Presented by,
MySQL AB® & O'Reilly Media, Inc.

Testing PHP/MySQL Applications with PHPUnit/DbUnit

Sebastian Bergmann
http://sebastian-bergmann.de/

April 15th 2008

Who I Am

- Sebastian Bergmann
- 30 years old next week
- Involved in the PHP Project since 2000
- Developer of PHPUnit
- Helps enterprises with (the introduction of) quality assurance for PHP projects

Who are you?

- Your experience with
  - PHP 5?
  - OOP?
  - Testing?
    - PHPUnit?
    - DbUnit?

Why test?

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

Debugging Sucks!

Testing Rocks!

What needs testing?

Web Application

- Backend
  - Business Logic
    - Database Layer
  - 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

Tools for Testing

- 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

Tools for Testing PHP / Web Applications

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

PHPUnit (http://phpun.it/)

- Test Framework
  - Member of the xUnit family of test frameworks
  - Mock Objects
  - DbUnit
- Integration
  - phpUnderControl
  - Selenium RC
- Even More Cool Stuff :-)
  - Code Coverage
  - Software Metrics and Project Mess Detection
  - ...

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
```

PHPUnit: A First Example

```php
<?php
class BankAccountTest extends PHPUnit_Framework_TestCase
{









}
```

PHPUnit: A First Example

```php
<?php
class BankAccountTest extends PHPUnit_Framework_TestCase
{

    public function testBalanceIsInitiallyZero()
    {

    }

}
```

PHPUnit: A First Example

```php
<?php
class BankAccountTest extends PHPUnit_Framework_TestCase
{

    public function testBalanceIsInitiallyZero()
    {
        $ba = new BankAccount;
        $this->assertEquals(0, $ba->getBalance());
    }

}
```

PHPUnit: A First Example

```php
<?php
class BankAccountTest extends PHPUnit_Framework_TestCase
{
    protected $ba;

    protected function setUp()
    {
        $this->ba = new BankAccount;
    }

    public function testBalanceIsInitiallyZero()
    {
        $this->assertEquals(0, $this->ba->getBalance());
    }

    // ...
}
```

PHPUnit: A First Example

```php
<?php
class BankAccountTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testBalanceCannotBecomeNegative()
    {
        try {
            $this->ba->withdrawMoney(1);
        }

        catch (BankAccountException $e) {
            $this->assertEquals(0, $this->ba->getBalance());

            return;
        }

        $this->fail();
    }

    // ...
}
```

PHPUnit: A First Example

```php
<?php
class BankAccountTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testBalanceCannotBecomeNegative()
    {
        try {
            $this->ba->depositMoney(-1);
        }

        catch (BankAccountException $e) {
            $this->assertEquals(0, $this->ba->getBalance());

            return;
        }

        $this->fail();
    }

    // ...
}
```

PHPUnit: A First Example

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

...

Time: 0 seconds


OK (3 tests)
```

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

BankAccount
 - Balance is initially zero
 - Balance cannot become negative
```

PHPUnit: The DbUnit Extension

- 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

PHPUnit: The DbUnit Extension

- 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

PHPUnit: The DbUnit Extension

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

class BankAccountDBTest extends PHPUnit_Extensions_Database_TestCase {

}
```

PHPUnit: The DbUnit Extension

```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);
    }









}
```

PHPUnit: The DbUnit Extension

```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');
    }



}
```

PHPUnit: The DbUnit Extension

```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');
    }
}
```

PHPUnit: The DbUnit Extension

```
<dataset>
  <account account_number="159349036496204 86" balance="100.00" />
  <account account_number="159364872302150 67" balance="1216.00" />
  <account account_number="123486123572361 85" balance="89.00" />
  <account account_number="159364872302150 67" balance="1216.00" />
</dataset>
```

PHPUnit: The DbUnit Extension

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

class BankAccountDBTest extends PHPUnit_Extensions_Database_TestCase {
    // ...

    public function testNewAccount() {









    }
}
```

PHPUnit: The DbUnit Extension

```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);









    }
}
```

PHPUnit: The DbUnit Extension

```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'
        );









