php|tek
PHP::innovation()
Chicago, May 16-18, 2007

Test Driven Development

Jason E. Sweat

blog.casey-sweat.us
jsweat_php@yahoo.com

May 15, 2007

Hyatt Regency Chicago, Illinois

$this->assertTrue(intro());

```
class PhpTekTestDrivenDevelopmentTestCase extends UnitTestCase {
  function TestAuthor() {
    $talk = new PhpTekTestDrivenDevelopment;
    $author = $talk->getAuthor();

    $this->assertTrue($author->introduction());
    $this->assertEqual('Jason', $author->first_name);
  }
  function TestPresentation() {
    $talk = new PhpTekTestDrivenDevelopment;

    $this->assertTrue($talk->introduceTesting());
    $this->assertTrue($talk->liveExample());
    $this->assertTrue($talk->introduceTestDrivenDevelopement());
    $this->assertTrue($talk->showSimpleTest());
    $this->assertTrue($talk->continueExample(new AudianceParticipation));
    $this->assertTrue($talk->questionsAndAnswers());
  }
}
```

What is Testing?

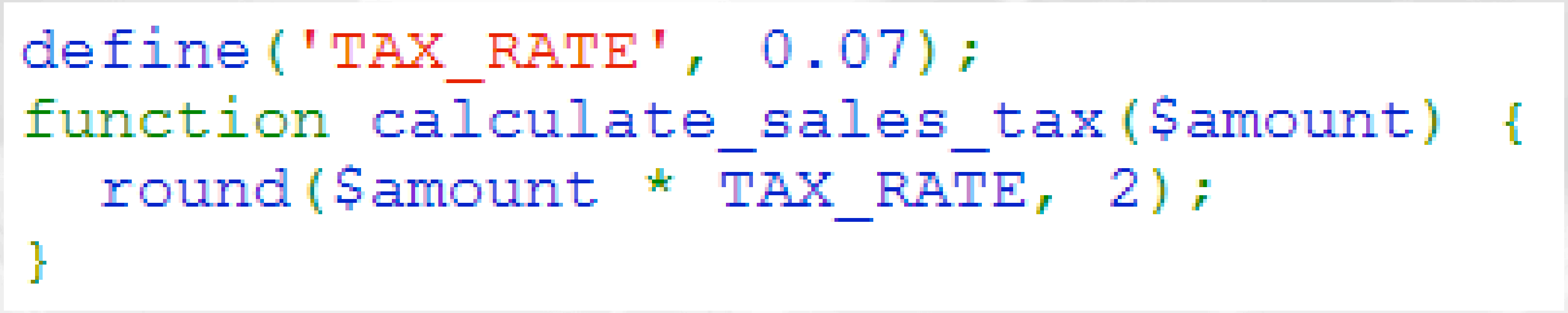
- Unit Tests are code written to exercise pieces—units—of your application and verify the results meet your expectations
- Various Unit Testing frameworks exist to let you run this tests in an automated manner
  - http://simpletest.org/
  - http://pear.php.net/package/PHPUnit2/
  - http://qa.php.net/write-test.php
  - Others (90% of all PHP testing frameworks are named phpunit) - http://www.google.com/search?q=phpunit
- Nearly all modeled off of junit
  - http://junit.org/
- TAP (Test Anything Protocol)

SimpleTest

- Many PHP Testing Frameworks available
- SimpleTest used here because
  - PHP4 or PHP5
  - Well documented (api, tutorials, articles)
  - Support for MockObjects
  - Support for WebTesting
  - Marcus Baker is a sharp coder, a great teacher, and a really great guy

Lets get started...

Some Code to Test



```php
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

A Test

```
class SalesTaxTestCase extends UnitTestCase {
    function testSalesTax() {
        $this->assertEqual(7, calculate_sales_tax(100));
    }
}
```

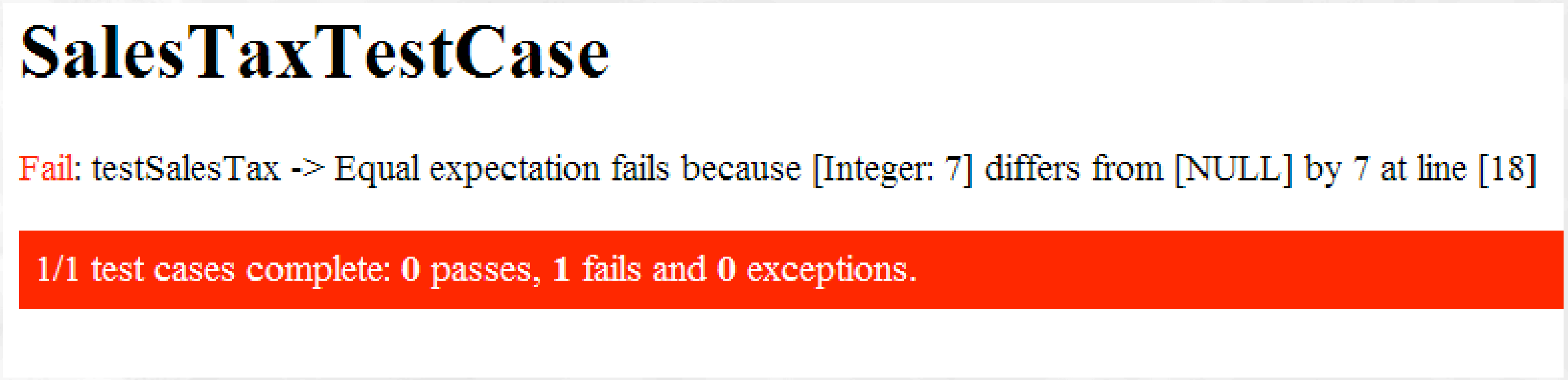
A Test

