number of pins knocked down, plus bonuses for strikes and spares.

1	4	4	5	6	▲	5	▲	0	1	7	▲	6	▲	2	▲	6
5	14	29	49	60	61	77	97	117	133							







# The Bowling Game Kata

## Introduction: Requirements

- Write a class named `BowlingGame` that has two methods
  - `roll($pins)` is called each time the player rolls a ball.
    - The argument is the number of pins knocked down.
  - `score()` is called only at the very end of the game and returns the total score for that game.

# The Bowling Game Kata

## Introduction: Design

- A game has 10 frames.
  - A frame has 1 or 2 rolls.
  - The 10<sup>th</sup> frame has 2 or 3 rolls and is different from all the other frames.
- The `score()` method must iterate over all the frames and calculate all their scores.
  - The score for a spare or a strike depends on the frame's successor.

# The Bowling Game Kata

## The first test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{

}
}
```

# The Bowling Game Kata

## The first test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    public function testGutterGame()
    {

    }
}
```

# The Bowling Game Kata

## The first test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    public function testGutterGame()
    {
        $game = new BowlingGame;

        for ($i = 0; $i < 20; $i++) {
            $game->roll(0);
        }

        $this->assertEquals(0, $game->score());
    }
}
```

# The Bowling Game Kata

## The first test

```
<?php
class BowlingGame
{
    public function roll($pins)
    {
    }

    public function score()
    {
        return 0;
    }
}
```

# The Bowling Game Kata

## The first test

```
sb@vmware ~ % phpunit BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
.
```

```
Time: 0 seconds
```

```
OK (1 test, 1 assertions)
```

# The Bowling Game Kata

## The first test

```
sb@vmware ~ % phpunit --ansi BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
.
```

```
Time: 0 seconds
```

```
OK (1 test, 1 assertions)
```

# Interlude

## Test-Driven Development

- Method of designing software, not just a method of testing software
  - Test
    - What do we want  $X$  to do?
    - How do we want to tell  $X$  to do it?
    - How will we know when  $X$  has done it?
  - Code
    - How does  $X$  do it?
- Tests drive the development
  - Tests written before code
  - No code without tests

# The Bowling Game Kata

## The first test

```
sb@vmware ~ % phpunit --skeleton-class BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

Wrote skeleton for "BowlingGame" to "BowlingGame.php".

```
<?php  
class BowlingGame  
{  
    /**  
     * @todo Implement roll().  
     */  
    public function roll()  
    {  
        // Remove the following line when you implement this method.  
        throw new RuntimeException('Not yet implemented.');    }  
  
    /**  
     * @todo Implement score().  
     */  
    public function score()  
    {  
        // Remove the following line when you implement this method.  
        throw new RuntimeException('Not yet implemented.');    }  
}  
?>
```

# The Bowling Game Kata

## The second test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testAllOnes()
    {
        $game = new BowlingGame;

        for ($i = 0; $i < 20; $i++) {
            $game->roll(1);
        }

        $this->assertEquals(20, $game->score());
    }
}
```

# The Bowling Game Kata

## The second test

```
sb@vmware ~ % phpunit BowlingGameTest
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
.F
```

```
Time: 0 seconds
```

```
There was 1 failure:
```

```
1) testAllOnes(BowlingGameTest)
```

```
Failed asserting that <integer:0> matches expected value <integer:20>.
/home/sb/BowlingGameTest.php:25
```

```
FAILURES!
```

```
Tests: 2, Assertions: 2, Failures: 1.
```

# The Bowling Game Kata

## The second test

```
sb@vmware ~ % phpunit --ansi BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
.F
```

```
Time: 0 seconds
```

```
There was 1 failure:
```

```
1) testAllOnes(BowlingGameTest)  
Failed asserting that <integer:0> matches expected value <integer:20>.  
/home/sb/BowlingGameTest.php:25
```

```
FAILURES!
```

```
Tests: 2, Assertions: 2, Failures: 1.
```

# The Bowling Game Kata

## The second test

```
<?php
class BowlingGame
{
    protected $score = 0;

    public function roll($pins)
    {
        $this->score += $pins;
    }

    public function score()
    {
        return $this->score;
    }
}
```

# The Bowling Game Kata

## The second test

```
sb@vmware ~ % phpunit BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
..
```

```
Time: 0 seconds
```

```
OK (2 tests, 2 assertions)
```

# The Bowling Game Kata

## The second test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testAllOnes()
    {
        $game = new BowlingGame;

        for ($i = 0; $i < 20; $i++) {
            $game->roll(1);
        }

        $this->assertEquals(20, $game->score());
    }
}
```

# The Bowling Game Kata

## The second test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testAllOnes()
    {
        $game = new BowlingGame;

        for ($i = 0; $i < 20; $i++) {
            $game->roll(1);
        }

        $this->assertEquals(20, $game->score());
    }
}
```

# Interlude

## Test Fixture

- One of the most time-consuming parts of writing tests is writing the code to set the world up in a known state and then return it to its original state when the test is complete.
- This known state is called the *fixture* of the test.

# Interlude

## Test Fixture

- In our BowlingGameTest example the test fixture was a single object.
- Most of the time, though, the fixture will be more complex than a simple object, and the amount of code needed to set it up will grow accordingly.
- The actual content of the test gets lost in the noise of setting up the fixture.
  - This problem gets even worse when you write several tests with similar fixtures.
- PHPUnit supports sharing the setup code through the setUp() and tearDown() template methods.

# The Bowling Game Kata

## The second test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    protected $game;

    protected function setUp()
    {
        $this->game = new BowlingGame;
    }

    // ...

    public function testAllOnes()
    {
        for ($i = 0; $i < 20; $i++) {
            $this->game->roll(1);
        }

        $this->assertEquals(20, $this->game->score());
    }

    // ...
}
```

# The Bowling Game Kata

## The second test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    protected $game;

    protected function setUp()
    {
        $this->game = new BowlingGame;
    }

    // ...

    public function testAllOnes()
    {
        for ($i = 0; $i < 20; $i++) {
            $this->game->roll(1);
        }

        $this->assertEquals(20, $this->game->score());
    }

    // ...
}
```

# The Bowling Game Kata

## The second test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    // ...

    protected function rollMany($n, $pins)
    {
        for ($i = 0; $i < $n; $i++) {
            $this->game->roll($pins);
        }
    }

    public function testGutterGame()
    {
        $this->rollMany(20, 0);
        $this->assertEquals(0, $this->game->score());
    }

    public function testAllOnes()
    {
        $this->rollMany(20, 1);
        $this->assertEquals(20, $this->game->score());
    }
}
```

