Unit testing in CakePHP

Making bullet resistant code.

Goals for next hour

- If you are not familiar with Unit Testing, introduce you to the concepts and practices of Unit testing.
- If you are familiar with Unit Testing, show how it works in CakePHP.
- Show you how to make and run tests and some extra spice on the side.

So who is this guy?

- 4 Years PHP experience.
- 1.5 Years CakePHP experience.
- Member of Core development team since May 2008.
- Developer: Cake Development Corporation.
- Blogger: http://mark-story.com.
- Designer of all kinds of things.

Co-Authored by:

- Tim Koschuetzki from Berlin, Germany.
- Alias “DarkAngelBGE”.
- Blogger: debuggable.com.
- Entrepreneur: Debuggable Ltd.
- 5 years PHP experience.
- 1.5 years CakePHP experience

Unit Testing?

- Unit tests are automated procedure that ensure the software you write, works the way you think it does.

- Unit = the smallest part of your application
- In procedural programming these would be your functions
- In OOP programming these are your Object's methods.

Testing is like making coffee

- Coffee grinds + water (inputs)
- Coffee making magic (function)
- Hot Coffee (output)
- Our tests would ensure that the coffee is black and hot and delicious.

Testing is an automated process

- Tests can be run repeatedly and always work the same.
- Tests shouldn't require additional configuration.
- Tests can be grouped or run individually.
- Green is your friend, Red is not.

Making assertions

```php
function testAssertions() {
    $this->assertNull($x);              //Pass if $x is null
    $this->assertTrue($x);              //Pass if $x is true
    $this->assertFalse($x);             //Pass if $x is false

    $this->assertEqual($x, $y);         //Pass if $x == $y
    $this->assertIdentical($x, $y);     //Pass if $x === $y
    $this->assertPattern($p, $x);       //Pass if $p matches $x
    $this->assertWantedPattern($p, $x); //Pass if $p matches $x
    $this->assertError($x);             //Pass if there is an error and message is the same.
}
```

Making assertions

```
function testAssertions() {
    $this->assertNotNull($x)                //Pass if $x is not null
    $this->assertNoPattern($p, $x);         //Pass if $p does not match $x
    $this->assertNoUnwantedPattern($p, $x)  //Pass if $ does not match $x
    $this->assertNotEqual($x, $y);          //Pass if $x != $y
    $this->assertNotIdentical($x, $y);      //Pass if $x !== $y
    $this->assertNoErrors();                //Pass if no error was raised

    $this->assertReference($x, $y);         //Pass if $x =& $y
    $this->assertCopy($x, $y);              //Pass if $x and $y are the same
    $this->assertIsA($x, $t);               //Pass if $x is a $t
}
```

SimpleTest example

- We have a function that censors text and replaces the banned words with ****

`removeBannedWords($text, $banned)`

- `$text` – Text to censor
- `$banned` – array of words to ban.

SimpleTest example

```php
<?php
    require 'vendors/simpletest/autorun.php';

    function removeBannedWords($text, $banned) {
        foreach ($banned as $word) {
            $regExp = '/' . preg_quote($word, '/') . '/';
            preg_replace($regExp, '****', $text);
        }
        return $text;
    }

    class BannedWordsTestCase extends UnitTestCase {
        function testWordBanning() {
            $text = 'This is bananas, and I really hate apples! They are so tasty!';
            $banned = array('apples', 'bananas');
            $result = removeBannedWords($text, $banned);
            $expected = 'This is ****, and I really hate ****! They are so tasty!';
            $this->assertEqual($result, $expected);
        }
    }
?>
```

SimpleTest example

```php
<?php
    require 'vendors/simpletest/autorun.php';

    function removeBannedWords($text, $banned) {
        foreach ($banned as $word) {
            $regExp = '/' . preg_quote($word, '/') . '/';
            preg_replace($regExp, '****', $text);
        }
        return $text;
    }

    class BannedWordsTestCase extends UnitTestCase {
        function testWordBanning() {
            $text = 'This is bananas, and I really hate apples! They are so tasty!';
            $banned = array('apples', 'bananas');
            $result = removeBannedWords($text, $banned);
            $expected = 'This is ****, and I really hate ****! They are so tasty!';
            $this->assertEqual($result, $expected);
        }
    }
?>
```

SimpleTest example

```php
<?php
    require 'vendors/simpletest/autorun.php';

    function removeBannedWords($text, $banned) {
        foreach ($banned as $word) {
            $regExp = '/' . preg_quote($word, '/') . '/';
            preg_replace($regExp, '****', $text);
        }
        return $text;
    }

    class BannedWordsTestCase extends UnitTestCase {
        function testWordBanning() {
            $text = 'This is bananas, and I really hate apples! They are so tasty!';
            $banned = array('apples', 'bananas');
            $result = removeBannedWords($text, $banned);
            $expected = 'This is ****, and I really hate ****! They are so tasty!';
            $this->assertEqual($result, $expected);
        }
    }
?>
```

SimpleTest example

```php
<?php
    require 'vendors/simpletest/autorun.php';

    function removeBannedWords($text, $banned) {
        foreach ($banned as $word) {
            $regExp = '/' . preg_quote($word, '/') . '/';
            preg_replace($regExp, '****', $text);
        }
        return $text;
    }

    class BannedWordsTestCase extends UnitTestCase {
        function testWordBanning() {
            $text = 'This is bananas, and I really hate apples! They are so tasty!';
            $banned = array('apples', 'bananas');
            $result = removeBannedWords($text, $banned);
            $expected = 'This is ****, and I really hate ****! They are so tasty!';
            $this->assertEqual($result, $expected);
        }
    }
?>
```

SimpleTest example

```php
<?php
    require 'vendors/simpletest/autorun.php';

    function removeBannedWords($text, $banned) {
        foreach ($banned as $word) {
            $regExp = '/' . preg_quote($word, '/') . '/';
            preg_replace($regExp, '****', $text);
        }
        return $text;
    }

    class BannedWordsTestCase extends UnitTestCase {
        function testWordBanning() {
            $text = 'This is bananas, and I really hate apples! They are so tasty!';
            $banned = array('apples', 'bananas');
            $result = removeBannedWords($text, $banned);
            $expected = 'This is ****, and I really hate ****! They are so tasty!';
            $this->assertEqual($result, $expected);
        }
    }
?>
```

SimpleTest example

```php
<?php
    require 'vendors/simpletest/autorun.php';

    function removeBannedWords($text, $banned) {
        foreach ($banned as $word) {
            $regExp = '/' . preg_quote($word, '/') . '/';
            preg_replace($regExp, '****', $text);
        }
        return $text;
    }

    class BannedWordsTestCase extends UnitTestCase {
        function testWordBanning() {
            $text = 'This is bananas, and I really hate apples! They are so tasty!';
            $banned = array('apples', 'bananas');
            $result = removeBannedWords($text, $banned);
            $expected = 'This is ****, and I really hate ****! They are so tasty!';
            $this->assertEqual($result, $expected);
        }
    }
?>
```

Fail?

unittest.php

Fail: BannedWordsTestCase -> testWordBanning -> Equal expectation fails at character 8 with [This is bananas, and I really hate apples! They are so tasty!] and [This is ****, and I really hate ****! They are so tasty!] at [/Users/markstory/Sites/presentation_scripts /unittest.php line 18]

1/1 test cases complete: 0 passes, 1 fails and 0 exceptions.

- Red bar means that a test has failed
- When a test fails, a message informs you of the expectation that was not met, and on which line that expectation can be found.
- Red bars appear anytime there are any errors / uncaught exceptions or failing tests.