```
class SalesTaxTestCase extends UnitTestCase {
    function testSalesTax() {
        $this->assertEqual(7, calculate_sales_tax(100));
    }
}
```

- Run It

```
$test = new SalesTaxTestCase;
$test->run(new HtmlReporter());
```

- What happened?



SalesTaxTestCase

Fail: testSalesTax -> Equal expectation fails because [Integer: 7] differs from [NULL] by 7 at line [18]

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

Fixing our Code

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

Fixing our Code

- We had:

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  return round($amount * TAX_RATE, 2);
}
```

Fixing our Code

- We had:

```php
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

```php
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  return round($amount * TAX_RATE, 2);
}
```

Fixing our Code

- We had:

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  return round($amount * TAX_RATE, 2);
}
```

Fixing our Code

- We had:

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  return round($amount * TAX_RATE, 2);
}
```

Fixing our Code

- We had:

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  return round($amount * TAX_RATE, 2);
}
```

Fixing our Code

- We had:

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  return round($amount * TAX_RATE, 2);
}
```

Fixing our Code

- We had:

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  return round($amount * TAX_RATE, 2);
}
```

Fixing our Code

- We had:

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  return round($amount * TAX_RATE, 2);
}
```

Fixing our Code

- We had:

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  return round($amount * TAX_RATE, 2);
}
```

Fixing our Code

- We had:

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  return round($amount * TAX_RATE, 2);
}
```

Fixing our Code

- We had:

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  return round($amount * TAX_RATE, 2);
}
```

Fixing our Code

- We had:

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

```
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  return round($amount * TAX_RATE, 2);
}
```

Fixing our Code

- We had:

```php
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

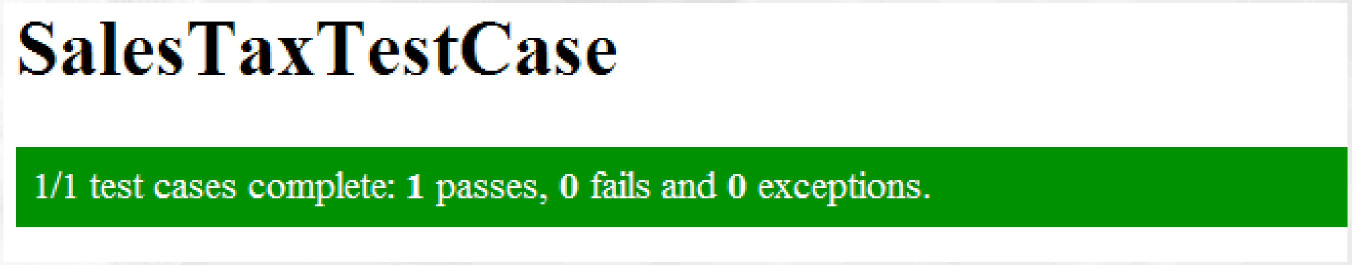
```php
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  return round($amount * TAX_RATE, 2);
}
```

Fixing our Code

- We had:

```php
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  round($amount * TAX_RATE, 2);
}
```