# The Bowling Game Kata

## The second test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    // ...

    protected function rollMany($n, $pins)
    {
        for ($i = 0; $i < $n; $i++) {
            $this->game->roll($pins);
        }
    }

    public function testGutterGame()
    {
        $this->rollMany(20, 0);
        $this->assertEquals(0, $this->game->score());
    }

    public function testAllOnes()
    {
        $this->rollMany(20, 1);
        $this->assertEquals(20, $this->game->score());
    }
}
```

# Interlude

## Refactoring

A code refactoring is any change to a computer program's code which improves its readability or simplifies its structure without changing its results. (Wikipedia)

1. All unit tests run correctly.
2. The code communicates its design principles.
3. The code contains no redundancies.
4. The code contains the minimal number of classes and methods.

# The Bowling Game Kata

## The third test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testOneSpare()
    {
        $this->game->roll(5);
        $this->game->roll(5); // Spare
        $this->game->roll(3);
        $this->rollMany(17, 0);
        $this->assertEquals(16, $this->game->score());
    }
}
```

# The Bowling Game Kata

## The third test

```
sb@vmware ~ % phpunit BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
..F
```

```
Time: 0 seconds
```

```
There was 1 failure:
```

```
1) testOneSpare(BowlingGameTest)
```

```
Failed asserting that <integer:13> matches expected value <integer:16>.  
/home/sb/BowlingGameTest.php:38
```

```
FAILURES!
```

```
Tests: 3, Assertions: 3, Failures: 1.
```

# The Bowling Game Kata

## The third test

```
<?php
class BowlingGame
{
    protected $score = 0;

    public function roll($pins)
    {
        $this->score += $pins;
    }

    public function score()
    {
        return $this->score;
    }
}
```

# The Bowling Game Kata

## The third test

```
<?php
class BowlingGame
{
    protected $score = 0;
    protected $rolls = array();

    public function roll($pins)
    {
        $this->score += $pins;
        $this->rolls[] = $pins;
    }

    public function score()
    {
        return $this->score;
    }
}
```

# The Bowling Game Kata

## The third test

```
<?php
class BowlingGame
{
    protected $rolls = array();

    public function roll($pins)
    {
        $this->rolls[] = $pins;
    }

    public function score()
    {
        $score = 0;

        foreach ($this->rolls as $roll) {
            $score += $roll;
        }

        return $score;
    }
}
```

# The Bowling Game Kata

## The third test

```
sb@vmware ~ % phpunit BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
..F
```

```
Time: 0 seconds
```

```
There was 1 failure:
```

```
1) testOneSpare(BowlingGameTest)
```

```
Failed asserting that <integer:13> matches expected value <integer:16>.  
/home/sb/BowlingGameTest.php:38
```

```
FAILURES!
```

```
Tests: 3, Assertions: 3, Failures: 1.
```

# The Bowling Game Kata

## The third test

```
<?php
class BowlingGame
{
    // ...

    public function score()
    {
        $score = 0;

        for ($i = 0; $i < count($this->rolls); $i++) {
            if ($this->rolls[$i] + $this->rolls[$i + 1] == 10) {
                // ...
            }

            $score += $this->rolls[$i];
        }

        return $score;
    }
}
```

# The Bowling Game Kata

## The third test

```
<?php
class BowlingGame
{
    // ...

    public function score()
    {
        $score = 0;
        $i      = 0;

        for ($frame = 0; $frame < 10; $frame++) {
            $score += $this->rolls[$i] + $this->rolls[$i + 1];
            $i      += 2;
        }

        return $score;
    }
}
```

# The Bowling Game Kata

## The third test

```
sb@vmware ~ % phpunit BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
..F
```

```
Time: 0 seconds
```

```
There was 1 failure:
```

```
1) testOneSpare(BowlingGameTest)
```

```
Failed asserting that <integer:13> matches expected value <integer:16>.  
/home/sb/BowlingGameTest.php:38
```

```
FAILURES!
```

```
Tests: 3, Assertions: 3, Failures: 1.
```

# The Bowling Game Kata

## The third test

```
<?php
class BowlingGame
{
    // ...

    public function score()
    {
        $score = 0;
        $i      = 0;

        for ($frame = 0; $frame < 10; $frame++) {
            // Spare
            if ($this->rolls[$i] + $this->rolls[$i + 1] == 10) {
                $score += 10 + $this->rolls[$i + 2];
            } else {
                $score += $this->rolls[$i] + $this->rolls[$i + 1];
            }

            $i += 2;
        }

        return $score;
    }
}
```

# The Bowling Game Kata

## The third test

```
sb@vmware ~ % phpunit BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
...
```

```
Time: 0 seconds
```

```
OK (3 tests, 3 assertions)
```

# The Bowling Game Kata

## The third test

```
<?php
class BowlingGame
{
    // ...

    public function score()
    {
        $score = 0;
        $i      = 0;

        for ($frame = 0; $frame < 10; $frame++) {
            // Spare
            if ($this->rolls[$i] + $this->rolls[$i + 1] == 10) {
                $score += 10 + $this->rolls[$i + 2];
            } else {
                $score += $this->rolls[$i] + $this->rolls[$i + 1];
            }

            $i += 2;
        }

        return $score;
    }
}
```

# The Bowling Game Kata

## The third test

```
<?php
class BowlingGame
{
    // ...

    public function score()
    {
        $score = 0;
        $i      = 0;

        for ($frame = 0; $frame < 10; $frame++) {
            // Spare
            if ($this->rolls[$i] + $this->rolls[$i + 1] == 10) {
                $score += 10 + $this->rolls[$i + 2];
            } else {
                $score += $this->rolls[$i] + $this->rolls[$i + 1];
            }

            $i += 2;
        }

        return $score;
    }
}
```

# The Bowling Game Kata

## The third test

```
<?php
class BowlingGame
{
    // ...

    public function score()
    {
        $score      = 0;
        $frameIndex = 0;

        for ($frame = 0; $frame < 10; $frame++) {
            // Spare
            if ($this->rolls[$frameIndex] + $this->rolls[$frameIndex + 1] == 10) {
                $score += 10 + $this->rolls[$frameIndex + 2];
            } else {
                $score += $this->rolls[$frameIndex] + $this->rolls[$frameIndex + 1];
            }

            $frameIndex += 2;
        }

        return $score;
    }
}
```

# The Bowling Game Kata

## The third test

```
<?php
class BowlingGame
{
    // ...

    public function score()
    {
        $score      = 0;
        $frameIndex = 0;

        for ($frame = 0; $frame < 10; $frame++) {
            if ($this->isSpare($frameIndex)) {
                $score += 10 + $this->rolls[$frameIndex + 2];
            } else {
                $score += $this->rolls[$frameIndex] + $this->rolls[$frameIndex + 1];
            }

            $frameIndex += 2;
        }

        return $score;
    }

    protected function isSpare($frameIndex)
    {
        return $this->rolls[$frameIndex] + $this->rolls[$frameIndex + 1] == 10;
    }
}
```

# The Bowling Game Kata

## The third test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testOneSpare()
    {
        $this->game->roll(5);
        $this->game->roll(5); // Spare
        $this->game->roll(3);
        $this->rollMany(17, 0);
        $this->assertEquals(16, $this->game->score());
    }
}
```

# The Bowling Game Kata

## The third test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testOneSpare()
    {
        $this->game->roll(5);
        $this->game->roll(5); // Spare
        $this->game->roll(3);
        $this->rollMany(17, 0);
        $this->assertEquals(16, $this->game->score());
    }
}
```