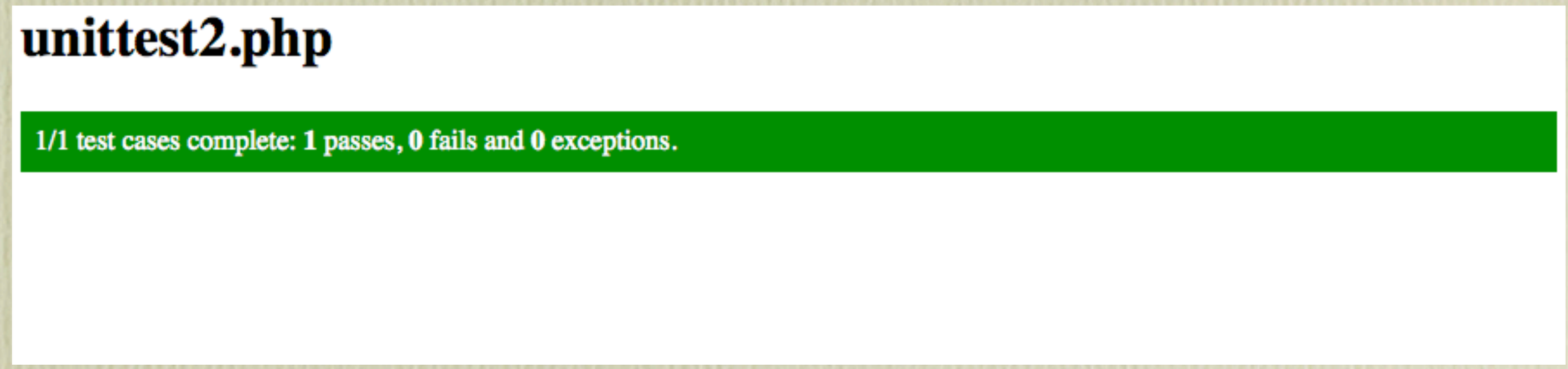
Is our function broken?

```php
<?php
    require 'vendors/simpletest/autorun.php';

    function removeBannedWords($text, $banned) {
        foreach ($banned as $word) {
            $regExp = '/' . preg_quote($word, '/') . '/';
            $text = preg_replace($regExp, '****', $text);
        }
        return $text;
    }

    class BannedWordsTestCase extends UnitTestCase {
        function testWordBanning() {
            $text = 'This is bananas, and I really hate apples! They are so tasty!';
            $banned = array('apples', 'bananas');
            $result = removeBannedWords($text, $banned);
            $expected = 'This is ****, and I really hate ****! They are so tasty!';
            $this->assertEqual($result, $expected);
        }
    }
?>
```

Pass-tastic!



- Green bar means everything is working as we planned.

Benefits of Unit Testing

- Know when a changeset breaks the expected behavior.
- Creates living documentation.
- Bug fixing is much easier, faster and less prone to creating new bugs.
- Faster than browser testing.
- Easy to repeat.

Benefits of Unit Testing

- Reduces developer hesitation to refactor code.
- Catch bugs earlier.
- Increase developer confidence.
- Reduce the need to manually test things.

Limitations of Unit Testing

- Time: Time is money and tests take time to write.
- Motivation: Testing is anti-lazy. Sometimes tests for Objects are longer than object code.
- Maintenance: More code to maintain. For example CakePHP code base (1/3 code, 2/3 tests)

Limitations of Unit Testing

- Doesn't catch programmer errors, for example reading the specs wrong.
- Only shows the presence of errors not the absence of them.
- Only catches errors you have tests for. If a situation isn't covered, testing will not catch any problems with it.

So why write Unit Tests?

- *Increase developer confidence* - You can be certain that you have done what was required.
- *Increase overall quality* - You have additional confidence that your project is functioning properly
- *Find problems earlier* - I personally have found many problems with implementations while I write tests.

So why write Unit Tests?

- *Reduce time lost click testing* - Click testing is slow and tedious. Much of this time can be replaced with automated unit tests.
- *Reduce 'voodoo' factor* - Since you have proven uses for your code you are less afraid to rebuild that ugly function and make it more useful.

Unit Testing in CakePHP

Unit Testing in CakePHP

- Download and install SimpleTest
  - http://simpletest.org
  - Place in `app/vendors` or `cake/vendors`
- Set `debug >= 1`
- Add `$test` Database connection.
- Visit `path/to/app/test.php`

Test Cases & Test Groups

Test Cases

- Contain tests for individual classes.

- Collections of test methods.

- ex. Menu Component, User Model

Test Groups

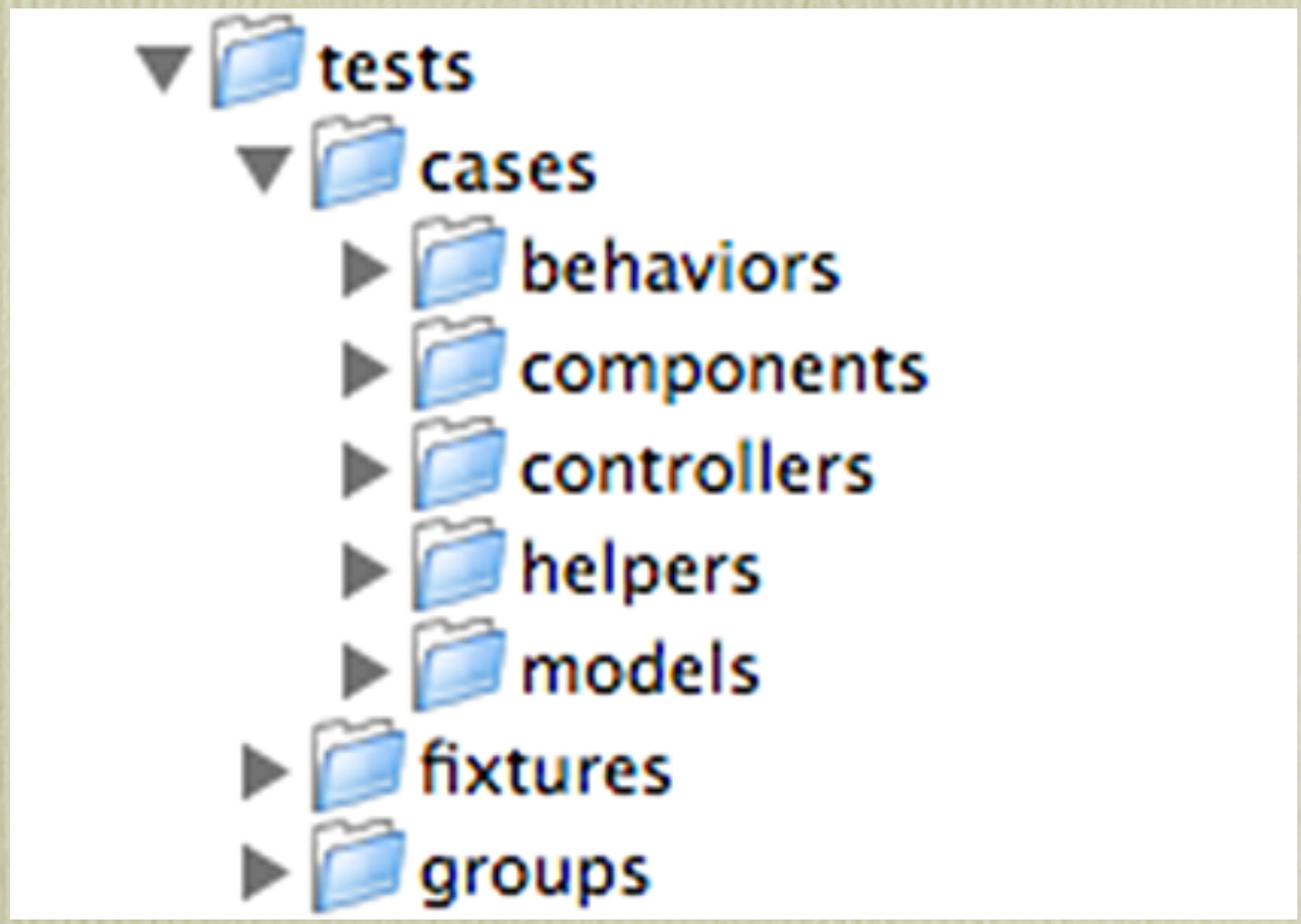
- Way to run multiple tests at once.