```php
define('TAX_RATE', 0.07);
function calculate_sales_tax($amount) {
  return round($amount * TAX_RATE, 2);
}
```

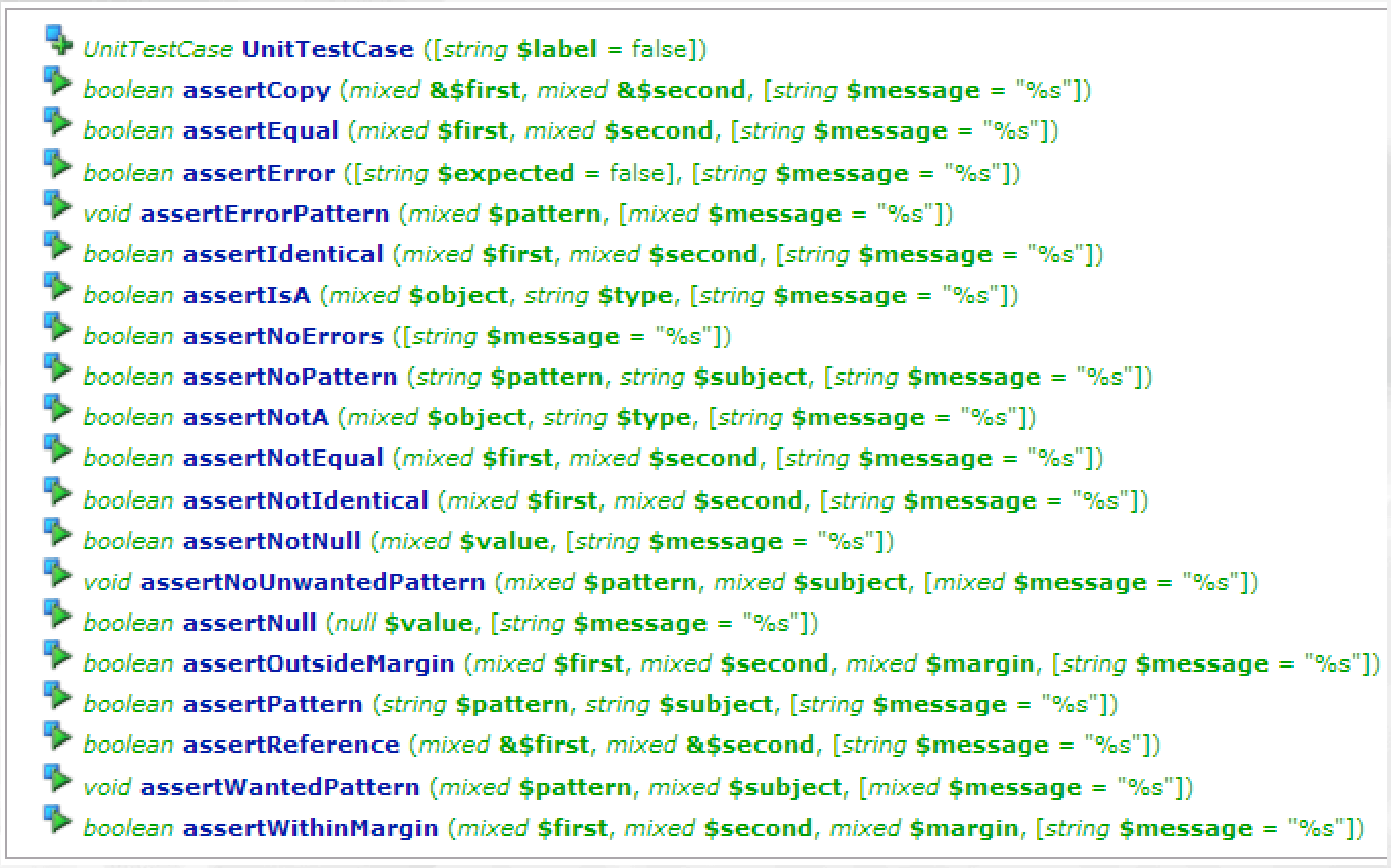


SalesTaxTestCase

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

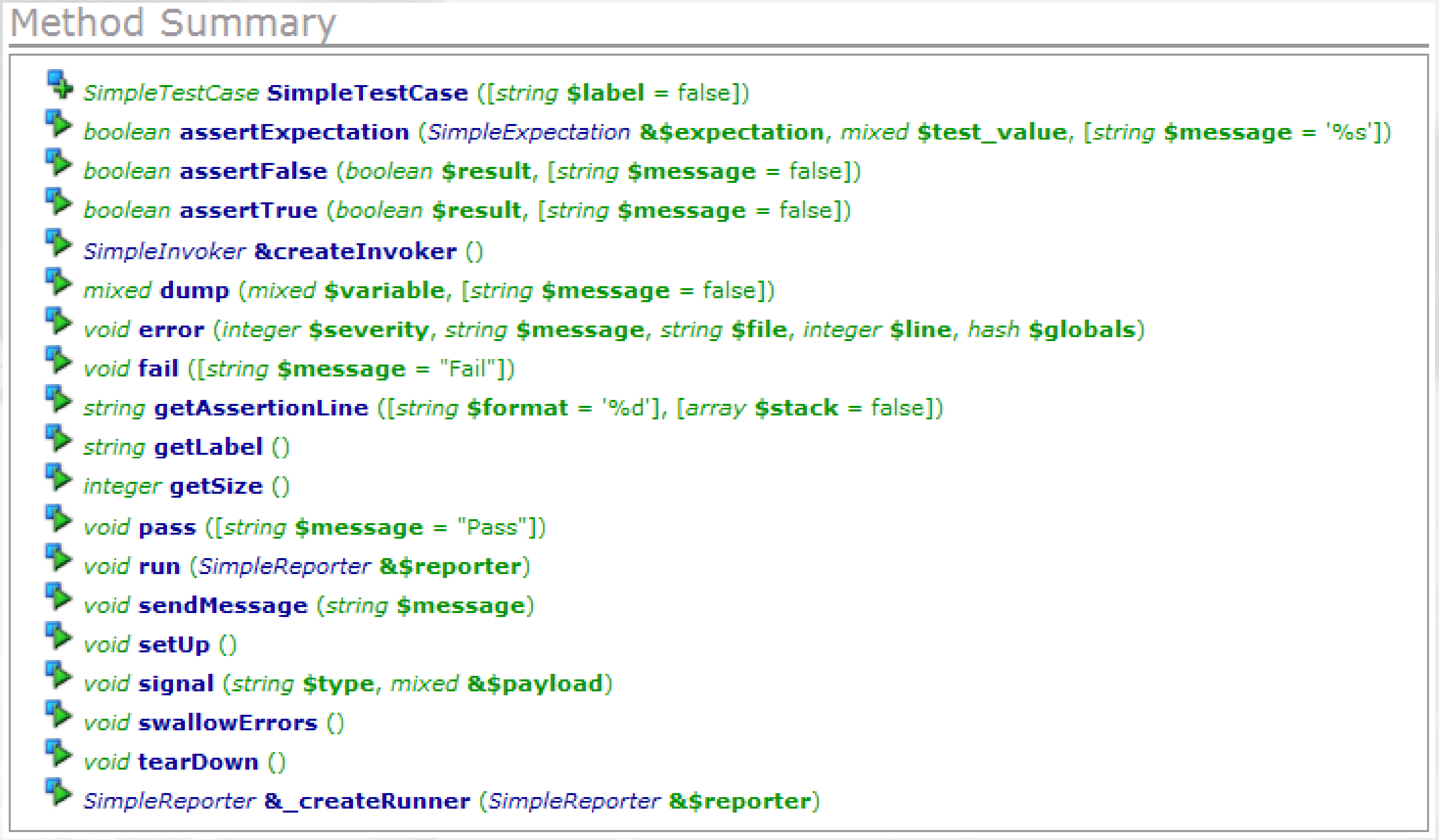
- Now we rerun the test and are rewarded with our green bar J

More Assertions



