Introduction to PHP Testing

Ran Mizrahi (@ranm8)

What is Wix?

Enables **Everyone** to create **amazing** websites

Need	Primary solution
Create a document	
Create a presentation	
Create a blog	
Create a web presence	WiX

What is Wix?

- Biggest website building platform in the world.
- Everyone - No technical knowledge needed.
- Amazing websites - allow self expression and flexibility.

Wix in Numbers

38,750,000+ Users

45,000+ Join Wix/Day

15,000+ Apps Installations/Day

Wix App Market

- Enables to develop responsive Wix web apps and sell apps to Wix's users.

- Hundreds of apps, each gives our users new amazing functionality.

- Wix App is actually a web app (HTML app) embedded in to a Wix site.

Why Do Software Projects Fail?!

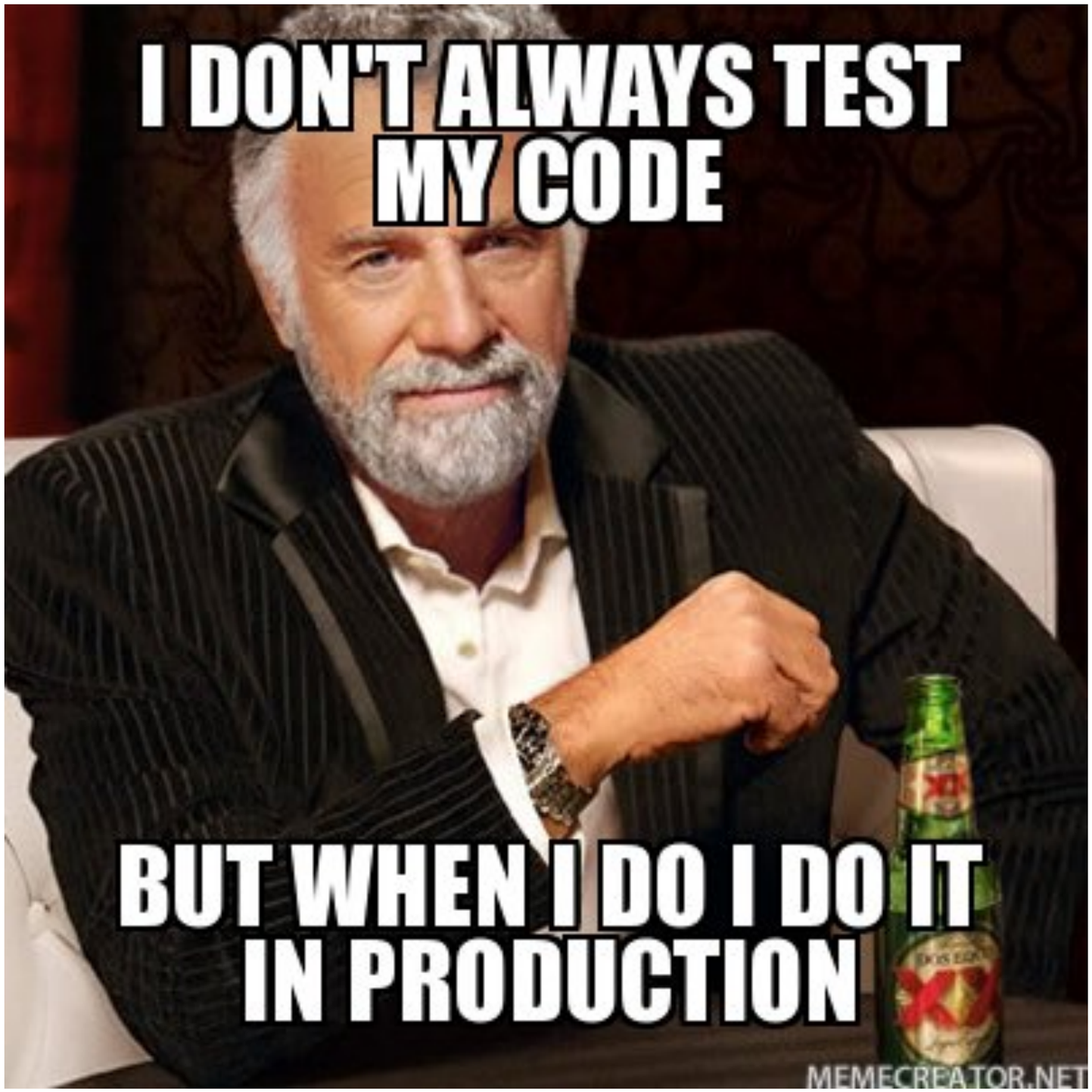
Production

- Deliver late or over budget.
- Deliver the wrong thing.
- Unstable in production.