- Allow you to test a subsystem or collection of objects

- Free group test called 'All tests'

- ex. All models, Access Control system, payment processing

File layout



- in `app/tests`
- Different case directories for different objects.

Filename conventions

Test Cases

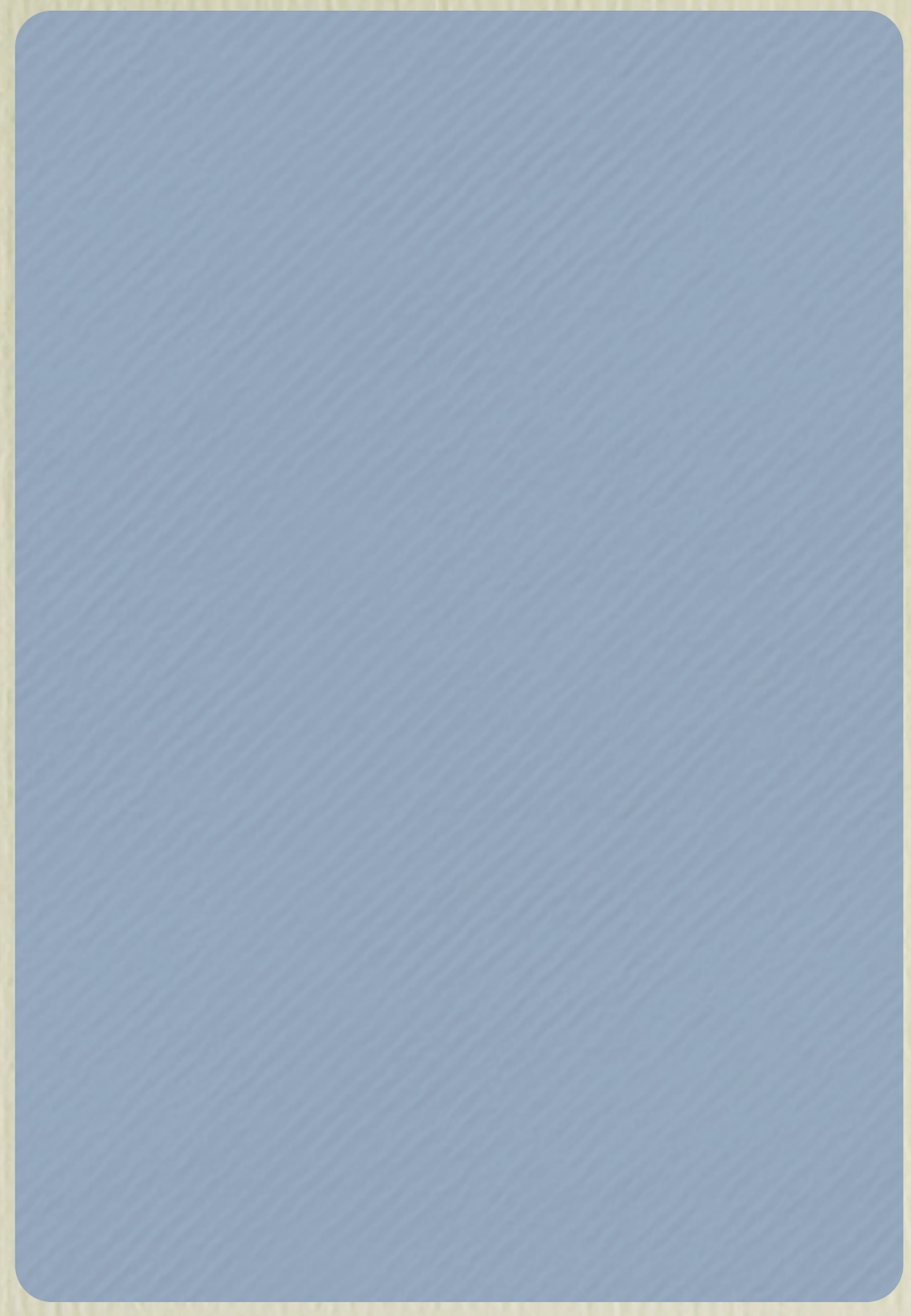
- Underscored and lowercase
- users_controller = `users_controller.test.php`
- post = `post.test.php`

Group Tests

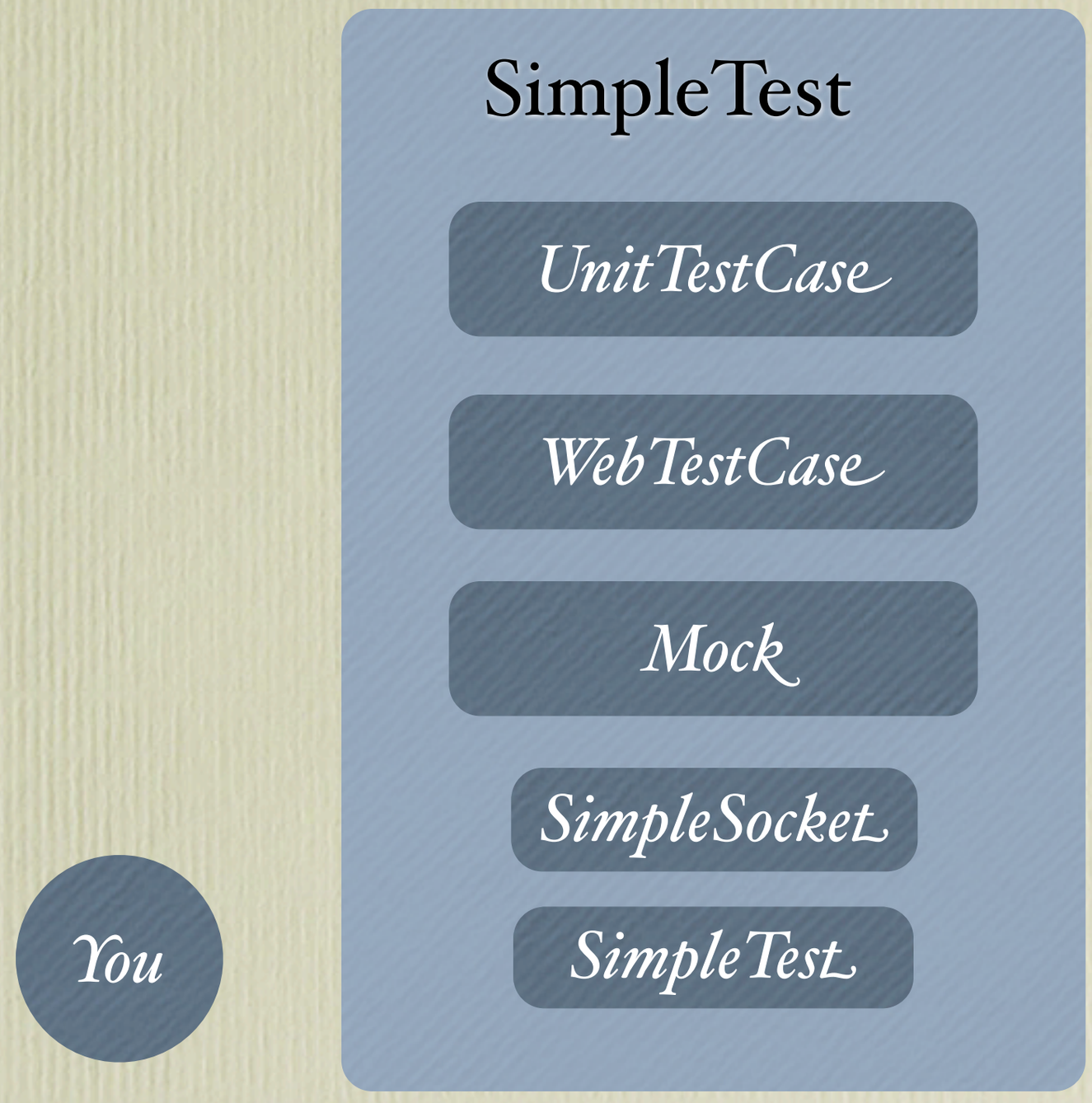
- Underscored and lowercase but any name
- `models.group.php`
- `payment_processing.group.php`

