

Unit and Acceptance Testing

Alex Ward

ONESite inc.

<award@onesite.com> or <daginus@gmail.com>

Jake Farrell

ONESite inc.

<jfarrell@onesite.com>



Unit Testing, what's the point?

- Keeps you from repeating past mistakes
- Ensures deployments will go smoothly by detecting problems before they occur
- Changing one part of the code can have unexpected effects on other parts of the code.
- Keeps you high and DRY (Don't repeat yourself) by reducing bugs.



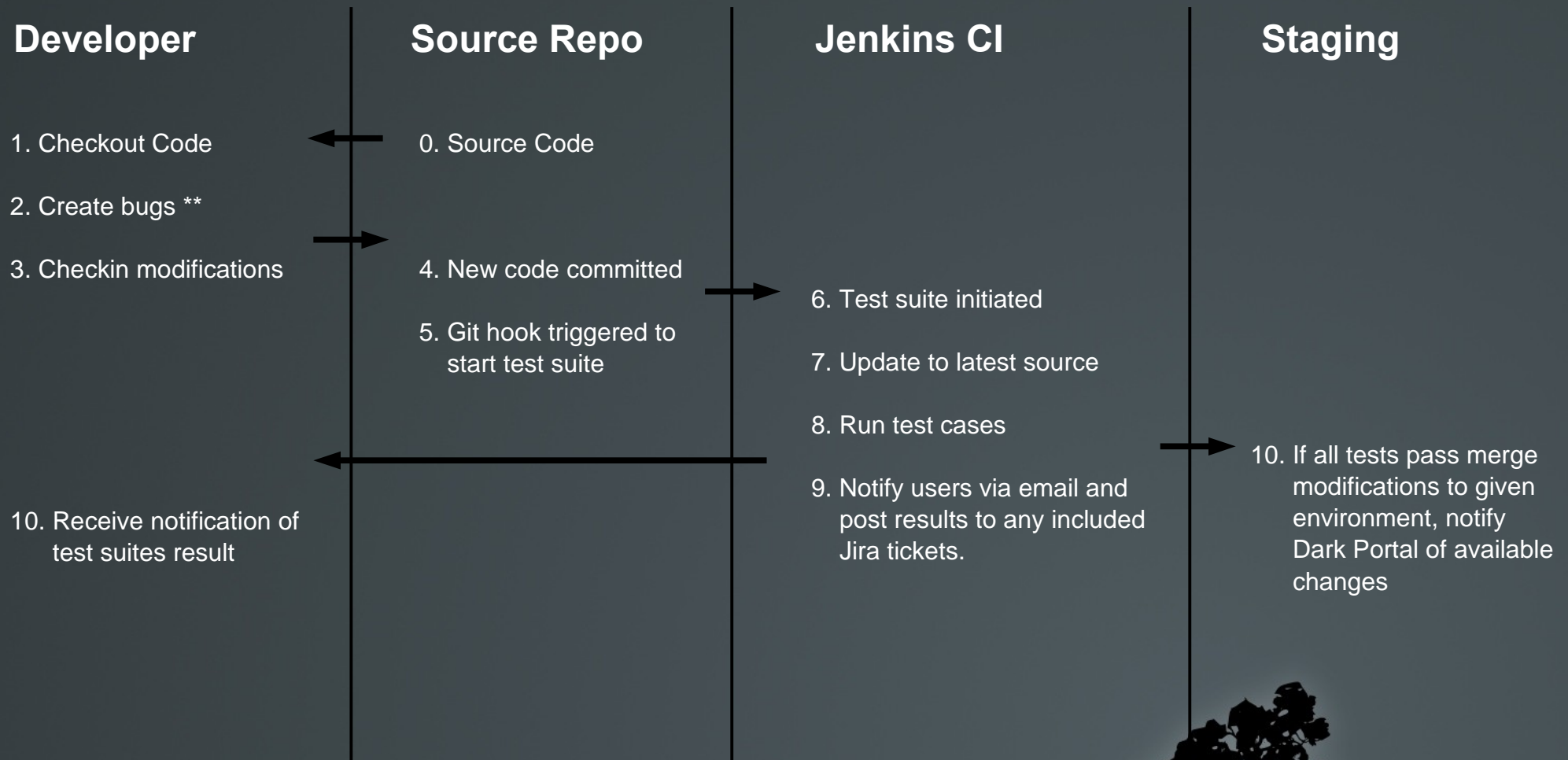
Continuous Integration

- Continually applying quality control to source code by testing during the development procedure.
- Allows for automated build/testing/deployment to reduce bugs and increase rate of production.
- Know a rollout is going to break something before the customer does!
- http://en.wikipedia.org/wiki/Continuous_integration






The butler, Jenkins





** <http://bit.ly/uZ728u>


Jenkins in Action

Jenkins

 [New Job](#)

 [People](#)

 [Build History](#)

 [Manage Jenkins](#)

Build Queue



No builds in the queue.

Build Executor Status

#	Status
1	Idle
2	Idle


All

+


S	W	Name ↓	Last Success	Last Failure	Last Duration
		Onesite Unit Test Demo	17 min (#5)	N/A	79 ms


Icon: [S](#) [M](#) [L](#)


[ENABLE AUTO REFRESH](#)


 [add description](#)

[Legend](#)

 [RSS for all](#)

 [RSS for failures](#)

 [RSS for just latest builds](#)

 [Help us localize this page](#)

Page generated: Dec 21, 2011 12:25:46 PM

[Jenkins ver. 1.443](#)

Is it really necessary?

- You can make perfectly deployable code without ever having run a unit test against it.
- Manual testing can also help detect bugs.
- Manual testing will never catch every situation.
- Neither will unit testing.
- A combination of testing strategies is needed to ensure bug minimization.



Is it really necessary?

- By implementing Unit Testing, Selenium Testing, a good QA Strategy, and Continuous integration bugs can be minimized to acts of whichever deity you might believe in.
- This leads to greater happiness among developers who don't have to do as many bugfixes, and clients who have working products