# The Bowling Game Kata

## The third test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    // ...

    protected function rollSpare()
    {
        $this->game->roll(5);
        $this->game->roll(5);
    }

    // ...

    public function testOneSpare()
    {
        $this->rollSpare();
        $this->game->roll(3);
        $this->rollMany(17, 0);
        $this->assertEquals(16, $this->game->score());
    }
}
```

# The Bowling Game Kata

## The fourth test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testOneStrike()
    {
        $this->game->roll(10); // Strike
        $this->game->roll(3);
        $this->game->roll(4);
        $this->rollMany(17, 0);
        $this->assertEquals(24, $this->game->score());
    }
}
```

# The Bowling Game Kata

## The fourth test

```
sb@vmware ~ % phpunit BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
...F
```

```
Time: 0 seconds
```

```
There was 1 failure:
```

```
1) testOneStrike(BowlingGameTest)
```

```
Failed asserting that <integer:17> matches expected value <integer:24>.  
/home/sb/BowlingGameTest.php:52
```

```
FAILURES!
```

```
Tests: 4, Assertions: 4, Failures: 1.
```

# The Bowling Game Kata

## The fourth test

```
<?php
class BowlingGame
{
    // ...

    public function score()
    {
        $score      = 0;
        $frameIndex = 0;

        for ($frame = 0; $frame < 10; $frame++) {
            if ($this->isSpare($frameIndex)) {
                $score += 10 + $this->rolls[$frameIndex + 2];
            }

            else {
                $score += $this->rolls[$frameIndex] + $this->rolls[$frameIndex + 1];
            }

            $frameIndex += 2;
        }

        return $score;
    }
}
```

# The Bowling Game Kata

## The fourth test

```
<?php
class BowlingGame
{
    // ...

    public function score()
    {
        $score      = 0;
        $frameIndex = 0;

        for ($frame = 0; $frame < 10; $frame++) {
            // Strike
            if ($this->rolls[$frameIndex] == 10) {
                $score += 10
                    + $this->rolls[$frameIndex + 1] + $this->rolls[$frameIndex + 2];
                $frameIndex++;
            }

            // ...
        }

        return $score;
    }
}
```

# The Bowling Game Kata

## The fourth test

```
sb@vmware ~ % phpunit BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
....
```

```
Time: 0 seconds
```

```
OK (4 tests, 4 assertions)
```

# The Bowling Game Kata

## The fourth test

```
<?php
class BowlingGame
{
    // ...

    public function score()
    {
        $score      = 0;
        $frameIndex = 0;

        for ($frame = 0; $frame < 10; $frame++) {
            // Strike
            if ($this->rolls[$frameIndex] == 10) {
                $score += 10
                    + $this->rolls[$frameIndex + 1] + $this->rolls[$frameIndex + 2];
                $frameIndex++;
            }

            // ...
        }

        return $score;
    }
}
```

# The Bowling Game Kata

## The fourth test

```
<?php
class BowlingGame
{
    // ...

    public function score()
    {
        $score      = 0;
        $frameIndex = 0;

        for ($frame = 0; $frame < 10; $frame++) {
            if ($this->isStrike($frameIndex)) {
                $score += 10
                    + $this->rolls[$frameIndex + 1] + $this->rolls[$frameIndex + 2];
                $frameIndex++;
            }

            // ...
        }

        return $score;
    }

    protected function isStrike($frameIndex)
    {
        return $this->rolls[$frameIndex] == 10;
    }
}
```

# The Bowling Game Kata

## The fourth test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testOneStrike()
    {
        $this->game->roll(10); // Strike
        $this->game->roll(3);
        $this->game->roll(4);
        $this->rollMany(17, 0);
        $this->assertEquals(24, $this->game->score());
    }
}
```

# The Bowling Game Kata

## The fourth test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    // ...

    protected function rollStrike()
    {
        $this->game->roll(10);
    }

    // ...

    public function testOneStrike()
    {
        $this->rollStrike();
        $this->game->roll(3);
        $this->game->roll(4);
        $this->rollMany(17, 0);
        $this->assertEquals(24, $this->game->score());
    }
}
```

# The Bowling Game Kata

## The fourth test

```
sb@vmware ~ % phpunit BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
....
```

```
Time: 0 seconds
```

```
OK (4 tests, 4 assertions)
```

# The Bowling Game Kata

## The fifth test

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testPerfectGame()
    {
        $this->rollMany(12, 10);
        $this->assertEquals(300, $this->game->score());
    }
}
```

# The Bowling Game Kata

## The fifth test

```
sb@vmware ~ % phpunit BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
.....
```

```
Time: 0 seconds
```

```
OK (5 tests, 5 assertions)
```

# The Bowling Game Kata

Final version of the BowlingGame class

```
<?php
class BowlingGame
{
    protected $rolls = array();

    public function roll($pins)
    {
        $this->rolls[] = $pins;
    }

    // ...
}
```

# The Bowling Game Kata

Final version of the BowlingGame class

```
<?php
class BowlingGame
{
    // ...

    protected function isSpare($frameIndex)
    {
        return $this->sumOfPinsInFrame($frameIndex) == 10;
    }

    protected function isStrike($frameIndex)
    {
        return $this->rolls[$frameIndex] == 10;
    }

    protected function sumOfPinsInFrame($frameIndex)
    {
        return $this->rolls[$frameIndex] +
            $this->rolls[$frameIndex + 1];
    }
}
```

# The Bowling Game Kata

Final version of the BowlingGame class

```
<?php
class BowlingGame
{
    // ...

    protected function spareBonus($frameIndex)
    {
        return $this->rolls[$frameIndex + 2];
    }

    protected function strikeBonus($frameIndex)
    {
        return $this->rolls[$frameIndex + 1] +
            $this->rolls[$frameIndex + 2];
    }
}
```

# The Bowling Game Kata

## Final version of the BowlingGame class

```
<?php
class BowlingGame
{
    // ...

    public function score()
    {
        $score      = 0;
        $frameIndex = 0;

        for ($frame = 0; $frame < 10; $frame++) {
            if ($this->isStrike($frameIndex)) {
                $score += 10 + $this->strikeBonus($frameIndex);
                $frameIndex++;
            }

            else if ($this->isSpare($frameIndex)) {
                $score += 10 + $this->spareBonus($frameIndex);
                $frameIndex += 2;
            }

            else {
                $score += $this->sumOfPinsInFrame($frameIndex);
                $frameIndex += 2;
            }
        }

        return $score;
    }
}
```

# The Bowling Game Kata

## The fifth test

```
sb@vmware ~ % phpunit BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
.....
```

```
Time: 0 seconds
```

```
OK (5 tests, 5 assertions)
```

# PHPUnit

## Agile Documentation

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testGutterGame() { // ... }

    public function testAllOnes() { // ... }

    public function testOneSpare() { // ... }

    public function testOneStrike() { // ... }

    public function testPerfectGame() { // ... }
}
```