Classes involved

Classes involved



Classes involved



Classes involved

SimpleTest

UnitTestCase

WebTestCase

Mock

SimpleSocket

SimpleTest

You

Classes involved

CakePHP Test Suite

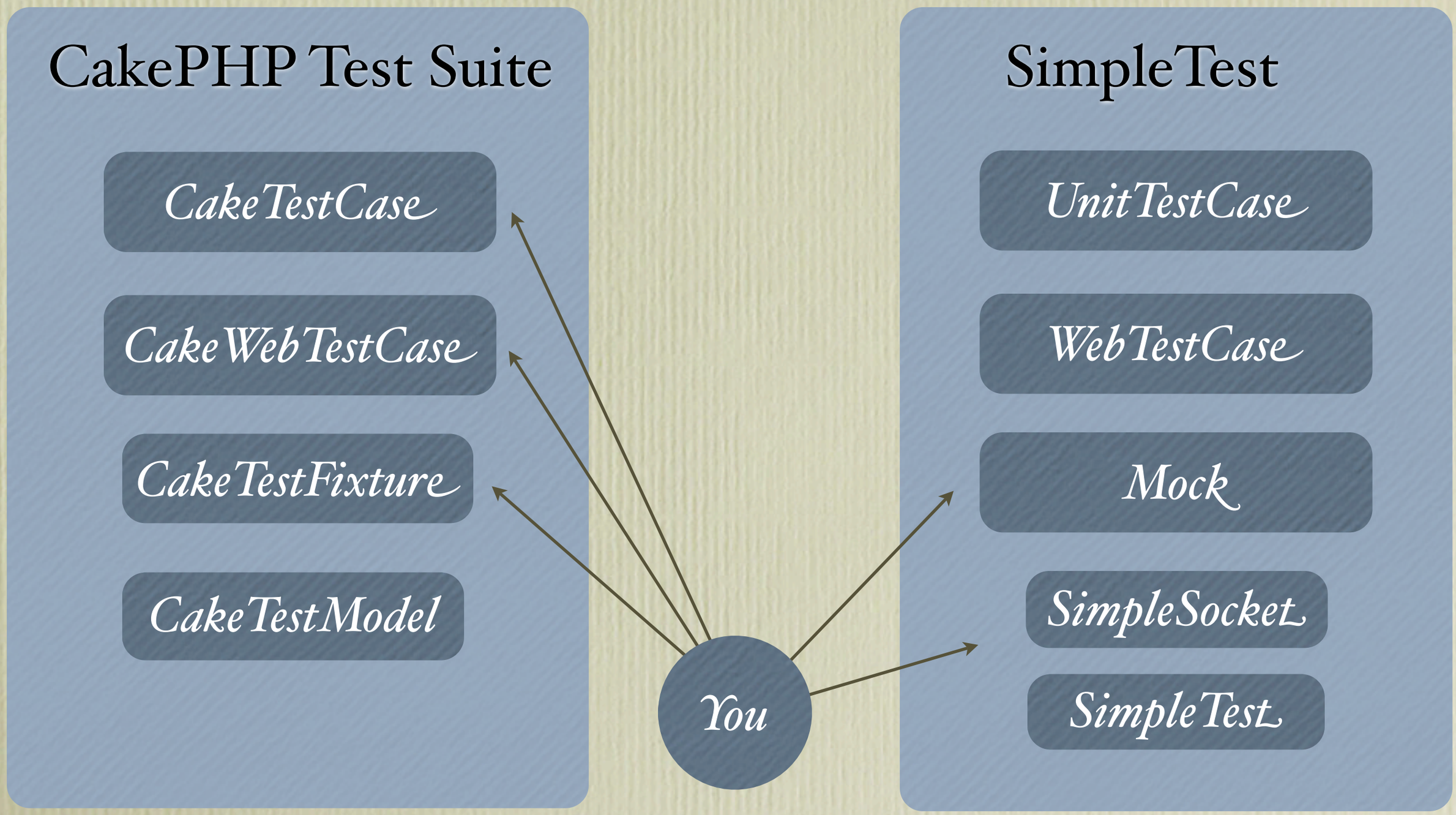
- *CakeTestCase*
- *CakeWebTestCase*
- *CakeTestFixture*
- *CakeTestModel*