```
UnitTestCase UnitTestCase ([string $label = false])
boolean assertCopy (mixed &$first, mixed &$second, [string $message = "%s"])
boolean assertEqual (mixed $first, mixed $second, [string $message = "%s"])
boolean assertError ([string $expected = false], [string $message = "%s"])
void assertErrorPattern (mixed $pattern, [mixed $message = "%s"])
boolean assertIdentical (mixed $first, mixed $second, [string $message = "%s"])
boolean assertIsA (mixed $object, string $type, [string $message = "%s"])
boolean assertNoErrors ([string $message = "%s"])
boolean assertNoPattern (string $pattern, string $subject, [string $message = "%s"])
boolean assertNotA (mixed $object, string $type, [string $message = "%s"])
boolean assertNotEqual (mixed $first, mixed $second, [string $message = "%s"])
boolean assertNotIdentical (mixed $first, mixed $second, [string $message = "%s"])
boolean assertNotNull (mixed $value, [string $message = "%s"])
void assertNoUnwantedPattern (mixed $pattern, mixed $subject, [mixed $message = "%s"])
boolean assertNull (null $value, [string $message = "%s"])
boolean assertOutsideMargin (mixed $first, mixed $second, mixed $margin, [string $message = "%s"])
boolean assertPattern (string $pattern, string $subject, [string $message = "%s"])
boolean assertReference (mixed &$first, mixed &$second, [string $message = "%s"])
void assertWantedPattern (mixed $pattern, mixed $subject, [mixed $message = "%s"])
boolean assertWithinMargin (mixed $first, mixed $second, mixed $margin, [string $message = "%s"])
```

More Assertions



Method Summary

```
SimpleTestCase SimpleTestCase ([string $label = false])
boolean assertExpectation (SimpleExpectation &$expectation, mixed $test_value, [string $message = '%s'])
boolean assertFalse (boolean $result, [string $message = false])
boolean assertTrue (boolean $result, [string $message = false])
SimpleInvoker &createInvoker ()
mixed dump (mixed $variable, [string $message = false])
void error (integer $severity, string $message, string $file, integer $line, hash $globals)
void fail ([string $message = "Fail"])
string getAssertionLine ([string $format = '%d'], [array $stack = false])
string getLabel ()
integer getSize ()
void pass ([string $message = "Pass"])
void run (SimpleReporter &$reporter)
void sendMessage (string $message)
void setUp ()
void signal (string $type, mixed &$payload)
void swallowErrors ()
void tearDown ()
SimpleReporter &_createRunner (SimpleReporter &$reporter)
```

Test Driven Development

- Now we have reviewed the SimpleTest framework
- Look at the agile development methodology of Test Driven Development
  - Popularized by XP – eXtreme Programming
- Turns the testing process on it's head
  - Instead of writing tests once you have your code
  - Write tests **before** you code

TDD Steps

- Write a test
- Observe the failure
  - Red bar
- Write the code to allow your test to pass
  - Do the simplest thing that will work
- Run the passing test
  - Green bar
- Refactor if required
  - Eliminate the sins of code duplication
- Repeat with the first step for new requirements

TDD Steps

- Write a test
- Observe the failure
  - Red bar
- Write the code to allow your test to pass
  - Do the simplest thing that will work
- Run the passing test
  - Green bar
- Refactor if required
  - Eliminate the sins of code duplication
- Repeat with the first step for new requirements

- Rule #1 - Audience Participation
  - We pick a project together
  - We start coding – TDD style
    - Code a test
    - Watch it fail
    - Code the application
    - Watch it pass
    - Refactor if necessary
    - Repeat
  - TDD Mantra – Red/Green/Refactor
  - Ask any questions that come up
  - After we get the feel for it, we can talk more about the benefits of TDD

Why Test?

- You already test your code
  - Run it to make sure there are no parse errors
  - Send in parameters and verify you get what you expect
  - var_dump() or printr() in those “tricky” spots to make sure you know what you are dealing with
  - Maybe a colleague or a QA person also poke around in some different areas
  - Perhaps you occasionally help test your colleagues code by hand (code review sessions?)
- Then boredom sets in
  - Why test those parts of the code you already tested? After all, you did not change anything in *that* part of your code.

Comments?

- You do comment your code right?
  - At least what you intend for major classes or functions to do
  - Maybe docblocks for automatic source code documentation
- Comments get stale
  - Do you always change comments when you change the code?
  - Do you trust someone else's comments regarding code?

Bug Hunts

- By definition, you have 0 productivity when you are debugging, rather than programming or designing
- The more complex and greater in scope your application is, the more intrusive debugging measures you will need to undertake
- Bugs do not always manifest immediately, you may have to sift back through weeks or months of code to locate it

Code Rot

- Systems evolve
  - Typically get bigger
  - More complex
  - Much of the complexity derives from interactions between different parts of the code
- Programmer turnover
  - Often people maintaining software are not the original authors
- Fear of changing the code sets in

Why Automated Tests?

- Testing changes is not the same as having tests
- Define explicit successful behavior for your code
  - Tests read like comments which can't lie
  - Tests are explicit documentation of how objects are expected to behave
- Build more test coverage over time
  - You continuously apply your successful behavior criteria, even after you are no longer working on that part of your application

So What Does Testing Buy You?

- Freedom
  - How can this be? Spending extra time writing tests to verify code I know is good has to be confining, not introducing freedom.
- Confidence
  - Know when you have solved a problem
  - Know changes you have made recently do not have unintended consequences in other parts of your application

UnitTesting Benefits

- Tests are easy to run, so you run them more often
- Tests are more complete than random manual validation in the area of your application you are currently working on
  - You are more likely to detect changes which affect other portions of your code base
- Test coverage is a key step in Refactoring
- Bug reports can be defined as test cases
  - Changing to a passing test indicates you have solved the bug
  - The test remains part of your test suite so you are sure the bug will not creep back into you code base

Terminology

- **UnitTests** – tests of a “unit” of code, typically a class or a function, generally written by the developer
- **AcceptanceTest** – test for the end functionality of an application, generally written to a customers specification a.k.a. functional tests
- **BlackBox Testing** – testing of only the publicly visible API of a system, i.e. you don’t know what is inside of the box a.k.a. behavioral testing
- **WhiteBox Testing** – testing with greater knowledge of the implementation (may give you greater initial confidence by may also lead to brittle tests) a.k.a. structural testing
- **Assertion** – a statement which creates an expectation about the code
- **TestMethod** – a function grouping one or more related assertions
- **TestCase** – a group of one or more related test methods
- **GroupTest** – several related test cases

SimpleTest in more depth- Roll Your Own Assertions

- Write a method in your TestCase which does not start with Test
  - Use combination of existing assertions
  - I call this a “helper method”
- Use AssertExpectation()
  - Subclass SimpleExpectation

Example Problem Requiring New Assertion