Maintenance

- Expensive maintenance.
- Long adjustment to market needs.
- Long development cycles.

Why Do Software Projects Fail?!



Untestable code... (Common PHP example)

```
function user($username, $firstName, $lastName, $mail) {

    if (!$firstName || !$lastName) {
        throw new Exception('First or last name are not valid');
    } else if(!is_string($mail) && !filter_var($mail, FILTER_VALIDATE_EMAIL)) {
        throw new Exception('Mail is not valid');
    } else if(!$username) {
        throw new Exception('Username is not valid');
    }

    $userServer = new UserServer();

    try {
        $res = $userServer->exec(array(
            'fullName' => $firstName . ' ' . $lastName,
            'username' => $username,
            'mail' => $mail
        ));

        if ($res->code !== 200) {
            $message = 'Failed saving user to server';
        }

        $message = 'Success!';
    } catch(Exception $e) {
        $message = 'Failed accessing user system';
    }

    echo $message;
}
```

Why Test Your Code???

The problems with untestable code:

- Tightly coupled.
- No separation of concerns.
- Not readable.
- Not predictable.
- Global states.
- Long methods.
- Large classes.

Why Test Your Code???

The problems with untestable code:

- Tightly coupled.
- No separation of concerns.
- Not readable.
- Not predictable.
- Global states.
- Long methods.
- Large classes.

