Laravel Unit Testing

BY DR. HASSAN AMIN

VP TECHNOLOGY

COEUS SOLUTIONS

Agenda

- What is unit test ?
- Why write unit tests ?
- Excuses for not Writing Unit Tests
- When to Implement Unit Testing
- Steps for Writing Unit Test
- Introduction to PHPUnit and Laravel Test Helpers
  - Assertions
  - Examples

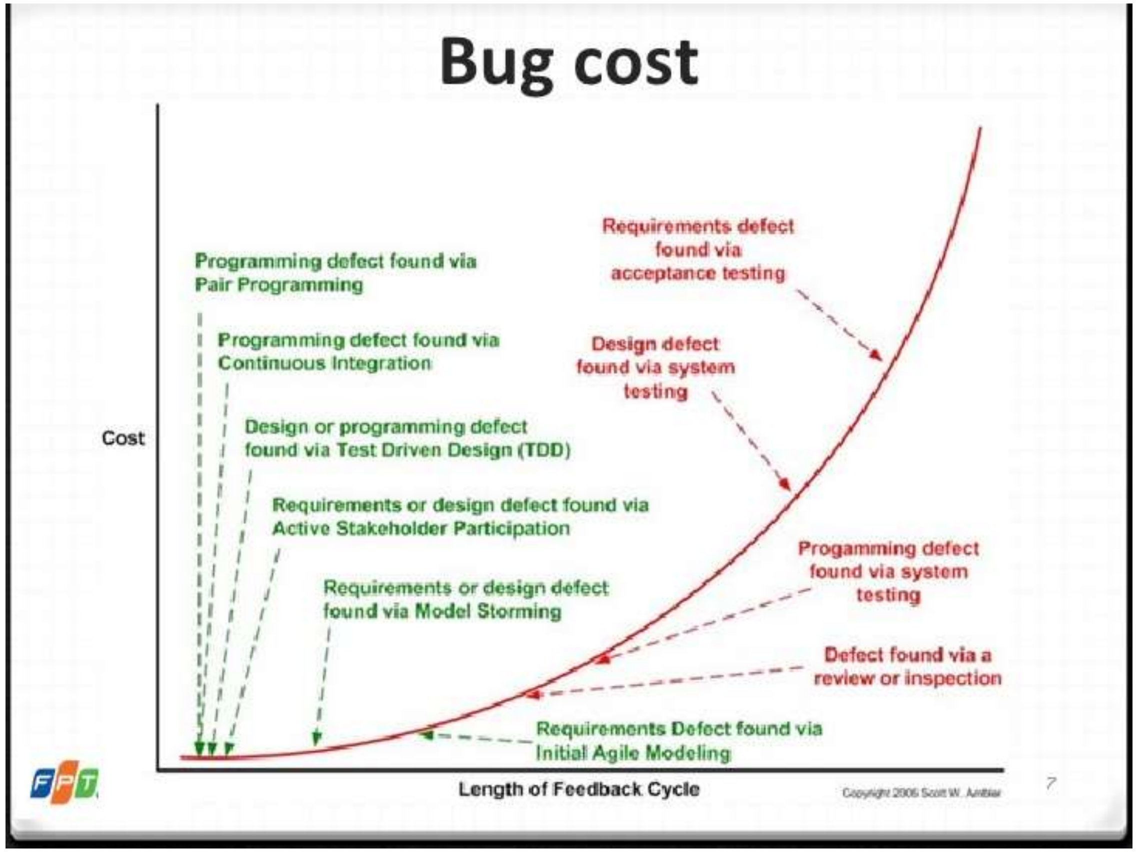
Unit Test

"Unit test checks a single assumption about behavior of the system"

Key Reasons for Writing Unit Tests

- Manual testing is tedious, time consuming and unreliable
- Pity the fool who does not write tests(Coding Horrors : https://blog.codinghorror.com/i-pity-the-fool-who-doesnt-write-unit-tests/)
- Unit testing reduces number of bugs in new features
- Unit testing reduces number of bugs in existing features
- Testing Makes Development Faster (http://www.onjava.com/pub/a/onjava/2003/04/02/javaxpckbk.html)
- Unit Testing is fun for those who love programming challenges
- Unit Tests offer best defense against folly of another programmer messing up with your code
- Unit tests act as a safety net when you change existing code
- Unit tests are best documentation of behavior of the system

Key Reasons for Writing Unit Tests (Cont’d)



Excuses for not Writing Unit Tests

"I don't know how to write tests."