# PHPUnit

## Agile Documentation

```
<?php
require_once 'BowlingGame.php';

class BowlingGameTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testScoreForGutterGameIs0() { // ... }

    public function testScoreForAllOnesIs20() { // ... }

    public function testScoreForOneSpareAnd3Is16() { // ... }

    public function testScoreForOneStrikeAnd3And4Is24() { // ... }

    public function testScoreForPerfectGameIs300() { // ... }
}
```

# PHPUnit

## Agile Documentation

```
sb@vmware ~ % phpunit --testdox BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

### BowlingGame

```
[x] Score for gutter game is 0  
[x] Score for all ones is 20  
[x] Score for one spare and 3 is 16  
[x] Score for one strike and 3 and 4 is 24  
[x] Score for perfect game is 300
```

# Behaviour-Driven Development

- Extreme Programming originally had the rule to test everything that could possibly break
- Now, however, the practice of testing in Extreme Programming has evolved into Test-Driven Development
- But the tools still force developers to think in terms of tests and assertions instead of specifications

*A New Look At Test-Driven Development,*  
Dave Astels.

[http://blog.daveastels.com/files/BDD\\_Intro.pdf](http://blog.daveastels.com/files/BDD_Intro.pdf)

# Behaviour-Driven Development

```
<?php
require_once 'PHPUnit/Extensions/Story/TestCase.php';
require_once 'BowlingGame.php';

class BowlingGameSpec extends PHPUnit_Extensions_Story_TestCase
{

}
}
```

# Behaviour-Driven Development

```
<?php
require_once 'PHPUnit/Extensions/Story/TestCase.php';
require_once 'BowlingGame.php';

class BowlingGameSpec extends PHPUnit_Extensions_Story_TestCase
{
    /**
     * @scenario
     */
    public function scoreForOneSpareAnd3Is16()
    {

    }

    // ...
}
```

# Behaviour-Driven Development

```
<?php
require_once 'PHPUnit/Extensions/Story/TestCase.php';
require_once 'BowlingGame.php';

class BowlingGameSpec extends PHPUnit_Extensions_Story_TestCase
{
    /**
     * @scenario
     */
    public function scoreForOneSpareAnd3Is16()
    {
        $this->given('New game')

    }

    // ...
}
```

# Behaviour-Driven Development

```
<?php
require_once 'PHPUnit/Extensions/Story/TestCase.php';
require_once 'BowlingGame.php';

class BowlingGameSpec extends PHPUnit_Extensions_Story_TestCase
{
    /**
     * @scenario
     */
    public function scoreForOneSpareAnd3Is16()
    {
        $this->given('New game')
            ->when('Player rolls', 5)

    }

    // ...
}
```

# Behaviour-Driven Development

```
<?php
require_once 'PHPUnit/Extensions/Story/TestCase.php';
require_once 'BowlingGame.php';

class BowlingGameSpec extends PHPUnit_Extensions_Story_TestCase
{
    /**
     * @scenario
     */
    public function scoreForOneSpareAnd3Is16()
    {
        $this->given('New game')
            ->when('Player rolls', 5)
            ->and('Player rolls', 5)

    }

    // ...
}
```

# Behaviour-Driven Development

```
<?php
require_once 'PHPUnit/Extensions/Story/TestCase.php';
require_once 'BowlingGame.php';

class BowlingGameSpec extends PHPUnit_Extensions_Story_TestCase
{
    /**
     * @scenario
     */
    public function scoreForOneSpareAnd3Is16()
    {
        $this->given('New game')
            ->when('Player rolls', 5)
            ->and('Player rolls', 5)
            ->and('Player rolls', 3)

    }

    // ...
}
```

# Behaviour-Driven Development

```
<?php
require_once 'PHPUnit/Extensions/Story/TestCase.php';
require_once 'BowlingGame.php';

class BowlingGameSpec extends PHPUnit_Extensions_Story_TestCase
{
    /**
     * @scenario
     */
    public function scoreForOneSpareAnd3Is16()
    {
        $this->given('New game')
            ->when('Player rolls', 5)
            ->and('Player rolls', 5)
            ->and('Player rolls', 3)
            ->then('Score should be', 16);
    }

    // ...
}
```

# Behaviour-Driven Development

```
<?php
require_once 'PHPUnit/Extensions/Story/TestCase.php';
require_once 'BowlingGame.php';

class BowlingGameSpec extends PHPUnit_Extensions_Story_TestCase
{
    // ...

    public function runGiven(&$world, $action, $arguments)
    {
        switch($action) {
            case 'New game': {
                $world['game'] = new BowlingGame;
                $world['rolls'] = 0;
            }
            break;

            default: {
                return $this->notImplemented($action);
            }
        }
    }
}
```

# Behaviour-Driven Development

```
<?php
require_once 'PHPUnit/Extensions/Story/TestCase.php';
require_once 'BowlingGame.php';

class BowlingGameSpec extends PHPUnit_Extensions_Story_TestCase
{
    // ...

    public function runWhen(&$world, $action, $arguments)
    {
        switch($action) {
            case 'Player rolls': {
                $world['game']->roll($arguments[0]);
                $world['rolls']++;
            }
            break;

            default: {
                return $this->notImplemented($action);
            }
        }
    }
}
```

# Behaviour-Driven Development

```
<?php
require_once 'PHPUnit/Extensions/Story/TestCase.php';
require_once 'BowlingGame.php';

class BowlingGameSpec extends PHPUnit_Extensions_Story_TestCase
{
    // ...

    public function runThen(&$world, $action, $arguments)
    {
        switch($action) {
            case 'Score should be': {
                for ($i = $world['rolls']; $i < 20; $i++) {
                    $world['game']->roll(0);
                }

                $this->assertEquals($arguments[0], $world['game']->score());
            }
            break;

            default: {
                return $this->notImplemented($action);
            }
        }
    }
}
```

# Behaviour-Driven Development

```
sb@vmware ~ % phpunit --story BowlingGameSpec  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
BowlingGameSpec
```

```
- Score for one spare and 3 is 16 [successful]
```

```
Given New game
```

```
When Player rolls 5
```

```
and Player rolls 5
```

```
and Player rolls 3
```

```
Then Score should be 16
```

```
Scenarios: 1, Failed: 0, Skipped: 0, Incomplete: 0.
```

```
sb@vmware ~ % phpunit --testdox BowlingGameSpec  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
BowlingGameSpec
```

```
[x] Score for one spare and 3 is 16
```

# PHPUnit

## Code Coverage

```
sb@vmware ~ % phpunit --coverage-html report BowlingGameTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
.....
```

```
Time: 0 seconds
```

```
OK (5 tests)
```

```
Generating report, this may take a moment.
```

# Organizing Test Suites

Application/

– Package/

- Application\_Package\_Class  
Application/Package/Class.php
- ...

– ...

– Tests/

- AllTests.php
- Package/
  - AllTests.php
  - Application\_Package\_ClassTest  
Application/Tests/Package/ClassTest.php

# Organizing Test Suites

Application/Tests/AllTests.php