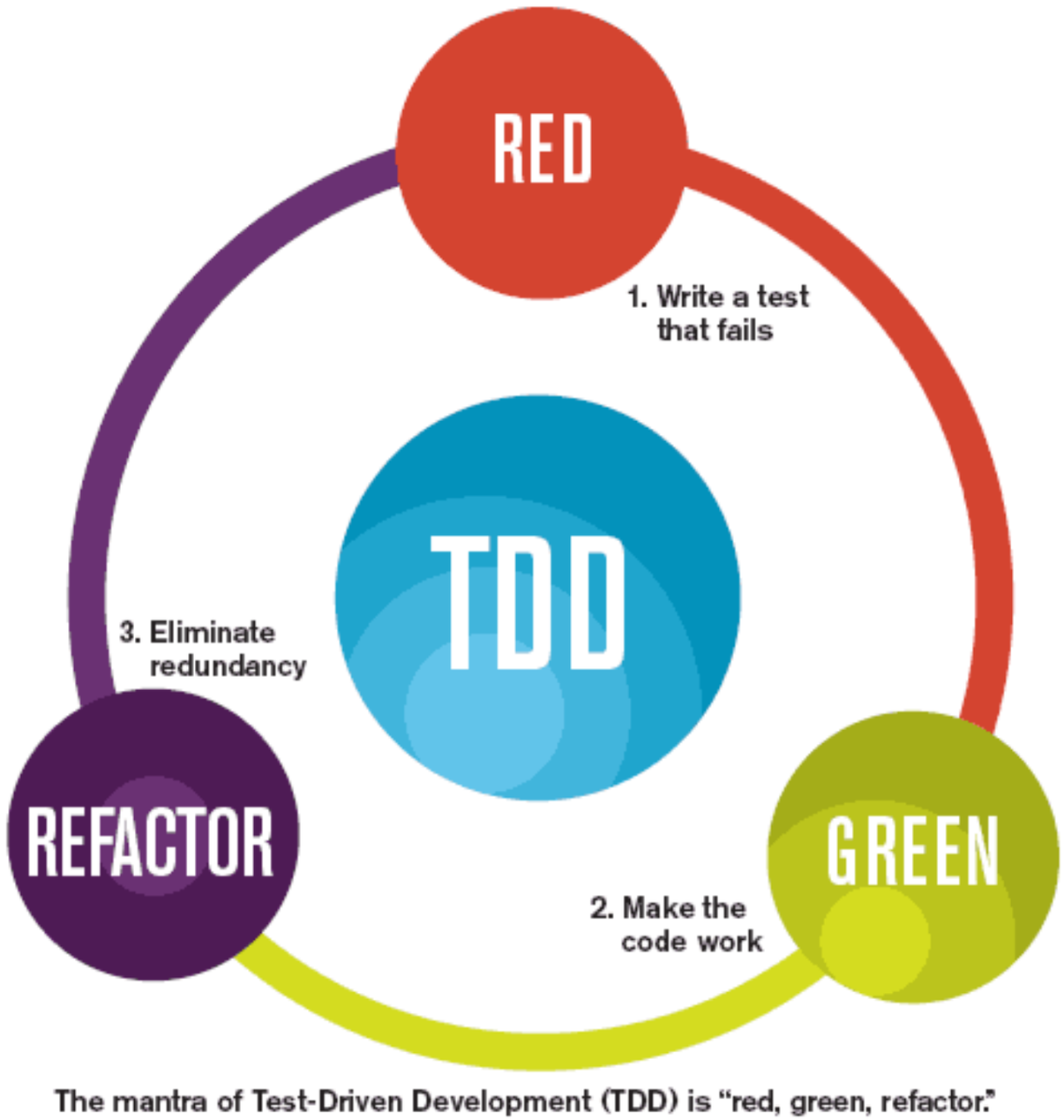
>

- Hard to maintain.
- High learning curve.
- Stability issues.
- You can never expect problems before they occur

Test-Driven Development To The Recuse!

Methodology for using automated unit tests to drive software design, quality and stability.

Test-Driven Development To The Recuse!