```
class AssertionTestCase extends UnitTestCase {
    function testFloatProblem() {
        $this->assertNotEqual(1.87654, 5629.62/3000);
        /*
        if assertEqual
        1) Equal expectation fails because Float: 1.87654] differs
           from [Float: 1.87654] at line [9]
                in testFloatProblem
                in AssertionTestCase
        */
    }
```

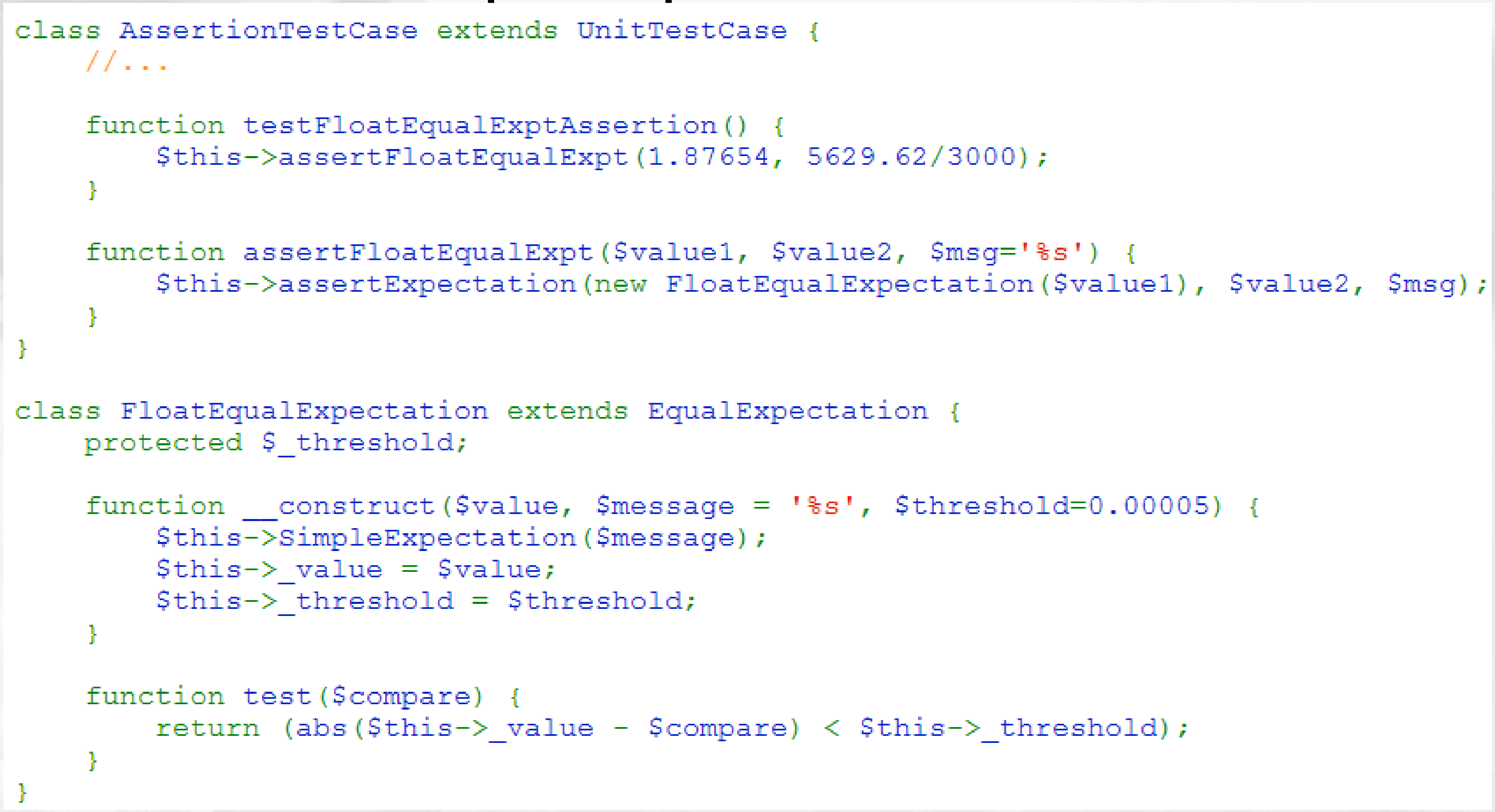
Adding a new assertion to the TestCase

- Add a new "helper method"
  - Function name must not start with "test"