Great, how do I do this?

- The hardest part of writing unit tests is figuring out how it could break.
 - The developer who wrote the system is likely to not to know all the places in which their application can break (due to familiarity with the code)
 - Pair programming for the test writing phase can help mitigate this problem
- Writing the tests is the easier part



Unit Testing Tools

- For PHP, phpunit is the standard testing framework
 - It is a pain to install, instead, Jake Farrell (@eatfresh) has forked phpunit and made all its dependencies git submodules. (It's awesome).
 - <https://github.com/jfarrell/phpunit>
 - `git clone git://github.com/jfarrell/phpunit.git`
- Documentation for phpunit is available online
 - <http://www.phpunit.de/manual/3.5/en/writing-tests-for-phpunit.html>



Test Writing Process

- Determine how your application is supposed to operate, and where it might fall apart
- Write tests to ensure that it does what it is supposed to do, and then write tests to ensure it doesn't break under various circumstances.
- Also remember to write tests to ensure it fails when it is supposed to.



Testing Examples (Demo site)

Covered in detail on the demonstration site

```
<?php
class StringMeth
{
    /**
     * Returns only the vowels in the given string.
     *
     * @param string $string
     *
     * @return string
     */
    public static function getVowels($string)
    {
        $vowels = preg_replace("/[^aeiou]/i", '', $string);

        return preg_split('//', $vowels, -1, PREG_SPLIT_NO_EMPTY);
    }

    /**
     * Takes the string and changes spaces into hyphens, and lowercases
     * the whole thing.
     *
     * @param String $string
     *
     * @return string
     */
    public static function slugify($string)
    {
        $string = trim($string);
        $string = preg_replace('/^[^a-zA-Z0-9 ]/', '', $string);
        $replaces = preg_replace("/ +/", '-', $string);

        return strtolower($replaces);
    }
}
```