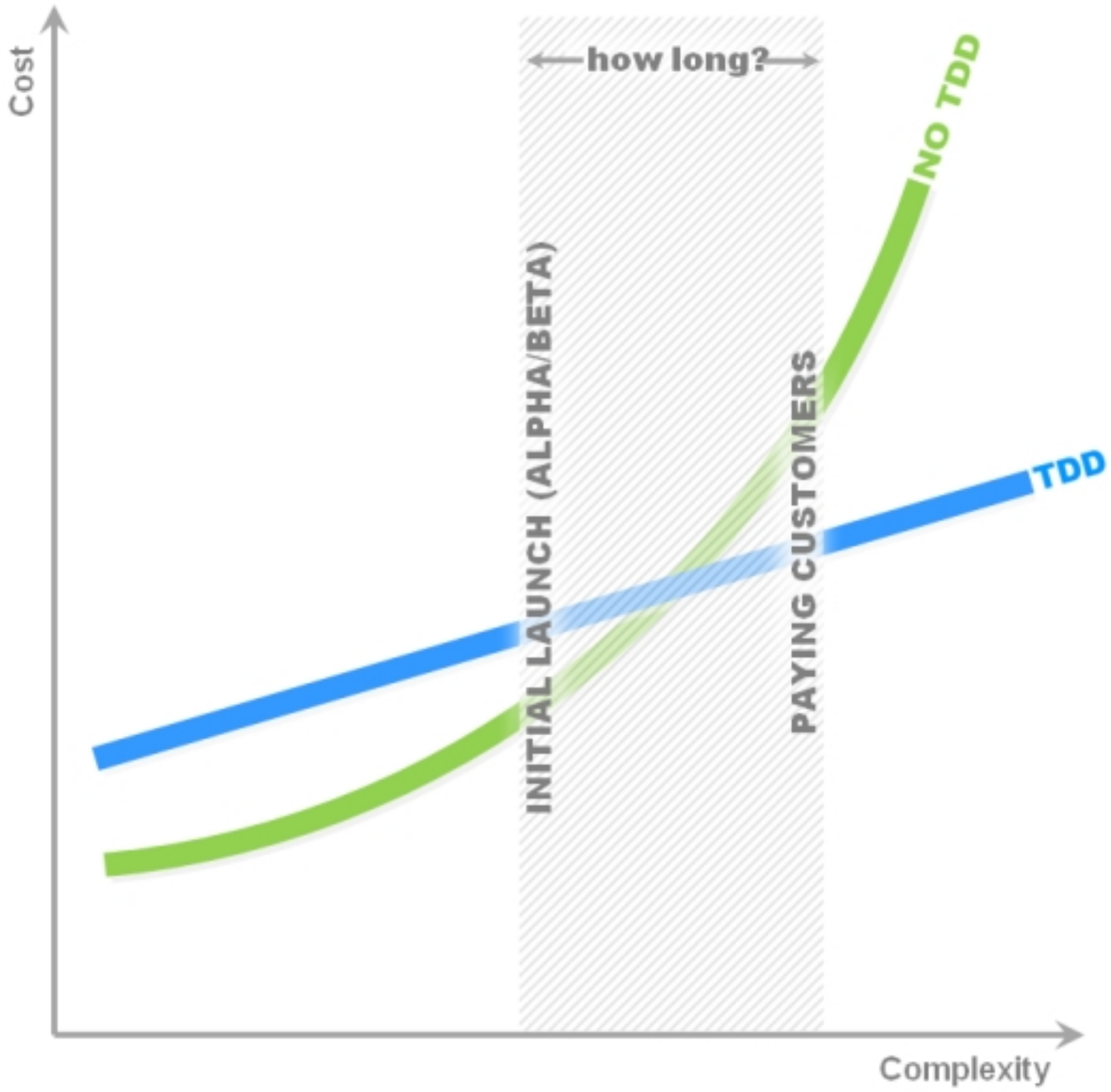


The mantra of Test-Driven Development (TDD) is "red, green, refactor."

How it's done :

- First the developer writes a failing test case that defines a desired functionality to the software.
- Makes the code pass those tests.
- Refactor the code to meet standards.

Seems Great But How Much Longer Does TDD Takes???



My experience:

- Initial progress will be slower.
- Greater consistency.
- Long tern cost is drastically lower
- After getting used to it, you can write TDD faster (-:

Studies:

- Takes 15-30% longer.
- 45-80% less bugs.
- Fixing bugs later on is dramatically faster.

The Three Rules of TDD

Rule #1
Your code should always fail before you implement the code

Rule #2
Implement the simplest code possible to pass your tests.

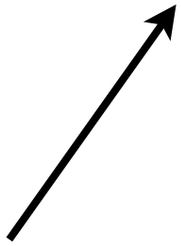
Rule #3
Refactor, refactor and refractor - **There is no shame in refactoring.**

BDD (Behavior-Driven Development)

Test-Driven Development

BDD (Behavior-Driven Development)

What exactly are we testing?!



Test-Driven Development

BDD (Behavior-Driven Development)

- Originally started in 2003 by Dan North, author of JBehave, the first BDD tool.
- Based on the TDD methodology.
- Aims to provide tools for both developers and business (e.g. product manager, etc.) to share development process together.

- The steps of BDD :
  - Developers and business personas write specification together.
  - Developer writes tests based on specs and make them fail.
  - Write code to pass those tests.
  - Refactor, refactor, refactor...

BDD (Behavior-Driven Development)

```
Feature: ls
  In order to see the directory structure
  As a UNIX user
  I need to be able to list the current directory's contents

  Scenario: List 2 files in a directory
    Given I am in a directory "test"
    And I have a file named "foo"
    And I have a file named "bar"
    When I run "ls"
    Then I should get:
      """
      bar
      foo
      """
```

* Behat example

Main Test Types

- Unit Testing
- Integration Testing
- Functional Testing

Challenges Testing PHP

- You cannot mock global functions in PHP since they are immutable.
- Procedural code is hard to mock and be tested.

TDD using PHPUnit

PHPUnit

PHPUnit is a popular PHP unit testing framework designed for making unit testing easy in PHP.

Main features:

- Supports both TDD style assertions.
- Support multiple output formats like jUnit, JSON, TAP.
- Proper exit status for CI support (using jUnit for test results and clover for code coverage report).
- Supports multiple code coverage report formats like HTML, JSON, PHP serialized and clover.
- Built-in assertions and matchers.

TDD using PHPUnit

Installing PHPUnit

Install mocha globally using pear:

```
$ pear config-set auto_discover 1
$ pear install pear.phpunit.de/PHPUnit
```

Install mocha globally using composer:

```
$ composer global require 'phpunit/phpunit=3.8.*'
```

Download the PHP archive:

```
$ wget https://phar.phpunit.de/phpunit.phar
$ chmod +x phpunit.phar
$ mv phpunit.phar /usr/local/bin/phpunit
```

TDD using PHPUnit

Basic test:

```php
class UserSpec extends PHPUnit_Framework_TestCase
{
    protected $invalidMail = 'invalid-mail';

    public function testThatFilterVarReturnsFalseWhenInvalid()
    {
        $this->assertFalse(filter_var($this->invalidMail, FILTER_VALIDATE_EMAIL));
    }
}
```

Run it..

```
$ phpunit UserTest.php
    PHPUnit 3.7.22 by Sebastian Bergmann.

    ...

    Time: 0 seconds, Memory: 2.50Mb

    OK (1 test, 1 assertion)
```