```
<?php
require_once 'PHPUnit/Framework.php';

require_once 'Application/Tests/Package/AllTests.php';

class AllTests
{
    public static function suite()
    {
        $suite = new PHPUnit_Framework_TestSuite('Project');
        $suite->addTest(Package_AllTests::suite());

        return $suite;
    }
}
?>
```

# Organizing Test Suites

Application/Tests/Package/AllTests.php

```
<?php
require_once 'PHPUnit/Framework.php';

require_once 'Application/Tests/Package/ClassTest.php';

class Package_AllTests
{
    public static function suite()
    {
        $suite = new PHPUnit_Framework_TestSuite('Package');
        $suite->addTestSuite('Package_ClassTest');

        return $suite;
    }
}
?>
```

# Organizing Test Suites

Application/Tests/Package/ClassTest.php

```
<?php
require_once 'PHPUnit/Framework.php';

require_once 'Application/Package/ClassTest.php';

class Package_ClassTest extends PHPUnit_Framework_TestCase
{
    public function testSomething()
    {
        // ...
    }
}
?>
```

# Organizing Test Suites

## Running the tests

- Executing `phpunit AllTests` in the Tests directory will run all tests.
- Executing `phpunit AllTests` in the Tests/Package directory will run only the tests for the `Application_Package_*` classes.
- Executing `phpunit ClassTest` in the Tests/Framework directory will run only the tests for the `Application_Package_Class` class (which are declared in the `Package_ClassTest` class).
- Executing `phpunit --filter testSomething ClassTest` in the Tests/Package directory will run only the test named `testSomething` from the `Package_ClassTest` class.

# Organizing Test Suites

## The @group annotation

```
<?php
class SomeTest extends PHPUnit_Framework_TestCase
{
    /**
     * @group specification
     */
    public function testSomething()
    {
    }

    /**
     * @group regression
     * @group bug2204
     */
    public function testSomethingElse()
    {
    }
}
```

# Organizing Test Suites

## The @group annotation

```
sb@vmware ~ % phpunit SomeTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
..
```

```
Time: 0 seconds
```

```
OK (2 tests, 2 assertions)
```

```
sb@vmware ~ % phpunit --group bug2204 SomeTest  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
.
```

```
Time: 0 seconds
```

```
OK (1 test, 1 assertion)
```

# Annotations

## @dataProvider

```
<?php
class DataTest extends PHPUnit_Framework_TestCase
{
    /**
     * @dataProvider providerMethod
     */
    public function testAdd($a, $b, $c)
    {
        $this->assertEquals($c, $a + $b);
    }

    public function providerMethod()
    {
        return array(
            array(0, 0, 0),
            array(0, 1, 1),
            array(1, 1, 3),
            array(1, 0, 1)
        );
    }
}
```

# Annotations

@dataProvider

```
sb@vmware ~ % phpunit DataTest
PHPUnit 3.3.0 by Sebastian Bergmann.

..F.

Time: 0 seconds

There was 1 failure:

1) testAdd(DataTest) with data (1, 1, 3)
Failed asserting that <integer:2> matches expected
value <integer:3>.
/home/sb/DataTest.php:19

FAILURES!
Tests: 4, Assertions: 4, Failures: 1.
```

# Annotations

## @expectedException

```
<?php
class ExceptionTest extends PHPUnit_Framework_TestCase
{
    /**
     * @expectedException InvalidArgumentException
     */
    public function testException()
    {
    }
}
```

# Annotations

`@expectedException`

```
sb@vmware ~ % phpunit ExceptionTest
PHPUnit 3.3.0 by Sebastian Bergmann.

F

Time: 0 seconds

There was 1 failure:

1) testException(ExceptionTest)
Expected exception InvalidArgumentException

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

# Annotations

## @test

```
<?php
class Specification extends PHPUnit_Framework_TestCase
{
    /**
     * @test
     */
    public function shouldDoSomething()
    {
    }

    /**
     * @test
     */
    public function shouldDoSomethingElse()
    {
    }
}
```

# Annotations

@test

```
sb@vmware ~ % phpunit --testdox Specification  
PHPUnit 3.3.0 by Sebastian Bergmann.
```

```
Specification
```

```
[x] Should do something
```

```
[x] Should do something else
```

# Annotations

## @assert

```
<?php
class Calculator
{
    /**
     * @assert (1, 2) == 3
     */
    public function add($a, $b)
    {
        return $a + $b;
    }

    /**
     * @assert (2, 1) == 1
     */
    public function sub($a, $b)
    {
        return $a - $b;
    }
}
```

# Annotations

@assert

```
sb@vmware ~ % phpunit Calculator
PHPUnit 3.3.0 by Sebastian Bergmann.

..

Time: 0 seconds

OK (2 tests, 2 assertions)
```