You

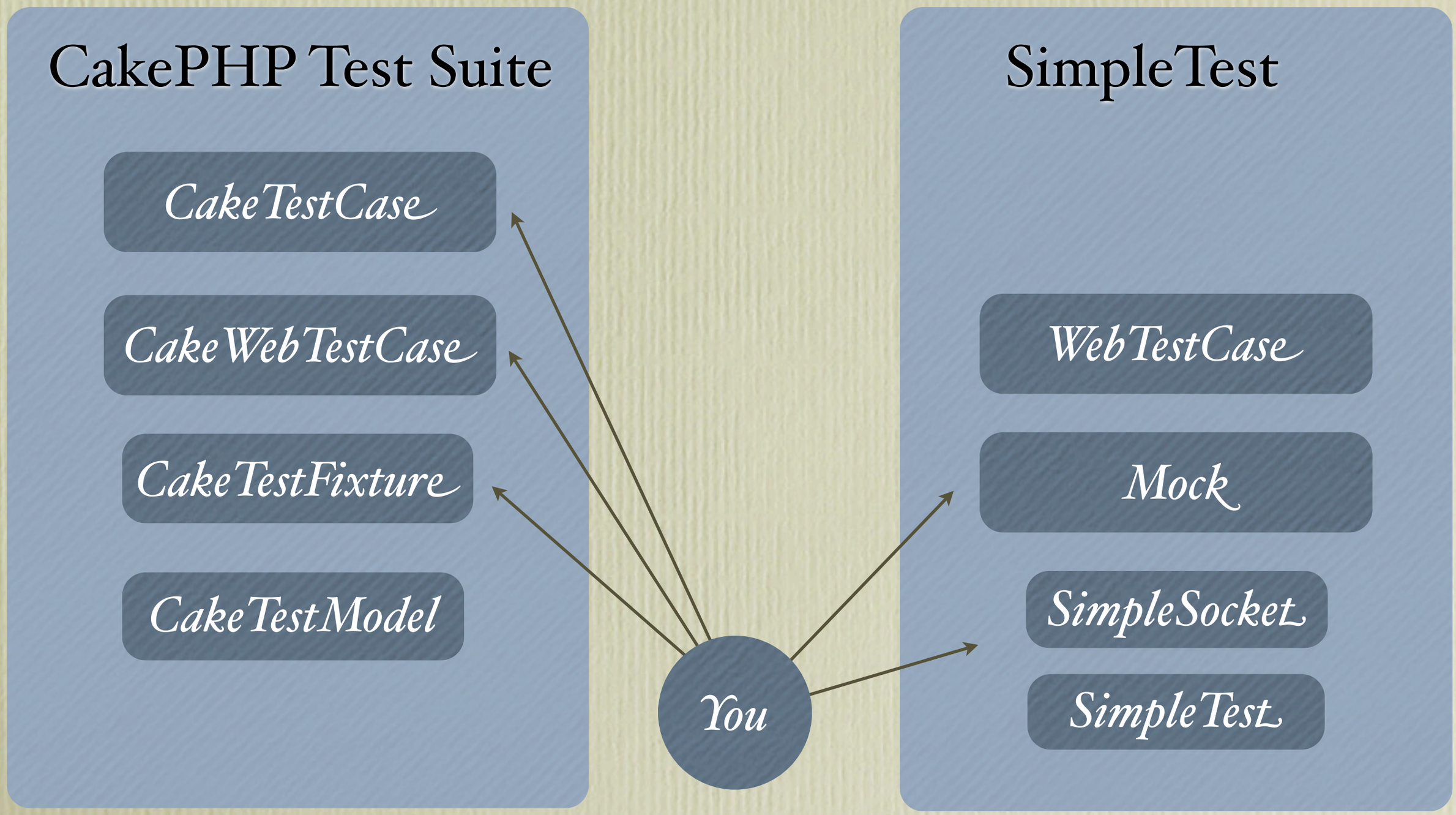
SimpleTest

- *UnitTestCase*
- *WebTestCase*
- *Mock*
- *SimpleSocket*
- *SimpleTest*

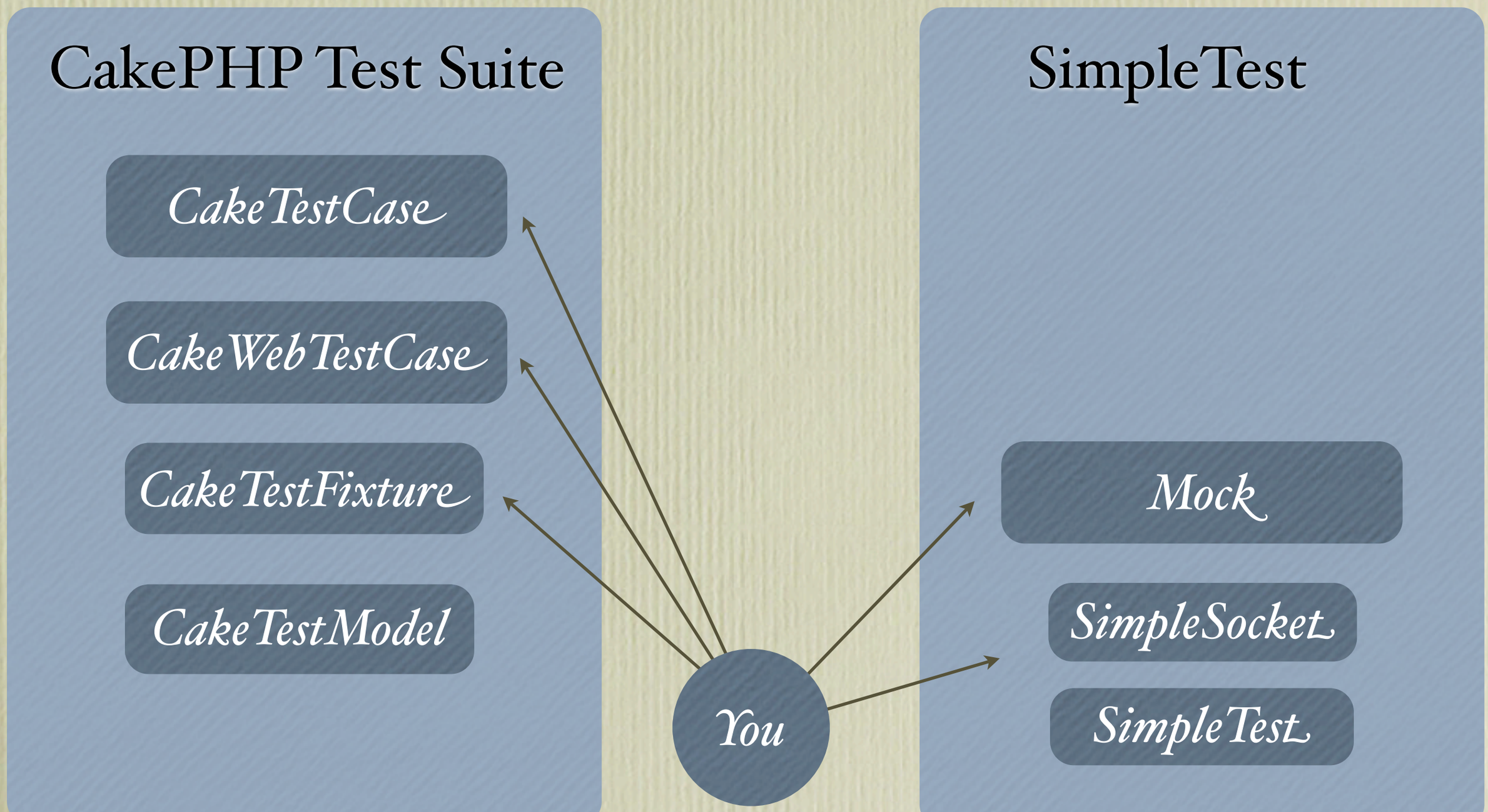
Classes involved



Classes involved



Classes involved



CakeTestCase

- Test case class that all App and Core test cases should extend.
- `CakeTestCase` adds several useful features to `UnitTestCase`

CakeTestCase Methods

- `before($method)`
- Announces the start of a *test method*.
- `after($method)`
- Announces the end of a *test method*.
- `start()`
- First method called in a *test case*.
- `end()`
- Last method called in a *test case*.

CakeTestCase Methods

- `startTest($method)`
- Called just before a *test method* is executed.
- `endTest($method)`
- Called just after a *test method* has completed.

CakeTestCase Methods

- `startCase()`
- Called before a *test case* is started.
- `endCase()`
- Called after a *test case* has run.

CakeTestCase Methods

```
testAction($url, $params);
```

- Run a controller action and get the results.
  - *result*: Whatever the action returns. Also simulates a requestAction()
  - *view*: The rendered view, without the layout
  - *contents*: The rendered view, within the layout.
  - *vars*: the view vars

CakeTestCase Methods

`assertTags($result, $expected)`

- Allows you to compare HTML/XML snippets to an array of expected values

`getTests()`

- Useful when debugging a specific case. Check the implementation in CakeTestCase before overriding.

CakeTestCase Method Order

```php
<?php
class ExamsControllerTestCase extends CakeTestCase {
    function startCase() {
        echo '<h4>startCase()</h4>';
    }

    function endCase() {
        echo '<h4>endCase()</h4>';
    }

    function before($method) {
        parent::before($method);
        echo '<h4>before ' . $method . '</h4>';
    }

    function after($method) {
        parent::after($method);
        echo '<h4>after '. $method . '</h4><br>';
    }

    function startTest($method) {
        echo '<h4>Starting method ' . $method . '</h4>';
    }

    function endTest($method) {
        echo '<h4>Ending method ' . $method . '</h4>';
    }

    function testStudyMethod() {
        echo '<h4>In testStudyMethod()</h4>';
        $this->assertTrue(true);
    }

    function testDoExam() {
        echo '<h4>In testDoExam()</h4>';
        $this->assertTrue(true);
    }
}
?>
```

CakeTestCase Method Order

```
before start
after start

before startCase
startCase()
after startCase

Starting method testStudyMethod
before testStudyMethod
In testStudyMethod()
Ending method testStudyMethod
after testStudyMethod

Starting method testDoExam
before testDoExam
In testDoExam()
Ending method testDoExam
after testDoExam

before endCase
endCase()
after endCase

before end
after end
```

Avoid `setUp()` & `tearDown()`

- When testing Models or using fixtures.
- Use `startTest()` and `endTest()` instead.
- `setUp()` & `tearDown()` run before `start()` which is where the fixtures are first created.
- Using `setUp()` & `tearDown()` will cause errors when testing models.

Fixtures - Predictable data

- Fixtures create test database tables and records, which you run tests against.
- A predictable and known application state is vital to testing.
- Helps ensures that inputs will always produce the same outputs given the same functionality of methods.
- Fixtures will use the `$test` connection if available. If not they will use `$default` with a `test_suite` prefix.

What is a Fixture?

- Fixture provides table schema and records to fill the table.
- Each table is a separate fixture.
- Schema and Records can be inside the fixture or imported from development data.

Life of a Fixture

- Fixture tables are created in `CakeTestCase::start()`
- Before each test method, fixture tables are populated with records in fixture.
- After each test method, fixture tables are truncated.
- Fixture tables are dropped in `CakeTestCase::end()`

Fixture field definitions

`type`:

- string (maps to VARCHAR)
- text (maps to TEXT)
- integer (maps to INT)
- float (maps to FLOAT)
- datetime (maps to DATETIME)

Fixture field definitions

- key: set to primary to make the field `AUTO_INCREMENT`, and `PRIMARY KEY` for the table.
- `length`: set to the specific length the field should take.
- `null`: set to either true (to allow `NULL`s) or false (to disallow `NULL`s)
- `default`: default value the field takes.