```
function testFloatEqualAssertion() {
    $this->assertFloatEqual(1.87654, 5629.62/3000);
}

function assertFloatEqual($value1, $value2, $msg='') {
    $this->assertTrue(abs($value1-$value2) < 0.00005, $msg);
}
```

Adding Assertions using Expectations

- Extend SimpleExpectation



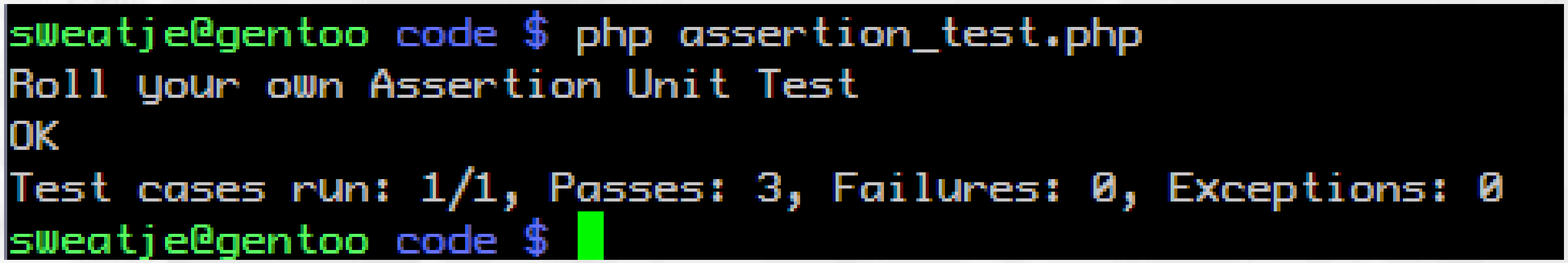
```
class AssertionTestCase extends UnitTestCase {
    //...

    function testFloatEqualExptAssertion() {
        $this->assertFloatEqualExpt(1.87654, 5629.62/3000);
    }

    function assertFloatEqualExpt($value1, $value2, $msg='%s') {
        $this->assertExpectation(new FloatEqualExpectation($value1), $value2, $msg);
    }
}

class FloatEqualExpectation extends EqualExpectation {
    protected $_threshold;

    function __construct($value, $message = '%s', $threshold=0.00005) {
        $this->SimpleExpectation($message);
        $this->_value = $value;
        $this->_threshold = $threshold;
    }

    function test($compare) {
        return (abs($this->_value - $compare) < $this->_threshold);
    }
}
```

Different Test Reporters

- Make it as easy as possible to test
  - Allow running from either command line or from browsing to a web page

```php
if (TextReporter::inCli()) {
    exit ($test->run(new TextReporter()) ? 0 : 1);
}
$test->run(new HtmlReporter());
```

- Now $test will run with the appropriate reporter



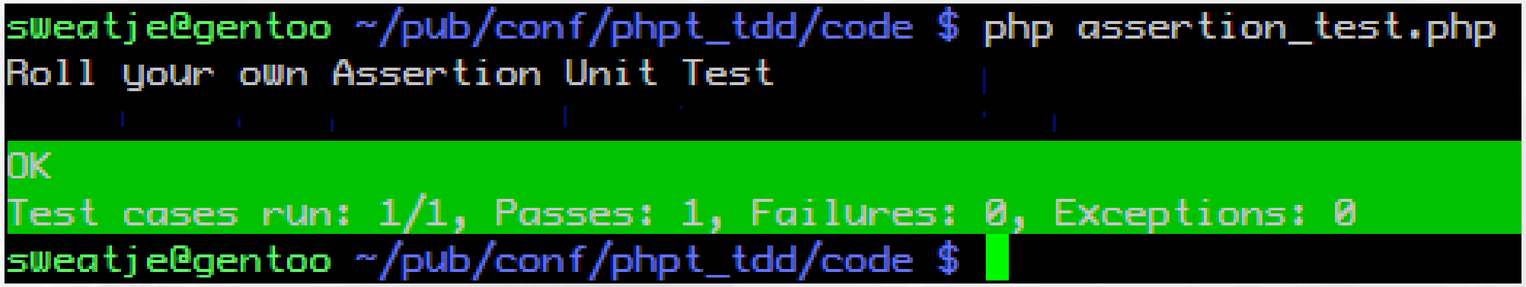
```
sweatje@gentoo code $ php assertion_test.php
Roll your own Assertion Unit Test
OK
Test cases run: 1/1, Passes: 3, Failures: 0, Exceptions: 0
sweatje@gentoo code $
```

If you miss your colored bars in CLI

- Change you choice of reporter