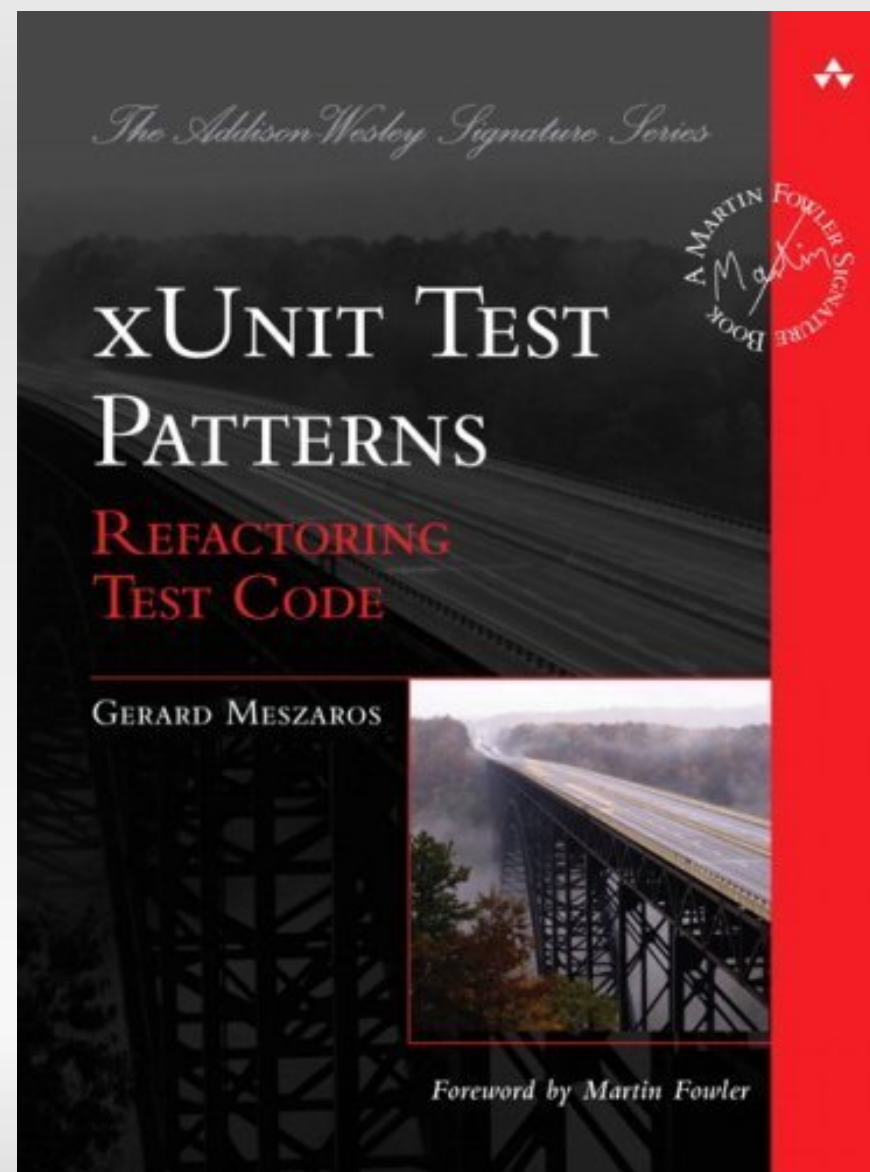
# Test Doubles

- *How can we verify logic independently when code it depends on is unusable?*
- *How can we avoid slow tests?*
- **We replace a component on which the SUT depends with a “test-specific equivalent”.**

# Test Doubles

## Terminology

- Dummy
  - Not the real object
- Fake
  - Usable for testing but not for real job
- Stub
  - Fake that returns canned data
- Spy
  - Stub that records called methods, etc.
- Mock
  - Spy with expectations



# Test Doubles

## Stubs

```
<?php
require_once 'PHPUnit/Framework.php';

class StubTest extends PHPUnit_Framework_TestCase
{
    public function testStub()
    {

    }
}
?>
```

# Test Doubles

## Stubs

```
<?php
require_once 'PHPUnit/Framework.php';

class StubTest extends PHPUnit_Framework_TestCase
{
    public function testStub()
    {
        $stub = $this->getMock('SomeClass');
    }
}
?>
```

# Test Doubles

Stubs: returnValue()

```
<?php
require_once 'PHPUnit/Framework.php';

class StubTest extends PHPUnit_Framework_TestCase
{
    public function testStub()
    {
        $stub = $this->getMock('SomeClass');
        $stub->expects($this->any())
            ->method('doSomething')
            ->will($this->returnValue('foo'));
    }
}
?>
```

# Test Doubles

Stubs: returnValue()

```
<?php
require_once 'PHPUnit/Framework.php';

class StubTest extends PHPUnit_Framework_TestCase
{
    public function testStub()
    {
        $stub = $this->getMock('SomeClass');
        $stub->expects($this->any())
            ->method('doSomething')
            ->will($this->returnValue('foo'));

        // Calling $stub->doSomething() will now return
        // 'foo'.
    }
}
?>
```

# Test Doubles

## Stubs: returnArgument()

```
<?php
class StubTest extends PHPUnit_Framework_TestCase
{
    public function testReturnArgumentStub()
    {
        $stub = $this->getMock(
            'SomeClass', array('doSomething')
        );

        $stub->expects($this->any())
            ->method('doSomething')
            ->will($this->returnArgument(0));

        // $stub->doSomething('foo') returns 'foo'
        // $stub->doSomething('bar') returns 'bar'
    }
}
```

# Test Doubles

## Stubs: returnCallback()

```
<?php
class StubTest extends PHPUnit_Framework_TestCase
{
    public function testReturnCallbackStub()
    {
        $stub = $this->getMock(
            'SomeClass', array('doSomething')
        );

        $stub->expects($this->any())
            ->method('doSomething')
            ->will($this->returnCallback('callback'));

        // $stub->doSomething() returns callback(...)
    }
}

function callback() {
    $args = func_get_args();
    // ...
}
```

# Test Doubles

## Stubs: `throwException()`

```
<?php
class StubTest extends PHPUnit_Framework_TestCase
{
    public function testThrowExceptionStub()
    {
        $stub = $this->getMock(
            'SomeClass', array('doSomething')
        );

        $stub->expects($this->any())
            ->method('doSomething')
            ->will($this->throwException(new Exception));

        // $stub->doSomething() throws Exception
    }
}
```

# Test Doubles

## Mock Objects

```
<?php
require_once 'PHPUnit/Framework.php' ;

class ObserverTest extends PHPUnit_Framework_TestCase {
    public function testUpdateIsCalledOnce() {

    }
}
?>
```

# Test Doubles

## Mock Objects

```
<?php
require_once 'PHPUnit/Framework.php';

class ObserverTest extends PHPUnit_Framework_TestCase {
    public function testUpdateIsCalledOnce() {
        $observer = $this->getMock(
            'Observer', array('update')
        );
    }
}
?>
```

# Test Doubles

## Mock Objects

```
<?php
require_once 'PHPUnit/Framework.php';

class ObserverTest extends PHPUnit_Framework_TestCase {
    public function testUpdateIsCalledOnce() {
        $observer = $this->getMock(
            'Observer', array('update')
        );

        $observer->expects($this->once())
            ->method('update')
            ->with($this->equalTo('something'));

    }
}
?>
```

# Test Doubles

## Mock Objects

```
<?php
require_once 'PHPUnit/Framework.php';

class ObserverTest extends PHPUnit_Framework_TestCase {
    public function testUpdateIsCalledOnce() {
        $observer = $this->getMock(
            'Observer', array('update')
        );

        $observer->expects($this->once())
            ->method('update')
            ->with($this->equalTo('something'));

        $subject = new Subject;
        $subject->attach($observer);
        $subject->doSomething();
    }
}
?>
```

- Michael Lively Jr. has ported the DbUnit extension for JUnit to PHPUnit
  - PHPUnit\_Extensions\_Database\_TestCase
    - is used to test database-driven projects and
    - puts your database into a known state between test runs
      - This avoids problems with one test corrupting the database for other tests
    - has the ability to export and import your database data to and from XML datasets

# DbUnit

- DbUnit uses PDO to connect to the database-under-test
- The tested application does not have to use PDO itself for this to work
  - You can therefore use `ext/mysqli` in your application and `ext/pdo_mysql` in your tests, for instance

# DbUnit

## BankAccountDBTest.php

```
<?php
require_once 'PHPUnit/Extensions/Database/Testcase.php';

class BankAccountDBTest extends PHPUnit_Extensions_Database_TestCase {

}
```

# DbUnit

## BankAccountDBTest.php

```
<?php
require_once 'PHPUnit/Extensions/Database/Testcase.php';

class BankAccountDBTest extends PHPUnit_Extensions_Database_TestCase {
    protected $pdo;

    public function __construct() {
        $this->pdo = PHPUnit_Util_PDO::factory(
            'mysql://test@localhost/test'
        );

        BankAccount::createTable($this->pdo);
    }
}
```