Sample Fixture

```php
<?php
class NodeFixture extends CakeTestFixture {
    var $name = 'Node';

    var $fields = array(
        'id' => array('type' => 'integer', 'key' => 'primary'),
        'user_id' => array('type' => 'integer'),
        'type' => array('type' => 'string', 'length' => 255, 'null' => false),
        'title' => array('type' => 'string', 'length' => 255, 'null' => false),
        'slug' => array('type' => 'string', 'length' => 255, 'null' => false),
        'published' => array('type' => 'integer', 'default' => '0', 'null' => false),
        'promoted' => array('type' => 'integer', 'default' => '0', 'null' => false),
        'comments_allowed' => array('type' => 'integer', 'default' => '0', 'null' => false),
        'views' => array('type' => 'integer', 'default' => '0', 'null' => false),
        'comment_count' => array('type' => 'integer', 'default' => '0', 'null' => false),
        'created' => 'datetime',
        'modified' => 'datetime'
    );

    var $records = array(
            array(  'id' => 1,
                    'user_id' => 1,
                    'type' => 'Post',
                    'title' => 'First Article',
                    'slug' => 'first-article',
                    'published' => '1',
                    'promoted' => '0',
                    'comments_allowed' => 1,
                    'views' => 5,
                    'comment_count' => 1,
                    'created' => '2007-02-18 10:39:23',
                    'modified' => '2007-03-18 10:41:31'),

        );
}
```

Sample Fixture

```php
<?php
class NodeFixture extends CakeTestFixture {
    var $name = 'Node';

    var $fields = array(
        'id' => array('type' => 'integer', 'key' => 'primary'),
        'user_id' => array('type' => 'integer'),
        'type' => array('type' => 'string', 'length' => 255, 'null' => false),
        'title' => array('type' => 'string', 'length' => 255, 'null' => false),
        'slug' => array('type' => 'string', 'length' => 255, 'null' => false),
        'published' => array('type' => 'integer', 'default' => '0', 'null' => false),
        'promoted' => array('type' => 'integer', 'default' => '0', 'null' => false),
        'comments_allowed' => array('type' => 'integer', 'default' => '0', 'null' => false),
        'views' => array('type' => 'integer', 'default' => '0', 'null' => false),
        'comment_count' => array('type' => 'integer', 'default' => '0', 'null' => false),
        'created' => 'datetime',
        'modified' => 'datetime'
    );

    var $records = array(
            array(  'id' => 1,
                    'user_id' => 1,
                    'type' => 'Post',
                    'title' => 'First Article',
                    'slug' => 'first-article',
                    'published' => '1',
                    'promoted' => '0',
                    'comments_allowed' => 1,
                    'views' => 5,
                    'comment_count' => 1,
                    'created' => '2007-02-18 10:39:23',
                    'modified' => '2007-03-18 10:41:31'),

        );
}
```

Sample Fixture

```php
<?php
class NodeFixture extends CakeTestFixture {
    var $name = 'Node';

    var $fields = array(
        'id' => array('type' => 'integer', 'key' => 'primary'),
        'user_id' => array('type' => 'integer'),
        'type' => array('type' => 'string', 'length' => 255, 'null' => false),
        'title' => array('type' => 'string', 'length' => 255, 'null' => false),
        'slug' => array('type' => 'string', 'length' => 255, 'null' => false),
        'published' => array('type' => 'integer', 'default' => '0', 'null' => false),
        'promoted' => array('type' => 'integer', 'default' => '0', 'null' => false),
        'comments_allowed' => array('type' => 'integer', 'default' => '0', 'null' => false),
        'views' => array('type' => 'integer', 'default' => '0', 'null' => false),
        'comment_count' => array('type' => 'integer', 'default' => '0', 'null' => false),
        'created' => 'datetime',
        'modified' => 'datetime'
    );

    var $records = array(
            array(  'id' => 1,
                    'user_id' => 1,
                    'type' => 'Post',
                    'title' => 'First Article',
                    'slug' => 'first-article',
                    'published' => '1',
                    'promoted' => '0',
                    'comments_allowed' => 1,
                    'views' => 5,
                    'comment_count' => 1,
                    'created' => '2007-02-18 10:39:23',
                    'modified' => '2007-03-18 10:41:31'),

        );
}
```

Import fixtures

- You can import the records of your current database as well.
- Can choose to only import table structure, but define records yourself. Or import both.
- Import by model or table and connection.

Importing Fixtures

```php
<?php
class NodeFixture extends CakeTestFixture {
    var $name = 'Node';

    var $import = array('table' => 'nodes', 'records' => true);

}
```

Import from a table and duplicate the records.

Importing Fixtures

Importing Fixtures

```php
<?php
class NodeFixture extends CakeTestFixture {
    var $name = 'Node';

    var $import = array('model' => 'Node', 'records' => true);
}
```

Import from a model, and duplicate the records.

Importing Fixtures

Importing Fixtures

```php
<?php
class NodeFixture extends CakeTestFixture {
    var $name = 'Node';

    var $import = array('model' => 'Node', 'connection' => 'shadow', 'records' => true);

}
```

Import model with shadow connection.

To import or not?

- Importing records ties your tests to your development data. Can make for lots of broken tests.
- Writing many records out in fixture can take extra time.

Putting it all together

Testing Models

Example Model

```php
<?php
class Node extends AppModel {
    var $name = 'Node';

/**
 * Update views.
 * Increment Node::views
 *
 * @return bool
 **/
    public function updateViews($id, $oldValue = null){
        $oldId = $this->id;
        $this->id = $id;
        if(!$oldValue){
            $oldValue = $this->field('views', array('Node.id' => $id));
        }
        $ret = $this->saveField('views', ($oldValue + 1) );
        $this->id = $oldId;
        return $ret;
    }

}
```

Test Case setup

```php
<?php
App::import('Model', 'Node');

class NodeTestCase extends CakeTestCase {
    var $fixtures = array(
        'app.node', 'app.user', 'app.post', 'app.group',
        'app.image', 'app.download', 'app.comment', 'app.tag',
        'app.tag_category', 'app.nodes_tag'
    );


    function startTest() {
        $this->Node =& ClassRegistry::init('Node');
    }

    function testUpdateViews() {
        $result = $this->Node->updateViews(1);
        $this->assertTrue($result);

        $this->Node->id = 1;
        $result = $this->Node->field('views');
        $expected = 6;
        $this->assertEqual($result, $expected);

        $expected = 10;
        $this->Node->updateViews(2, 9);
        $this->Node->id = 2;
        $result = $this->Node->field('views');
        $this->assertEqual($result, $expected);
    }
```

Test Case setup

```php
<?php
App::import('Model', 'Node');

class NodeTestCase extends CakeTestCase {
    var $fixtures = array(
        'app.node', 'app.user', 'app.post', 'app.group',
        'app.image', 'app.download', 'app.comment', 'app.tag',
        'app.tag_category', 'app.nodes_tag'
    );


    function startTest() {
        $this->Node =& ClassRegistry::init('Node');
    }

    function testUpdateViews() {
        $result = $this->Node->updateViews(1);
        $this->assertTrue($result);

        $this->Node->id = 1;
        $result = $this->Node->field('views');
        $expected = 6;
        $this->assertEqual($result, $expected);

        $expected = 10;
        $this->Node->updateViews(2, 9);
        $this->Node->id = 2;
        $result = $this->Node->field('views');
        $this->assertEqual($result, $expected);
    }
```