```php
if (TextReporter::inCli()) {
    require_once 'simpletest/ui/colortext_reporter.php';
    exit ($test->run(new ColorTextReporter()) ? 0 : 1);
}
$test->run(new HtmlReporter());
```

- And view the output



```
sweatje@gentoo ~/pub/conf/phpt_tdd/code $ php assertion_test.php
Roll your own Assertion Unit Test

OK
Test cases run: 1/1, Passes: 1, Failures: 0, Exceptions: 0
sweatje@gentoo ~/pub/conf/phpt_tdd/code $
```

MockObjects

- A MockObject is an object which you can use to substitute for another object in a test, and validate expected interactions took place during the test
- MockObjects have two main roles:
  - Respond appropriately to method calls (this is the “actor” role, that of the ServerStub testing pattern)
  - Verify method calls were made on the Mock Object (this is the “critic” role, and what distinguishes a Mock from a Stub)

Using MockObjects

- SimpleTest has an implementation to dynamically generate the MockObjects from your existing class
- Use of MockObjects in your testing
  - Isolates your code to just the unit you are testing
  - Focuses your attention on interface rather than implementation

Some Code to Test

```php
class RemoveAction {
    protected $access;
    protected $model;

    public function __construct($access, $model) {
        $this->access = $access;
        $this->model = $model;
    }

    public function perform($post) {
        if ($this->access->can('delete model')) {
            $this->model->remove($post->get('id'));
        }
    }

}
```

We Don't Need More Than The API

- We can flesh out the details of the other classes implementations later

```
class Access {
    public function can($permision) {}
}
class Model {
    public function remove($id) {}
}
class Post {
    public function get($key) {}
}
```

Testing with MockObjects

```
Mock::generate('Access');
Mock::generate('Model');
Mock::generate('Post');

class MockTestCase extends UnitTestCase {
    function testNoAccess() {
        //setup the mock objects
        $access = new MockAccess($this);
        $access->setReturnValue('can', false);
        $access->expectOnce('can', array('delete model'));

        $model = new MockModel($this);
        $model->expectNever('remove');

        $post = new MockPost($this);
        $post->expectNever('get');

        //perform the test
        $action = new RemoveAction($access, $model);
        $action->perform($post);