Testing Examples (Demo Site)

```
<?php
require_once dirname(__FILE__) . "/../application/libraries/stringmeth.php";

class StringMethTest extends PHPUnit_Framework_TestCase
{
    public function testStripConsonants()
    {
        $string = "Hello";
        $vowels = array(
            'e',
            'o',
        );

        $this->assertEquals(StringMeth::getVowels($string), $vowels);
    }

    public function testSlugify()
    {
        $string = "Cookies are delicious";
        $expected = 'cookies-are-delicious';

        $this->assertEquals(StringMeth::slugify($string), $expected);

        $string = "Hi Mom! I like muffins!";
        $expected = 'hi-mom-i-like-muffins';

        $this->assertEquals(StringMeth::slugify($string), $expected);

        $string = "Hey there billy, I like cheese. ";
        $expected = "hey-there-billy-i-like-cheese";
        $this->assertEquals(StringMeth::slugify($string), $expected);
    }
}
```


Testing Examples (Demo Site)

```
cthos@cthos-desktop ~/Documents/Code/onesite-unit-test-demo $ php ../phpunit/phpunit.php tests/
PHPUnit @package_version@ by Sebastian Bergmann.

.F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) StringMethTest::testSlugify
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-hey-there---billy-i-like-cheese--
+hey-there-billy-i-like-cheese

/home/cthos/Documents/Code/onesite-unit-test-demo/tests/StringMethTest.php:34

FAILURES!
Tests: 2, Assertions: 4, Failures: 1.
cthos@cthos-desktop ~/Documents/Code/onesite-unit-test-demo $
```

Testing Examples (ONESite)

IFrame Scrubber

```
public function testIframeRemoved()
{
    $scrubber = new one_security_InputScrubber();

    $text = '<iframe width="420" height="315" src="http://www.boop.com/embed/iHGUpxtfcoc" frameborder="0" allowfullscreen></iframe>';

    $scrubber->strip_text($text);

    $this->assertEquals('iFrame Removed', $text);
}

/**
 * Tests that youtube videos are are not scrubbed.
 *
 * @return void
 */
public function testYoutubeRemains()
{
    $scrubber = new one_security_InputScrubber();

    $text = $old_text = '<iframe width="420" height="315" src="http://www.youtube.com/embed/iHGUpxtfcoc" frameborder="0" allowfullscreen></iframe>';

    $scrubber->strip_text($text);

    $this->assertEquals($old_text, $text);
}

/**
 * Tests that the above behavior functions when there are more than one
 * iframe in the mix.
 *
 * @return void
 */
public function testMultiIframe()
{
    $scrubber = new one_security_InputScrubber();

    $text = $old_text = '<iframe width="420" height="315" src="http://www.youtube.com/embed/iHGUpxtfcoc" frameborder="0" allowfullscreen></iframe>
    <iframe width="420" height="315" src="http://www.onesite.com/embed/iHGUpxtfcoc" frameborder="0" allowfullscreen></iframe>
    <iframe width="420" height="315" src="http://www.branmuffins.com/embed/iHGUpxtfcoc" frameborder="0" allowfullscreen></iframe>';

    $scrubber->strip_text($text);

    $this->assertNotEquals($old_text, $text);
    $this->assertEquals(strpos($text, 'branmuffins.com'), false);
    $this->assertNotEquals(strpos($text, 'youtube.com'), false);
}
```


Testing Examples (ONEsite)

Test Results

```
[award@dev01 test]$ phpunit Security/BC
PHPUnit 3.5.13

Time: 1 second, Memory: 26.75Mb

There was 1 failure:

1) InputScrubberTest::testLegacyYoutubeRemains
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
- <iframe width="420" height="315" src="http://www.youtube.com/embed/iHGUpxtfcoc" frameborder="0" allowfullscreen></iframe>
+iFrame Removed

/onesite/devspace/award/onesite/test/Security/InputScrubberTest.php:91

FAILURES!
Tests: 7, Assertions: 10, Failures: 1.
[award@dev01 test]$
```

I failed at this test, because the current branch I am on does not have all the appropriate patching done from the main branch. Which makes for a good example of seeing a unit test fail.



Unit Testing can be Complex

- Once you have to start taking various dependencies into account, such as reading/writing from a database, interacting across multiple subsystems, tests can become complicated
- Dependency injection is a way to get dummy dependencies into your objects so you can test them without worry.