# DbUnit

## BankAccountDBTest.php

```
<?php
require_once 'PHPUnit/Extensions/Database/Testcase.php';

class BankAccountDBTest extends PHPUnit_Extensions_Database_TestCase {
    protected $pdo;

    public function __construct() {
        $this->pdo = PHPUnit_Util_PDO::factory(
            'mysql://test@localhost/test'
        );

        BankAccount::createTable($this->pdo);
    }

    protected function getConnection() {
        return $this->createDefaultDBConnection($this->pdo, 'mysql');
    }
}
```

# DbUnit

## BankAccountDBTest.php

```
<?php
require_once 'PHPUnit/Extensions/Database/Testcase.php';

class BankAccountDBTest extends PHPUnit_Extensions_Database_TestCase {
    protected $pdo;

    public function __construct() {
        $this->pdo = PHPUnit_Util_PDO::factory(
            'mysql://test@localhost/test'
        );

        BankAccount::createTable($this->pdo);
    }

    protected function getConnection() {
        return $this->createDefaultDBConnection($this->pdo, 'mysql');
    }

    protected function getDataSet() {
        return $this->createFlatXMLDataSet('/path/to/seed.xml');
    }
}
```

# DbUnit

## seed.xml

```
<dataset>  
  <account account_number="15934903649620486" balance="100.00" />  
  <account account_number="15936487230215067" balance="1216.00" />  
  <account account_number="12348612357236185" balance="89.00" />  
  <account account_number="15936487230215067" balance="1216.00" />  
</dataset>
```

# DbUnit

## BankAccountDBTest.php

```
<?php
require_once 'PHPUnit/Extensions/Database/Testcase.php';

class BankAccountDBTest extends PHPUnit_Extensions_Database_TestCase {
    // ...

    public function testNewAccount() {

    }
}
```

# DbUnit

## BankAccountDBTest.php

```
<?php
require_once 'PHPUnit/Extensions/Database/Testcase.php';

class BankAccountDBTest extends PHPUnit_Extensions_Database_TestCase {
    // ...

    public function testNewAccount() {
        $ba = new BankAccountDB('12345678912345678', $this->pdo);
    }
}
```

# DbUnit

## BankAccountDBTest.php

```
<?php
require_once 'PHPUnit/Extensions/Database/Testcase.php';

class BankAccountDBTest extends PHPUnit_Extensions_Database_TestCase {
    // ...

    public function testNewAccount() {
        $ba = new BankAccountDB('12345678912345678', $this->pdo);

        $set = $this->createFlatXMLDataSet(
            '/path/to/after-new-account.xml'
        );
    }
}
```

# DbUnit

## BankAccountDBTest.php

```
<?php
require_once 'PHPUnit/Extensions/Database/Testcase.php';

class BankAccountDBTest extends PHPUnit_Extensions_Database_TestCase {
    // ...

    public function testNewAccount() {
        $ba = new BankAccountDB('12345678912345678', $this->pdo);

        $set = $this->createFlatXMLDataSet(
            '/path/to/after-new-account.xml'
        );

        $this->assertTablesEqual(
            $set->getTable('account'),
            $this->getConnection()
                ->createDataSet()
                ->getTable('account')
        );
    }
}
```

# DbUnit

## after-new-account.xml

```
<dataset>  
  <account account_number="15934903649620486" balance="100.00" />  
  <account account_number="15936487230215067" balance="1216.00" />  
  <account account_number="12348612357236185" balance="89.00" />  
  <account account_number="15936487230215067" balance="1216.00" />  
  <account account_number="12345678912345678" balance="0.00" />  
</dataset>
```

# Test against SQLite if you can

- When testing PHP code that uses PDO to connect to a database, it makes sense to keep your SQL compatible with SQLite
  - No server  $\Rightarrow$  No inter-process communication
  - In-Memory Databases  $\Rightarrow$  No Disk I/O

	User	System	CPU	Total
PDO / MySQL	3.95s	0.87s	40%	12.046s
PDO / SQLite (file)	5.01s	1.54s	63%	10.359s
PDO / SQLite (memory)	3.16s	0.68s	99%	3.849s

# Selenium

- Selenium
  - Test web applications in a web browser
    - Browser Compatibility Testing
    - System Functional Testing
  - Runs in the browser
- Selenium IDE
  - IDE for Selenium tests
    - Extension for Firefox
    - Record, execute, edit, debug tests in the browser

