Unit Testing PHP Apps

By: Vineet Garg

ATLOGYS
Technical Consulting

Introduction to Unit Testing

- **Unit** can be defined as the smallest testable part of an application.
- **Unit Testing** is a software testing method by which individual units of source code are tested to determine if they are functioning as expected.
- **Unit Testing** is performed by using the White Box Testing method.
- **Unit Testing** is the first level of testing and is performed prior to Integration Testing.

ATLOGYS
Technical Consulting

Benefits of Unit Testing

- Improves code design by forcing you to design in a more structured and modular way.
- Helps in detecting low level coding bugs.
- Makes it easier to test and refactor code.
- Early bugs are easier and cheaper to fix.
- Reduction in number of bugs found by QA team.
- Works as a documentation. Going through the unit tests can show you the intended functionality of the code including all edge cases.
- Unit tests can be reused.

ATLOGYS
Technical Consulting

Disadvantage of Unit Testing

- Development time increases (approx. 30%) .
- Additional Complexity.
- Difficult to ensure that written test case covers all corner cases.
- Continuous code tweaking increases investment in writing/modifying test cases.

ATLOGYS
Technical Consulting

Tasks in Unit Testing

- **Create Unit Test Plan :** Create a test plan on how the unit testing is to be performed.
- **Create Unit Test Cases/Scripts :** After proper planning the test cases/scripts are created. Ensure that the created tests covers all the code and are relevant to the tested functionality.
- **Perform Unit Test :** Once the test cases are built, unit testing is performed.

ATLOGYS
Technical Consulting

Writing Unit Testable Code

- Keep code DRY (Don’t Repeat Yourself).
- Remove Dependency injection, code that depend on global state like sessions, global variables is difficult to test.
- Keep functions small and specific.
- High **cohesion** and **loose coupling** is the definition of good, maintainable, testable code.

ATLOGYS Technical Consulting

Unit Testing Frameworks

- **Some of the frameworks for PHP language -**
  - PHPUnit (by Sebastian Bergmann)
  - Simple Test (by Marcus Baker)
  - Lime (by Fabien Potencier) for Symfony web application framework.
  - Many more…

  See Full list
  http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#PHP

ATLOGYS
Technical Consulting

PHPUnit

- PHPUnit is a programmer-oriented testing framework for PHP.
- PHPUnit uses assertions to verify that behaviour of the unit of code under test behaves as expected.
- It was created by Sebastian Bergmann.
- Latest Version 4.1.0
- Website : http://phpunit.de/
- Installation : PHPUnit comes preinstalled with xampp, if you are not using xampp follow the link below to install: http://phpunit.de/manual/current/en/phpunit-book.html#installation

ATLOGYS Technical Consulting

Writing tests: Assertions

- **Testing array operations with PHPUnit:**

```
<?php
class StackTest extends PHPUnit_Framework_TestCase
{
    public function testPushAndPop()
    {
        $stack = array();
        $this->assertEquals(0, count($stack));

        array_push($stack, 'foo');
        $this->assertEquals('foo', $stack[count($stack)-1]);
        $this->assertEquals(1, count($stack));

        $this->assertEquals('foo', array_pop($stack));
        $this->assertEquals(0, count($stack));
    }
}
?>
```

ATLOGYS
Technical Consulting

Contd…

- In the code snippet you will note that the class ‘Stacktest’ extends the ‘PHPUnit_Framework_TestCase’, every test case class you write should extend the given class.
- **assertEquals(x, y)** is an assertion method that checks that two values are equal or not. If values are equal the test will pass or else it will fail.
- There are number of assertion methods that can be used to facilitate testing. Complete list can be found at the link below: http://phpunit.de/manual/current/en/phpunit-book.html#appendixes.assertions

ATLOGYS
Technical Consulting

Testing Dependencies

- PHPUnit uses **@depends** annotation to express dependencies
- This annotation can be used in scenario where input for one method depends on output of another method.
- The tests are run in order of dependencies, a test will not run if the method it is depending on fails.
- In the snippet**, testPush()** method depends on **testEmpty()** method.
- **testEmty()** method will run only if **testPush()** method passes test else it will be skipped.

```php
<?php
class StackTest extends PHPUnit_Framework_TestCase
{
    public function testEmpty()
    {
        $stack = array();
        $this->assertEmpty($stack);

        return $stack;
    }

    /**
     * @depends testEmpty
     */
    public function testPush(array $stack)
    {
        array_push($stack, 'foo');
        $this->assertEquals('foo', $stack[count($stack)-1]);
        $this->assertNotEmpty($stack);

        return $stack;
    }

    /**
     * @depends testPush
     */
    public function testPop(array $stack)
    {
        $this->assertEquals('foo', array_pop($stack));
        $this->assertEmpty($stack);
    }
}
?>
```

ATLOGYS
Technical Consulting

Data Providers

- Suppose a method needs to be tested on number of input variations. One way is to write statements manually for each input but that can be quite cumbersome. To solve this problem PHPUnit uses **@dataprovider** annotation. Using which we can write a method which provides input data.
- A data provider method must be public and either return an array of arrays or an object that implements the Iterator interface and yields an array for each iteration step. For each array that is part of the collection the test method will be called with the contents of the array as its arguments.