    }
```

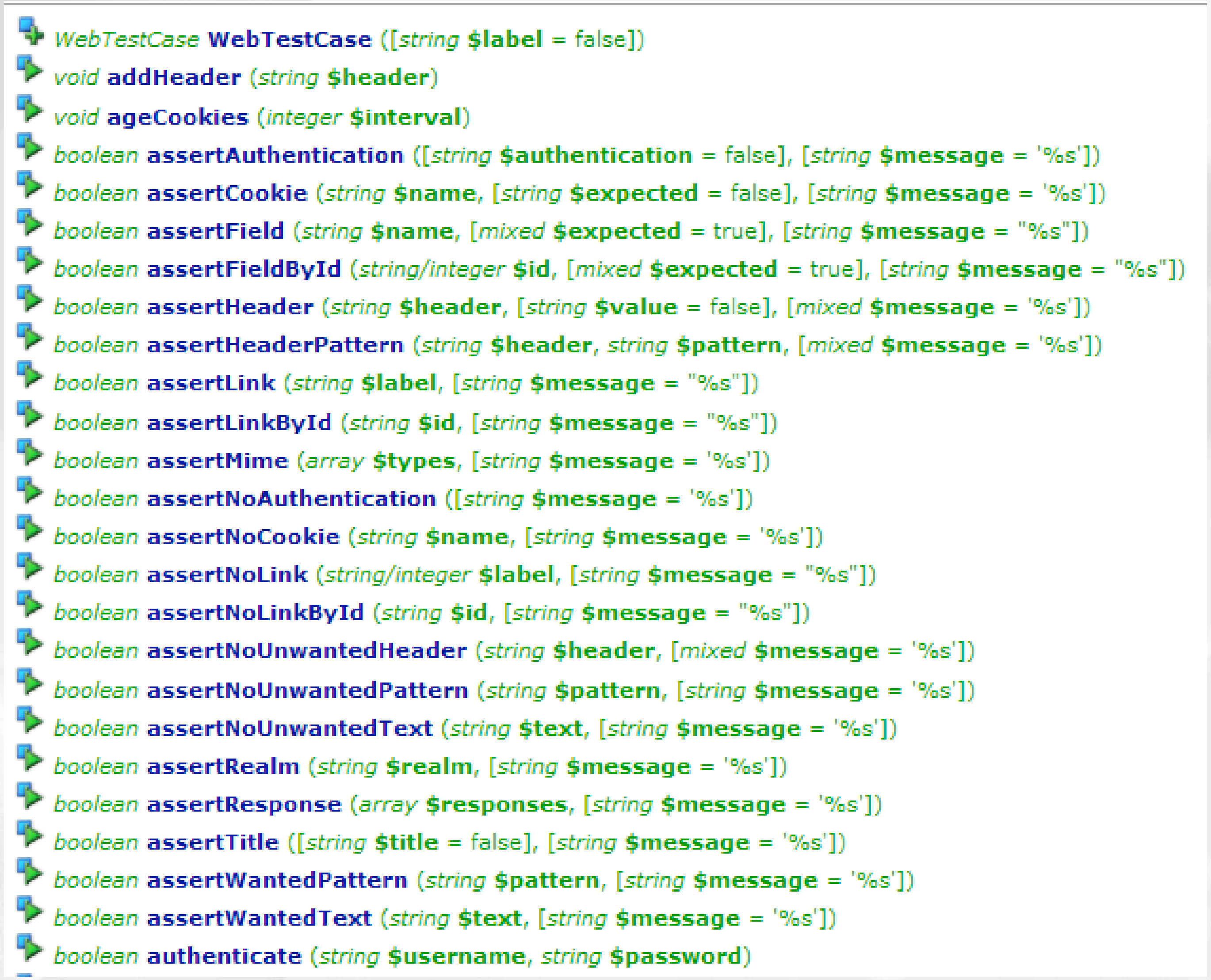
And Another Test Method

```
function testHasAccess() {
    //setup the mock objects
    $access = new MockAccess($this);
    $access->setReturnValue('can', true);
    $access->expectOnce('can', array('delete model'));

    $test_id = 4;

    $post = new MockPost($this);
    $post->setReturnValue('get', $test_id);
    $post->expectOnce('get', array('id'));

    $model = new MockModel($this);
    $model->expectOnce('remove', array($test_id));

    //perform the test
    $action = new RemoveAction($access, $model);
    $action->perform($post);

}
```

WebTesting

- Testing the application by using the site
  - Application acts like a user
- Similar to jWebUnit (http://jwebunit.sf.net/)

WebTestCase Assertions



```
WebTestCase WebTestCase ([string $label = false])
void addHeader (string $header)
void ageCookies (integer $interval)
boolean assertAuthentication ([string $authentication = false], [string $message = '%s'])
boolean assertCookie (string $name, [string $expected = false], [string $message = '%s'])
boolean assertField (string $name, [mixed $expected = true], [string $message = "%s"])
boolean assertFieldById (string/integer $id, [mixed $expected = true], [string $message = "%s"])
boolean assertHeader (string $header, [string $value = false], [mixed $message = '%s'])
boolean assertHeaderPattern (string $header, string $pattern, [mixed $message = '%s'])
boolean assertLink (string $label, [string $message = "%s"])
boolean assertLinkById (string $id, [string $message = "%s"])
boolean assertMime (array $types, [string $message = '%s'])
boolean assertNoAuthentication ([string $message = '%s'])
boolean assertNoCookie (string $name, [string $message = '%s'])
boolean assertNoLink (string/integer $label, [string $message = "%s"])
boolean assertNoLinkById (string $id, [string $message = "%s"])
boolean assertNoUnwantedHeader (string $header, [mixed $message = '%s'])
boolean assertNoUnwantedPattern (string $pattern, [string $message = '%s'])
boolean assertNoUnwantedText (string $text, [string $message = '%s'])
boolean assertRealm (string $realm, [string $message = '%s'])
boolean assertResponse (array $responses, [string $message = '%s'])
boolean assertTitle ([string $title = false], [string $message = '%s'])
boolean assertWantedPattern (string $pattern, [string $message = '%s'])
boolean assertWantedText (string $text, [string $message = '%s'])
boolean authenticate (string $username, string $password)
```

TDD Benefits

- Steady predictable development cycle
- Automatically builds more complete code coverage in your tests
- Shifts focus towards interfaces between related objects, as opposed to just implementation details
  - Towards a goal of higher cohesion – lower coupling design
    - Good idea to begin with
    - Make testing easier
- Builds team communications
- No Big Up Front Design
  - You code evolves with actual use cases, not what you think you might need

Acronyms to Remember

- TDD – Test Driven Development
- DRY – Don't Repeat Yourself
- YAGNI – You Ain't Gonna Need It
  - Do the simplest thing that works
- XP – eXtreme Programming
- BUFD – Big Up Front Design

TDD Live Environment

- OS
  - Linux
  - Running on vmware workstation
- PHP
  - Version 5.1.2
  - CLI and mod_php
- Apache2
  - Version 2.0.55
- MySQL
  - Version 4.1.14
- PHP Software
  - Simpletest – cvs
  - ADOdb – 450
  - phpMyAdmin – 2.7.0

Resources

- Kent Beck *Test-driven development: by example* Addison-Wesley, 2003
- Martin Fowler *Refactoring: improving the design of existing code* Addison-Wesley, 1999
- Jason E. Sweat *Php Architect's Guide to Php Design Patterns* Marco Tabini & Associates, 2005

Online Resources

- General Testing Links
  - http://www.testdriven.com/
  - http://www.mockobjects.com/
- Recommended Testing Frameworks
  - http://simpletest.org/
  - http://www.edwardh.com/jsunit/
- Articles
  - http://www.developerspot.com/tutorials/php/test-driven-development/

Conclusion

- Introduced you to automated testing
- Reviewed SimpleTest as a unit testing framework
- Examined the Test Driven Development process
- Tried our hand at a live example

I hope you all are now "Test Infected"
http://junit.sourceforge.net/doc/testinfected/testing.htm