"Writing tests is too hard."

"I don't have enough time to write tests."

"Testing is not my job."

"My code is too simple for tests."

Reference

http://www.onjava.com/pub/a/onjava/2003/04/02/javaxpckbk.html

When to Implement Unit Testing

- When building new classes that contain important business logic.
- When you have functionality that is used and relied upon heavily throughout your app.
- When you want to refactor some functionality that doesn't already have unit tests and is relatively easy to turn into testable code without altering the existing business logic (this never happens…)

Steps for Writing Unit Test

In every test, we generally perform the following three distinct tasks:

1. Arrange : We arrange or initialize some data.

2. Execute : We execute a function to act on this data.

3. Verify : We assert or verify that the output matches what we expected.

Example

```
class HelperTest extends TestCase {
public function testSum() {
$data = [1,2,3];                              // 1) Arrange
$result = Helper::sum($data);       // 2) Act
$this->assertEquals(6, $result);    // 3) Assert
}
public function testSomethingElse() {
// ...
}
}
```

Introduction to PHPUnit and Laravel Test Helpers

- PHPUnit is one of the oldest and most well known unit testing packages for PHP
- It is primarily designed for unit testing, which means testing your code in the smallest components possible
- The purpose of this presentation is to introduce you to the basics of PHPUnit testing, using both the default PHPUnit assertions, and the Laravel test helpers.
  - The aim is for you to be confident writing basic tests for your applications

Getting Started

- Create a new Laravel Application :-

```
$ composer create-project laravel/laravel testapp
```

- To create a new test class, we can either create a new file manually - or run the helpful Artisan make:test command provided by Laravel.
- In order to create a test class called BasicTest, we just need to run this artisan command:

```
$ php artisan make:test BasicTest
```

Laravel Testing Conventions

- The convention for test classes is that they are stored within ./tests/ in your application directory.
  - Inside test folder, each test class is named as <name>Test.php.
  - This format allows PHPUnit to find each test class — it will ignore anything that does not end in Test.php.
- In a new Laravel application, you will notice two files in the ./tests/ directory: ExampleTest.php and TestCase.php.
- The TestCase.php file is a bootstrap file for setting up the Laravel environment within our tests.
- This allows us to use Laravel facades in tests, and provides the framework for the testing helpers, which we will look at shortly.

Structure of Laravel Test Class

```
<?php
class BasicTest extends TestCase
{
public function testExample()
   {
      $this->assertTrue(true);
   }
}
```

Running Tests with PHPUnit

- You can run your PHPUnit tests by running the phpunit command:

```
$ ./vendor/bin/phpunit
```

- If phpunit is not found, then it can easily be installed on Ubuntu using

```
$ sudo apt-get install phpunit
```

- If you are sure the method name is unique you can only filter by method name :-

```
$ phpunit --filter {TestMethodName}
```

- However it is safer to specify the file path/reference as well

```
$ phpunit --filter {TestMethodName} {FilePath}
```

Basic PHPUnit Assertions

- Assertion is a true/false statement for checking some assumption about a unit
- PHPUnit has 90 different types of assertions
- Seven of the basic PHPUnit assertions to write tests are:
  - assertTrue()
  - assertFalse()
  - assertEquals()
  - assertNull()
  - assertContains()
  - assertCount()
  - assertEmpty()

Basic Assertions (Cont’d)

- assertTrue() and assertFalse() allow you to assert that a value equates to either true or false.
  - This means they are perfect for testing methods that return boolean values.
- assertEquals() is used to compare the actual value of the variable to the expected value.
- assertContains() asserts that an expected value exists within the provided array,
- assertCount() asserts the number of items in the array matches the specified amount,
- assertEmpty() asserts that the provided array is empty.