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

    public function additionProvider()
    {
        return array(
          array(0, 0, 0),
          array(0, 1, 1),
          array(1, 0, 1),
          array(1, 1, 3)
        );
    }
}
?>
```

ATLOGYS Technical Consulting

Fixtures

- When testing the methods that operates on the data coming from database its becomes very important to ensure that once a test case is executed the state of database should remain as it was before running the tests.
- For this PHPunit provides fixtures which can be used to source the data for methods.
- The **setUp()** and **tearDown()** template methods are run once for each test method (and on fresh Instances) of the test case class. **setUp()** method will setup the data to be used in running test case and after the execution is completed **tearDown()** will bring down the data source to its initial source.

```php
<?php
class StackTest extends PHPUnit_Framework_TestCase
{
    protected $stack;

    protected function setUp()
    {
        $this->stack = array();
    }

    public function testEmpty()
    {
        $this->assertTrue(empty($this->stack));
    }

    public function testPush()
    {
        array_push($this->stack, 'foo');
        $this->assertEquals('foo', $this->stack[count($this->stack)-1]);
        $this->assertFalse(empty($this->stack));
    }

    public function testPop()
    {
        array_push($this->stack, 'foo');
        $this->assertEquals('foo', array_pop($this->stack));
        $this->assertTrue(empty($this->stack));
    }
}
?>
```

ATLOGYS
Technical Consulting

Mock database using fixture

- This is a simple example of mocking a database using fixture in CakePHP framework.
- **PostFixture** class mocks the model **Post** which references post table containing all posts.
- **PostTest** class uses the fixture **PostFixture** to test the post model.
- **Setup()** function loads all the records from post table in the **Post** variable which is used by test case **testAllPosts** to verify the count of records in table.

```php
<?php
class PostFixture extends CakeTestFixture {
    // public $import = 'Posts';
    public $import = array('model' => 'Post', 'records' => true);
}

class PostTest extends CakeTestCase {
    public $fixtures = array('app.post');

    public function setUp() {
        parent::setUp();
        $this->Post = ClassRegistry::init('Post');
    }


    public function testAllPosts() {
        $this->loadFixtures('Post');

        $result = $this->Post->all_posts();
        $expected = array(
            array('Article' => array('id' => 1, 'title' => 'First Article')),
            array('Article' => array('id' => 2, 'title' => 'Second Article')),
            array('Article' => array('id' => 3, 'title' => 'Third Article'))
        );

        $this->assertEquals(sizeof($result), 3);

    }

    public function testCreate(){

        $res = $this->Post->add(array('title' => 'testr', 'body' => 'body'));
        $this->assertTrue($res);
    }
}
```

ATLOGYS
Technical Consulting

PHPUnit with MVC Frameworks

- In order to facilitate fast, easy and secure development MVC frameworks are used.
- Some of the popular PHP frameworks:
  - Zend
  - Cake PHP
  - Codeigniter
  - Symphony
- All frameworks above support PHPUnit natively except for codeigniter. For instance if you are using cake PHP you can follow the link below, to view the complete test guide: http://book.cakephp.org/2.0/en/development/testing.html

ATLOGYS
Technical Consulting

Contd...

- Cake PHP provides its own command line utility which can be used to run test cases, you can also run test cases from your browser in an easy way and view the detailed test report with code coverage(using Xdebug).
- Cake PHP also provide the support to test models, controllers and views and easy creation of test suites.

ATLOGYS
Technical Consulting

Resources

- http://phpunit.de/manual/current/en/phpunit-book.html
- http://code.tutsplus.com/tutorials/how-to-write-testable-and-maintainable-code-in-php--net-31726
- http://www.ibm.com/developerworks/xml/library/os-refactoringphp/
- http://www.sitepoint.com/tutorial-introduction-to-unit-testing-in-php-with-phpunit/
- https://jtreminio.com/2013/03/unit-testing-tutorial-introduction-to-phpunit/
- http://phpunit.de/presentations.html