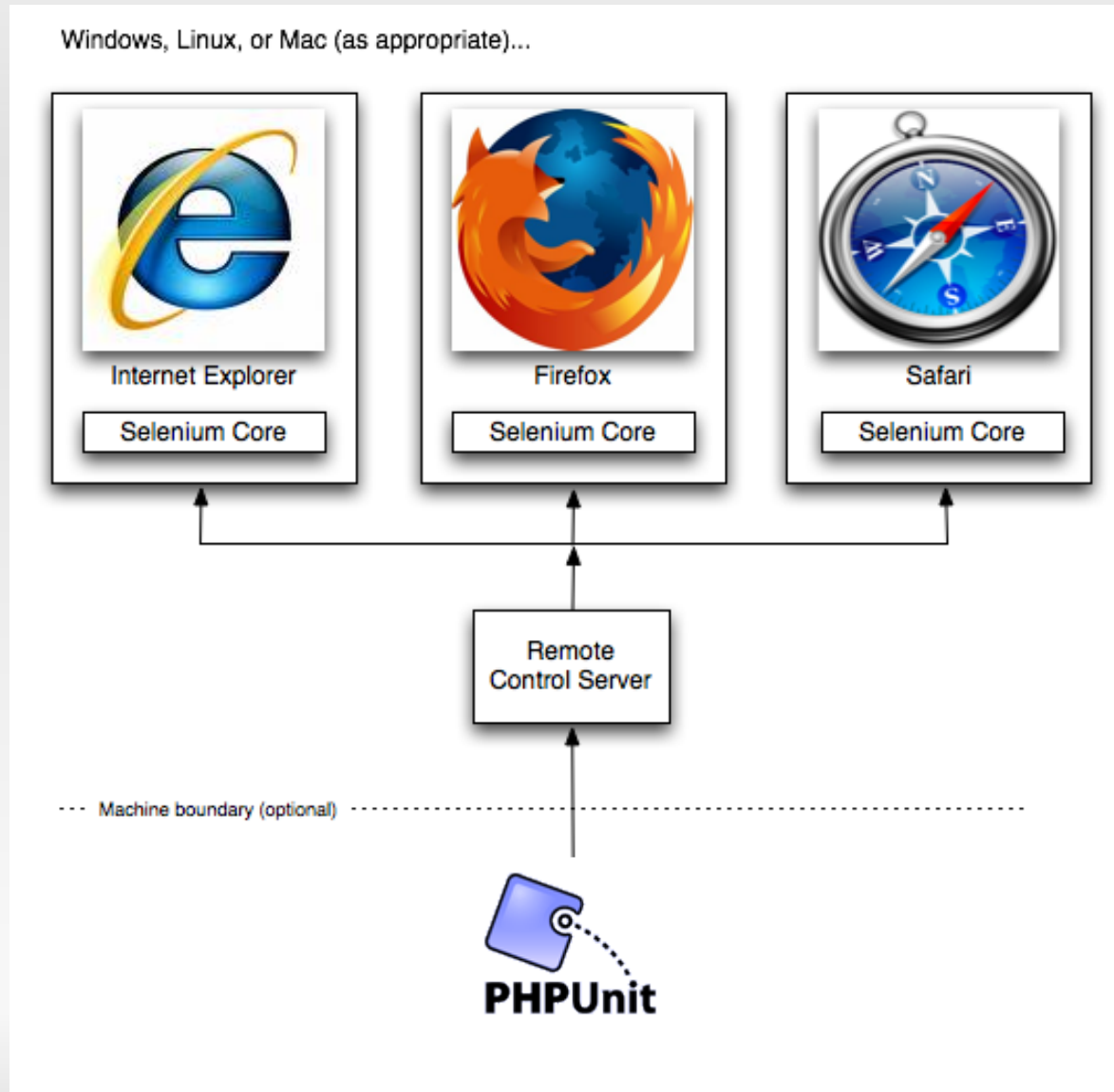
# Selenium

## Selenium RC

- Selenium RC
  - Automated execution of Selenium tests
  - Tests can be specified in any language
    - PHP Bindings: PEAR Testing\_Selenium
    - PHPUnit natively speaks the Selenium RC protocol
  - One test can be executed on multiple OS / Browser combinations

# Selenium

## Selenium RC



# Continuous Integration

- Software development practice where members of a team integrate their work frequently

# Continuous Integration

- Software development practice where members of a team integrate their work frequently
  - Usually each person integrates at least daily, leading to multiple integrations per day

# Continuous Integration

- Software development practice where members of a team integrate their work frequently
  - Usually each person integrates at least daily, leading to multiple integrations per day
- Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible

# Continuous Integration

- Software development practice where members of a team integrate their work frequently
  - Usually each person integrates at least daily, leading to multiple integrations per day
- Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible
- This approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly

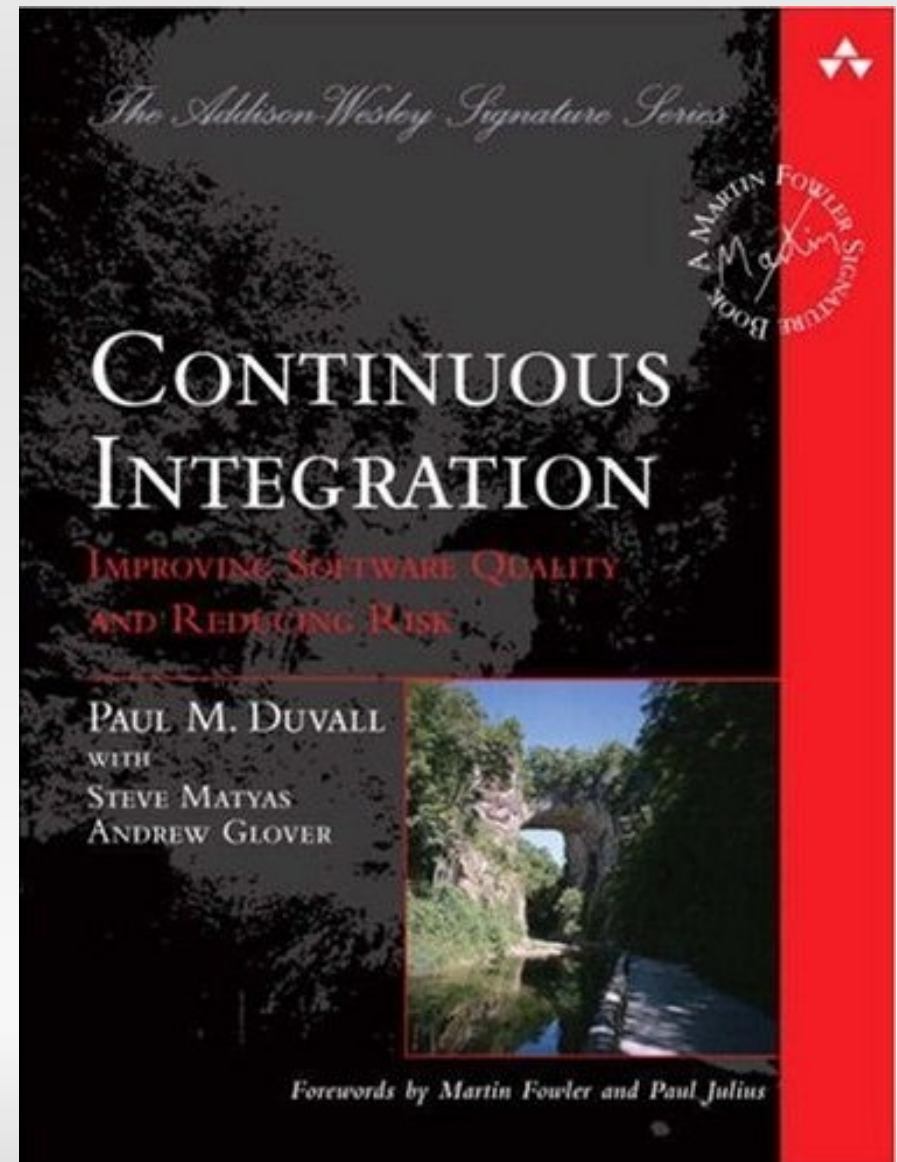
# Continuous Integration

Continuous Integration

Paul M. Duvall

Addison-Wesley, 2007

ISBN 978-0321336385



# Continuous Integration

## Software

- CruiseControl
  - phpUnderControl
- Bamboo
- Hudson
- Xinc
- ...

# CruiseControl

CruiseControl is a framework for a continuous build process

- It includes, but is not limited to, plugins for email notification, Ant, and various source control tools
- A web interface is provided to view the details of the current and previous builds

# CruiseControl

## phpUnderControl

phpUnderControl is customization of CruiseControl that caters to the needs of PHP projects

- PHPUnit
- PHPDocumentor
- PHP\_CodeSniffer
- (PHP\_Depend)
- (PHP\_CompatInfo)
- ...

# The End

- Thank you for your interest!
- These slides will be available shortly on <http://sebastian-bergmann.de/talks/>.

# License

This presentation material is published under the Attribution-Share Alike 3.0 Unported license.

You are free:

- ✓ **to Share** – to copy, distribute and transmit the work.
- ✓ **to Remix** – to adapt the work.

Under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

For any reuse or distribution, you must make clear to others the license terms of this work.

Any of the above conditions can be waived if you get permission from the copyright holder.

Nothing in this license impairs or restricts the author's moral rights.