Test Case setup

```php
<?php
App::import('Model', 'Node');

class NodeTestCase extends CakeTestCase {
    var $fixtures = array(
        'app.node', 'app.user', 'app.post', 'app.group',
        'app.image', 'app.download', 'app.comment', 'app.tag',
        'app.tag_category', 'app.nodes_tag'
    );


    function startTest() {
        $this->Node =& ClassRegistry::init('Node');
    }

    function testUpdateViews() {
        $result = $this->Node->updateViews(1);
        $this->assertTrue($result);

        $this->Node->id = 1;
        $result = $this->Node->field('views');
        $expected = 6;
        $this->assertEqual($result, $expected);

        $expected = 10;
        $this->Node->updateViews(2, 9);
        $this->Node->id = 2;
        $result = $this->Node->field('views');
        $this->assertEqual($result, $expected);
    }
```

Test Case setup

```php
<?php
App::import('Model', 'Node');

class NodeTestCase extends CakeTestCase {
    var $fixtures = array(
        'app.node', 'app.user', 'app.post', 'app.group',
        'app.image', 'app.download', 'app.comment', 'app.tag',
        'app.tag_category', 'app.nodes_tag'
    );


    function startTest() {
        $this->Node =& ClassRegistry::init('Node');
    }

    function testUpdateViews() {
        $result = $this->Node->updateViews(1);
        $this->assertTrue($result);

        $this->Node->id = 1;
        $result = $this->Node->field('views');
        $expected = 6;
        $this->assertEqual($result, $expected);

        $expected = 10;
        $this->Node->updateViews(2, 9);
        $this->Node->id = 2;
        $result = $this->Node->field('views');
        $this->assertEqual($result, $expected);
    }
```

Test Case setup

```php
<?php
App::import('Model', 'Node');

class NodeTestCase extends CakeTestCase {
    var $fixtures = array(
        'app.node', 'app.user', 'app.post', 'app.group',
        'app.image', 'app.download', 'app.comment', 'app.tag',
        'app.tag_category', 'app.nodes_tag'
    );


    function startTest() {
        $this->Node =& ClassRegistry::init('Node');
    }

    function testUpdateViews() {
        $result = $this->Node->updateViews(1);
        $this->assertTrue($result);

        $this->Node->id = 1;
        $result = $this->Node->field('views');
        $expected = 6;
        $this->assertEqual($result, $expected);

        $expected = 10;
        $this->Node->updateViews(2, 9);
        $this->Node->id = 2;
        $result = $this->Node->field('views');
        $this->assertEqual($result, $expected);
    }
```

Test Case setup

```php
<?php
App::import('Model', 'Node');

class NodeTestCase extends CakeTestCase {
    var $fixtures = array(
        'app.node', 'app.user', 'app.post', 'app.group',
        'app.image', 'app.download', 'app.comment', 'app.tag',
        'app.tag_category', 'app.nodes_tag'
    );


    function startTest() {
        $this->Node =& ClassRegistry::init('Node');
    }

    function testUpdateViews() {
        $result = $this->Node->updateViews(1);
        $this->assertTrue($result);

        $this->Node->id = 1;
        $result = $this->Node->field('views');
        $expected = 6;
        $this->assertEqual($result, $expected);

        $expected = 10;
        $this->Node->updateViews(2, 9);
        $this->Node->id = 2;
        $result = $this->Node->field('views');
        $this->assertEqual($result, $expected);
    }
```

Test Case setup

```php
<?php
App::import('Model', 'Node');

class NodeTestCase extends CakeTestCase {
    var $fixtures = array(
        'app.node', 'app.user', 'app.post', 'app.group',
        'app.image', 'app.download', 'app.comment', 'app.tag',
        'app.tag_category', 'app.nodes_tag'
    );


    function startTest() {
        $this->Node =& ClassRegistry::init('Node');
    }

    function testUpdateViews() {
        $result = $this->Node->updateViews(1);
        $this->assertTrue($result);

        $this->Node->id = 1;
        $result = $this->Node->field('views');
        $expected = 6;
        $this->assertEqual($result, $expected);

        $expected = 10;
        $this->Node->updateViews(2, 9);
        $this->Node->id = 2;
        $result = $this->Node->field('views');
        $this->assertEqual($result, $expected);
    }
```

Model testing tips

- Using `ClassRegistry::init()` and the benefits over `new Widget()`
  - `useDbConfig` changed, and all relations built properly as well!
  - No need for test model classes in most instances.
- Isolate tests with `ClassRegistry::flush()`
  - Object states are persisted between test cases in the `ClassRegistry`.
  - `ClassRegistry::flush()` tosses those objects in the bin. giving you a clean slate for each test.

The importance of fixtures

- Fixtures allow you to avoid having to mock out your database connections.
- Fixtures allow for known data, making your tests invulnerable to application data anomalies.
- Easily create data to cover any cases you want.

Unit tests vs. Functional tests

- In pure unit testing everything not being tested is mocked.
- The example given and all the core model tests are better classified as functional tests.
- Functional tests, test a slice of an application.
- These tests run slower, but give a more complete test.

Testing Helpers

and using assertTags()

Asserting Helper output

- Testing HTML is a pain. But must be done.
- Can use `assertPattern()` with insane regular expressions.
- However, you will curse regular expressions as they quickly become impossible to read and understand.

Regular Expression vs. assertTags()

```
    function testInputRegularExp() {
        $result = $this->Form->input('Contact.password');
        $pattern = '#<div\s*class\="input\s*password">\s*<label
for\="ContactPassword">Password</label>\s*<input\s*type\="password"\s*name\="data\[Contact\]\[password\]"\s*value\=""\sid\="ContactPasswo
rd"\s*/></div>#';
        $this->assertPattern($pattern, $result);
    }
```

Regular Expression vs. assertTags()

```
function testInput() {
    $result = $this->Form->input('Contact.password');
    $expected = array(
        'div' => array('class' => 'input password'),
        'label' => array('for' => 'ContactPassword'),
        'Password',
        '/label',
        array('input' => array('type' => 'password', 'name' => 'data[Contact][password]', 'value' => '', 'id' => 'ContactPassword')),
        '/div'
    );
    $this->assertTags($result, $expected);
}
```

`assertTags()` tips

- Assertion must be for entirety of output.
- Order of attributes doesn't matter
- Contents of attributes are case and whitespace sensitive.
- Watch your keys, wrap multiple keys in `array()`
- Use `true` as third parameter for extra debug information and help.

Example Helper test

```php
<?php
App::import('Helper', array('Html', 'Form'));
App::import('Core', 'View');

class FormHelperTestCase extends CakeTestCase {

    function startTest() {
        $this->Form =& new FormHelper();
        $this->Form->Html =& new HtmlHelper();
        $this->View =& new View($this->Controller);
    }

    function testInput() {
        $result = $this->Form->input('Contact.password');
        $expected = array(
            'div' => array('class' => 'input password'),
            'label' => array('for' => 'ContactPassword'),
            'Password',
            '/label',
            array('input' => array('type' => 'password', 'name' => 'data[Contact][password]', 'value' => '', 'id' => 'ContactPassword')),
            '/div'
        );
        $this->assertTags($result, $expected);
    }

    function endTest() {
        unset($this->Form, $this->View);
    }
}
?>
```

Example Helper test

```php
<?php
App::import('Helper', array('Html', 'Form'));
App::import('Core', 'View');

class FormHelperTestCase extends CakeTestCase {

    function startTest() {
        $this->Form =& new FormHelper();
        $this->Form->Html =& new HtmlHelper();
        $this->View =& new View($this->Controller);
    }

    function testInput() {
        $result = $this->Form->input('Contact.password');
        $expected = array(
            'div' => array('class' => 'input password'),
            'label' => array('for' => 'ContactPassword'),
            'Password',
            '/label',
            array('input' => array('type' => 'password', 'name' => 'data[Contact][password]', 'value' => '', 'id' => 'ContactPassword')),
            '/div'
        );
        $this->assertTags($result, $expected);
    }

    function endTest() {
        unset($this->Form, $this->View);
    }
}
?>
```