and so on

Example | Box Class

Checkout relevant code for Box Class over here :-

https://semaphoreci.com/community/tutorials/getting-started-with-phpunit-in-laravel

Example 1| assertTrue and assertFalse

```
<?php
use App\Box;
class BasicTest extends TestCase
{
    public function testHasItemInBox()
    {
        $box = new Box(['cat', 'toy', 'torch']);

        $this->assertTrue($box->has('toy'));
        $this->assertFalse($box->has('ball'));
    }
}
```

Example 2 | assertEqual and assertNull

```
public function testTakeOneFromTheBox()
  {
      $box = new Box(['torch']);


      $this->assertEquals('torch', $box->takeOne());


      // Null, now the box is empty
      $this->assertNull($box->takeOne());
  }
```

Example 3 | assertCount, assertContains and assertEmpty

```
public function testStartsWithALetter()
   {
       $box = new Box(['toy', 'torch', 'ball', 'cat', 'tissue']);
       $results = $box->startsWith('t');
       $this->assertCount(3, $results);
       $this->assertContains('toy', $results);
       $this->assertContains('torch', $results);
       $this->assertContains('tissue', $results);
       // Empty array if passed even
       $this->assertEmpty($box->startsWith('s'));
   }
```

Laravel Test Helpers

- When you are building an application that includes complex views, navigation and forms, you will want to test these components too.
- This is where Laravel's test helpers make things just as easy as unit testing simple components.
- Here’s some of the Laravel test helpers :-
  - visit
  - see
  - dontsee
  - click(‘link')
  - seePageIs(‘link');

Example 1 | Using Laravel Test Helpers

Code :

```
public function testBasicExample()
    {
        $this->visit('/')
            ->see('Laravel 5');
    }
```

- Without any prior knowledge of how the test helpers work, we can assume it means something like this:
  - when I visit / (webroot)
  - I should see 'Laravel 5‘
- A simple test like this may prevent you from deploying a broken application if a change somewhere else causes the page to no longer display the right information.

Example 2 | Using Laravel Test Helpers Test Page Content

```
public function testDisplaysAlpha()
    {
        $this->visit('/alpha')
             ->see('Alpha')
             ->dontSee('Beta');
    }
```

- Another basic test, that visits page **/alpha** checks for text **‘Alpha’** and also ensures that pages does not contain text **‘Beta’**

Example 3 | Using Laravel Test Helpers Test Page Links

```
public function testClickNextForBeta()
  {
    $this->visit('/alpha')
        ->click('Next')
        ->seePageIs('/beta');
  }
```

- Another great way to test is to visit a page, click on specific links and check loading of corresponding page !

Example 4 | Using Laravel Test Helpers Test Database

```
public function testDatabase()
  {
    // Make call to application...


    $this->seeInDatabase('users', ['name'=>'john','email' => 'john@clivern.com']);
  }
```

- ➢ Laravel also provides a variety of helpful tools to make it easier to test your database driven applications.
- ➢ As seen above, you may use the seeInDatabase helper to assert that data exists in the database matching a given set of criteria.
- ➢Here data is being checked in user table with columns name and email

Example 5 | Using Laravel Test Helpers Test Session

```
public function testSession()
  {
      $this->withSession(['foo' => 'bar']);
      $this->assertSessionHas('foo','bar');
      $this->assertSessionMissing('foo1');
  }
```

Example 6 | Using Laravel Test Helpers Test Model

- Let write unit test for our Cat model !

```
class Cat extends Model
{
   //
   //
   public function setAgeAttribute($age)
  {
      $this->attributes['age'] = $age * 7;
   }
}
```

Example 6 | Using Laravel Test Helpers Test Model (Cont’d)

```
public function testAutomaticallyCompensatesForCatYears()
    {
        $cat = new Cat;
        $cat->age = 6;
        $this->assertEquals(42, $cat->age); // true
    }
```

References

https://laravel.com/docs/5.2/testing#sessions-and-authentication

https://semaphoreci.com/community/tutorials/getting-started-with-phpunit-in-laravel

Questions