    }
}
```

PHPUnit: The DbUnit Extension

```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')
        );
    }
}
```

PHPUnit: The DbUnit Extension

```
<dataset>
  <account account_number="159349036496204 86" balance="100.00" />
  <account account_number="159364872302150 67" balance="1216.00" />
  <account account_number="123486123572361 85" balance="89.00" />
  <account account_number="159364872302150 67" balance="1216.00" />
  <account account_number="123456789123456 78" balance="0.00" />
</dataset>
```

Avoid testing against MySQL 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 ⇒ No inter-process communication
  - In-Memory Databases ⇒ No Disk I/O

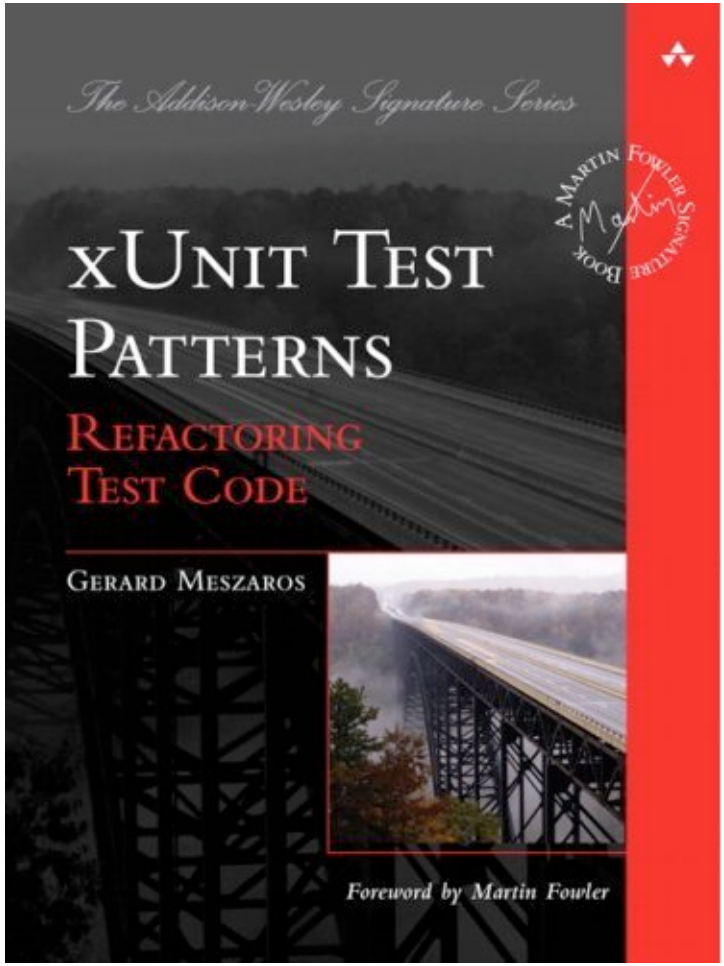
	User	System	CPU	Total
PDO / MySQL	3.95s	0.87s	40%	12.046s
PDO / SQLite (file)	5.01s	1.54s	63%	10.359s
PDO / SQLite (memory)	3.16s	0.68s	99%	3.849s

Stubs and Mocks

- Tests that only test one thing are more informative than tests where failure can come from many sources
- How can you isolate your tests from external influences?
- Simply put, by replacing the expensive, messy, unreliable, slow, complicated resources with stubs that are automatically generated for the purpose of your tests

Stubs and Mocks

- 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



Stubs and Mocks

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

class ObserverTest extends PHPUnit_Framework_TestCase {
    public function testUpdateIsCalledOnce() {









    }
}
?>
```

Stubs and Mocks

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

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









    }
}
?>
```

Stubs and Mocks

```php
<?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'));




    }
}
?>
```

Stubs and Mocks

```php
<?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();
    }
}
?>
```

Mocking the Database

- Ideally the *business logic* and the *persistence logic* of a domain entity such as the bank account is (modelled and) implemented in separate classes
  - Separation of Concerns
- This allows for mocking either object to test the other in isolation
  - But you still want to test the *real interaction* between the two, too
- Future Work
  - Leverage MySQL Proxy to mock the database

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.