Back to our code

First, Let's Write The Tests!

```
function user($username, $firstName, $lastName, $mail) {

    if (!$firstName || !$lastName) {
        throw new Exception('First or last name are not valid');
    } else if(!is_string($mail) && !filter_var($mail, FILTER_VALIDATE_EMAIL)) {
        throw new Exception('Mail is not valid');
    } else if(!$username) {
        throw new Exception('Username is not valid');
    }

    $userServer = new UserServer();

    try {
        $res = $userServer->exec(array(
            'fullName' => $firstName . ' ' . $lastName,
            'username' => $username,
            'mail' => $mail
        ));

        if ($res->code !== 200) {
            $message = 'Failed saving user to server';
        }

        $message = 'Success!';
    } catch(Exception $e) {
        $message = 'Failed accessing user system';
    }

    echo $message;
}
```

First, Let's Write The Tests!

What to test in our case:

- Validations.
- Full name getter.
- Post user save actions.

What not to test :

- UserServer object (-: we should isolate our code and test it only.

First, Let's Write The Tests!

```
class UserSpec extends PHPUnit_Framework_TestCase
{
    protected $user;

    protected $userServerMock;

    protected $goodResponse;

    public function setUp()
    {
        $this->goodResponse = new stdClass();
        $this->goodResponse->code = 200;

        $this->userServerMock = $this->getMock('UserServer', array('exec'));

        $this->user = new User('anakin', 'Anakin', 'Skywalker', 'anakin@wix.com', $this-
>userServerMock);
    }

   /**
    * @expectedException Exception
    */
    public function testThatInvalidMailThrows()
    {
        $this->user->setMail('invalid-mail');
    }

    public function testThatValidMailWorks()
    {
        $validMail = 'me@mail.com';
        $this->user->setMail($validMail);

        $this->assertEquals($validMail, $this->user->getMail());
    }
```

First, Let's Write The Tests!

```
public function testThatUserServerExecIsCalledWhenSavingAUser()
{
    $this->userServerMock->expects($this->once())
        ->method('exec')
        ->will($this->returnValue($this->goodResponse));

    $this->user->save();
}

public function testThatFullNameIsConcatinatedCorrcetly()
{
    $this->userServerMock->expects($this->once())
        ->method('exec')
        ->with($this->callback(function($data) {
            return $data['fullName'] === 'Anakin Skywalker';
        }))
        ->will($this->returnValue($this->goodResponse));

        $this->user->save();
    }
}
```

Run those tests

Now, Let's Write The Code

```php
class User
{
    protected $username;

    protected $firstName;

    protected $lastName;

    protected $mail;

    protected $userServer;

    public function __construct($username, $firstName, $lastName, $mail, $userServer)
    {
        $this->username = $username;
        $this->firstName = $firstName;
        $this->lastName = $lastName;
        $this->setMail($mail);

        $this->userServer = $userServer;
    }

    public function setMail($mail)
    {
        if (!filter_var($mail, FILTER_VALIDATE_EMAIL)) {
            throw new Exception('Mail is not valid');
        }

        $this->mail = $mail;

        return $this;
    }
```

Now, Let's Write The Code

```
public function getMail() {
    return $this->mail;
}

protected function getFullName()
{
    return $this->firstName . ' ' . $this->lastName;
}

public function getUsername()
{
    return $this->username;
}

public function save() {
    $response = $this->userServer->exec(array(
        'fullName' => $this->getFullName(),
        'mail' => $this->getMail(),
        'username' => $this->getUsername()
    ));

    if (!$this->isSuccessful($response->code)) {
        return 'Failed saving to user server';
    }

    return 'Success!';
}

protected function isSuccessful($code)
{
    return $code >= 200 && $code < 300;
}
}
```

Run those tests, again!

Running The Tests

PHPUnit tests can run and provide different types of results:

- CLI - as demonstrated before using the “phpunit” command.
- CI (e.g. jUnit) - (e.g. $ phpunit --log-junit=[file]).
- HTML||clover code coverage (e.g. $ phpunit --coverage-html=[dir]).
- Many other formats (JSON, TAP, Serialized PHP and more).

Benefits of Testing Your Code

- Short feedback/testing cycle.
- High code coverage of tests that can be at run any time to provide feedback that the software is functioning.
- Provides detailed spec/docs of the application.
- Less time spent on debugging and refactoring.
- Know what breaks early on.
- Enforces code quality (decoupled) and simplicity.
- Help you focus on writing one job code units.

Thank you!

Questions?