ONESite Unit Testing

- To handle complex dependencies we:
 - Have a development database which can have tests done against it without consequence
 - Can override network options on our testing server to test edge cases
 - Have automated scripts to push any database schema changes across environments.
 - Dark Portal/Jenkins integration



Selenium IDE Testing

- A weakness of phpunit is that it is unable to test how a user would interact with a live site.
- Selenium IDE is a Firefox plugin which allows you to write tests which will simulate user interaction and allow you to test javascript flows.
- <http://seleniumhq.org/projects/ide/>
- It also comes with a way to interface with a language directly, to test php server responses.



What is Selenium Good at?

- Testing any flow that relies on javascript
 - Ensuring popups display
 - Social Login flows
- Complex, multi-page flows which relies heavily on ux
 - Making sure elements on the page display and are visible



Selenium Example

selenium-example.html - Selenium IDE 1.5.0

Base URL:

Fast Slow

Test Case

selenium-example

Command	Target	Value
open	http://test.onsite.com	
verifyText	link=Download Presentation	Download Presentation
clickAndWait	link=Cucumber	
verifyTextPres...	exact:1: Describe behaviour in plain text	

Command

Target

Value

Find

Runs: 1

Failures: 0

Log Reference UI-Element Rollup

Info Clear

[info] Executing: |open | http://test.onsite.com | |

[info] Executing: |verifyText | link=Download Presentation | Download Presentation |

[info] Executing: |clickAndWait | link=Cucumber | |

[info] Executing: |verifyTextPresent | exact:1: Describe behaviour in plain text | |

ONESite uses for Selenium

- Testing the social login widget, to ensure that all social login providers appear to function properly
- Ensuring feedback tab expands and collapses as required
- Ensuring that links on a page actually go to their appropriate locations
- Automated creation of blog posts and other user content.



TDD (Test Driven Development)

- TDD changes up the order in which you write unit tests.
- Write your tests first. They will fail. Then write your application in order to make those tests pass
 - Make sure you do this in small chunks, it is designed for continuous progress.
 - http://en.wikipedia.org/wiki/Test-driven_development



Acceptance Testing

- Acceptance testing is the corrolary to Unit Testing
- Where Unit Testing is designed to ensure you write your code correctly, Acceptance testing is designed to ensure you have written the correct code
 - As in, have met the requirements for the project.



Cucumber

- Cucumber is a BDD (Behavior Driven Development) tool designed to make acceptance testing awesome
- Write domain specific language, that looks like plain english.
- Write your application to make those tests pass
- Everyone gets exactly what they're expecting



Cucumber

Covered in detail on the demo site.

```
cthos@cthos-desktop ~/Documents/Code/onesite-unit-test-demo $ cucumber
```

```
Feature: Visit the Presentation Page
```

```
  In order to share my presentation with awesome people
```

```
  I need to make sure that the page is visible with all required info
```

```
  Scenario: Download Link Present
```

```
    Given that I have opened "http://test.onesite.com/presentation/index"
```

```
    When I click on the link "Download Presentation"
```

```
    Then I download a file called "presentation.odp"
```

```
# features/presentation.feature:5
```

```
# features/presentation.feature:6
```

```
# features/presentation.feature:7
```

```
# features/presentation.feature:8
```

```
1 scenario (1 undefined)
```

```
3 steps (3 undefined)
```

```
0m0.001s
```

You can implement step definitions for undefined steps with these snippets:

```
Given /^that I have opened "([^"]*)"$/ do |arg1|
```

```
  pending # express the regexp above with the code you wish you had
end
```

```
When /^I click on the link "([^"]*)"$/ do |arg1|
```

```
  pending # express the regexp above with the code you wish you had
end
```

```
Then /^I download a file called "([^"]*)"$/ do |arg1|
```

```
  pending # express the regexp above with the code you wish you had
end
```

If you want snippets in a different programming language,
just make sure a file with the appropriate file extension
exists where cucumber looks for step definitions.

```
cthos@cthos-desktop ~/Documents/Code/onesite-unit-test-demo $
```