Example Helper test

```php
<?php
App::import('Helper', array('Html', 'Form'));
App::import('Core', 'View');

class FormHelperTestCase extends CakeTestCase {

    function startTest() {
        $this->Form =& new FormHelper();
        $this->Form->Html =& new HtmlHelper();
        $this->View =& new View($this->Controller);
    }

    function testInput() {
        $result = $this->Form->input('Contact.password');
        $expected = array(
            'div' => array('class' => 'input password'),
            'label' => array('for' => 'ContactPassword'),
            'Password',
            '/label',
            array('input' => array('type' => 'password', 'name' => 'data[Contact][password]', 'value' => '', 'id' => 'ContactPassword')),
            '/div'
        );
        $this->assertTags($result, $expected);
    }

    function endTest() {
        unset($this->Form, $this->View);
    }
}
?>
```

Example Helper test

```php
<?php
App::import('Helper', array('Html', 'Form'));
App::import('Core', 'View');

class FormHelperTestCase extends CakeTestCase {

    function startTest() {
        $this->Form =& new FormHelper();
        $this->Form->Html =& new HtmlHelper();
        $this->View =& new View($this->Controller);
    }

    function testInput() {
        $result = $this->Form->input('Contact.password');
        $expected = array(
            'div' => array('class' => 'input password'),
            'label' => array('for' => 'ContactPassword'),
            'Password',
            '/label',
            array('input' => array('type' => 'password', 'name' => 'data[Contact][password]', 'value' => '', 'id' => 'ContactPassword')),
            '/div'
        );
        $this->assertTags($result, $expected);
    }

    function endTest() {
        unset($this->Form, $this->View);
    }
}
?>
```

Example Helper test

```php
<?php
App::import('Helper', array('Html', 'Form'));
App::import('Core', 'View');

class FormHelperTestCase extends CakeTestCase {

    function startTest() {
        $this->Form =& new FormHelper();
        $this->Form->Html =& new HtmlHelper();
        $this->View =& new View($this->Controller);
    }

    function testInput() {
        $result = $this->Form->input('Contact.password');
        $expected = array(
            'div' => array('class' => 'input password'),
            'label' => array('for' => 'ContactPassword'),
            'Password',
            '/label',
            array('input' => array('type' => 'password', 'name' => 'data[Contact][password]', 'value' => '', 'id' => 'ContactPassword')),
            '/div'
        );
        $this->assertTags($result, $expected);
    }

    function endTest() {
        unset($this->Form, $this->View);
    }
}
?>
```

Example Helper test

```php
<?php
App::import('Helper', array('Html', 'Form'));
App::import('Core', 'View');

class FormHelperTestCase extends CakeTestCase {

    function startTest() {
        $this->Form =& new FormHelper();
        $this->Form->Html =& new HtmlHelper();
        $this->View =& new View($this->Controller);
    }

    function testInput() {
        $result = $this->Form->input('Contact.password');
        $expected = array(
            'div' => array('class' => 'input password'),
            'label' => array('for' => 'ContactPassword'),
            'Password',
            '/label',
            array('input' => array('type' => 'password', 'name' => 'data[Contact][password]', 'value' => '', 'id' => 'ContactPassword')),
            '/div'
        );
        $this->assertTags($result, $expected);
    }

    function endTest() {
        unset($this->Form, $this->View);
    }
}
?>
```

Testing Controllers

and using testAction()

testAction() examples

```
function testLogin() {
    $result = $this->testAction('/users/login', array('return' => 'view'));
    debug($result);
}
```

Return a rendered view without a layout

testAction() examples

```
function testLogin() {
    $result = $this->testAction('/users/login', array('return' => 'contents'));
    debug($result);
}
```

Return a rendered view *with* a layout

testAction() examples

```php
function testLogin() {
    $result = $this->testAction('/users/login', array('return' => 'result'));
    debug($result);
}
```

Return a the result returned from action.

testAction() examples

```
function testLogin() {
    $result = $this->testAction('/users/login', array('return' => 'vars', 'fixturize' => true));
    debug($result);
}
```

Return the viewVars and use fixturized tables

testAction()

- Simulate a dispatcher call to a url with specific params.
- Optionally use fixture data.
- Supply an array of url parameters (both GET and POST possible).

Real World Controller Testing

- Often you can't use testAction()
- Need to get your hands dirty.
- Real world test time!

I Mock you!

- Mock objects are fake objects.
- Mocks objects pretend to to be the object they have mocked.
- Mocks can be actors or critics.

Critical Mocking

- Critic mock objects provide critical and introspective methods.
- Critics are generally used to determine if an inner object has had its methods called.

Mock Actors

- Mocks in an actor role are used to feed the test case subject values it may need.
- Often they don't make expectations. But they can.

Making Mocks

- Using `Mock::generate()`
  - You can mock methods that don't exist.
  - This allows you to test methods that haven't been written.

Partial Mocking bird

- In addition to full Mocks you can generate a partial mock.
- Partial Mocks only have some methods mocked. The rest are real methods.
- Partial mocks are great for testing classes that you want partial interaction with.
- Partial mocks can have expectations as well.

Code coverage

- How much is enough?
- 100% is almost impossible for non-trival applications.
- Getting close to 100% can require 3x code.
- A healthier goal is 40%-80%.

Code coverage

CakePHP: the rapid development php framework

CakePHP Test Suite v 1.2.0.0

- App
  - Test Groups
  - Test Cases
- Core
  - Test Groups
  - Test Cases

Individual test case: libs/view/helpers/rss.test.php

1/1 test cases complete: 22 passes, 0 fails and 0 exceptions.

Code Coverage: 83.84%

```
121  * @param  mixed  $content  Content (<item />'s belonging to this channel
122  * @return string An RSS <channel />
123  */
124     function channel($attrib = array(), $elements = array(), $content = null) {
125             $view =& ClassRegistry::getObject('view');
126
127             if (!isset($elements['title']) && !empty($view->pageTitle)) {
128                     $elements['title'] = $view->pageTitle;
129             }
130             if (!isset($elements['link'])) {
131                     $elements['link'] = '/';
132             }
133             if (!isset($elements['description'])) {
134                     $elements['description'] = '';
135             }
```

```
164  * @param  array  $items    The list of items to be mapped
165  * @param  mixed  $callback A string function name, or array containing an object
166  *                          and a string method name
167  * @return string A set of RSS <item /> elements
168  */
169     function items($items, $callback = null) {
170             if ($callback != null) {
171                     $items = array_map($callback, $items);
172             }
173
174             $out = '';
175             $c = count($items);
176
177             for ($i = 0; $i < $c; $i++) {
178                     $out .= $this->item(array(), $items[$i]);
```

What to test?

What you should test automatically	*What you should test in the browser*	*What you most often don't need to test or can't*
Anything that has a clear contract `[input->magic->output]`	"No tasks found " message in the view	CakePHP's features (most are well tested, but this doesn't mean you shouldn't test)
Anything that went wrong already	(In most cases) MVC Application flow	Whether your site withstands all security attacks.
Anything real money or human lives depend on	Your interface	If your controller sets the right view vars in trivial cases.
Anything you are not confident about	CSS and Javascript	Your users.

Pointers from Tim & I

- Writing assertions first makes you think about your interface - and often helps me notice design errors
- However, sometimes I need to hack out a function and go back and forth between the test and the subject
- use getTests() to limit the tests that run
- Give your tests descriptive names or use custom error messages

Pointers from Tim & I

- Play with the testing shell - ultimate automation is key to world domination!
- Keep tests isolated - two problems need two tests
- Leave the last test broken when you are done for the day -> it's the starting point for tomorrow

Pointers from Tim & I

- If you are lost, start from scratch
- Test-driven-development takes getting used to - don't force yourself to use it; better to write tests after code is written than writing no tests
- Do regular breaks
- If testing does not help you start with a problem, explain that problem to someone else

Things we skipped

- Testing Behaviors.
- Testing Components.
- Using CakeWebTestCase

May the green bar be with you!

Thanks you for coming to CakeFest and listening to me. Kind regards from Tim!

Thank you to the rest of the core team for working so hard on CakePHP.

Questions?