WAL Consulting

Testing Framework Notes

Software Engineers:

Will Tollefson, Luke Klinker, Anu Ghimire

For our automated testing, we used PHPUnit, which is by far the most popular unit testing framework for PHP. We also set this up on our Jenkins continuous integration server.

It was difficult to set this framework up, which wasn't something we were concerned about when initially decided on it. The difficulty wasn't in actually writing the tests, our difficulty with this came from getting the PHPUnit to run with our project. We didn't really know anything about the project structure for implementing the tests.

Most of what we did was just basic unit tests. After getting the setup done, it ended up being very easy to add these tests to our code. One drawback for these tests is that you couldn't test the php code that you added directly into the html. Since you can integrate php directly with your html and even echo back html from within the php code, it felt weird that this type of thing isn't possible. We worked around this issue by just creating object classes for our html pages. That way, on the actual html page, you just have to create an object, then all of the php code is run from within the object. We were then able to run the automated tests against the objects we created.

Another issue with PHPUnit was that, by default, it doesn't have much logging output on a failed test. There was another external plugin that you would have had to use to get the logging working along with the tests. This, of course, would have taken more time to implement and wasn't something we felt the need to add.

Overall, after the setup, PHPUnit was pretty simple to manage and use. We wish that there would have been easier to use with the html integration and logging, but it is a tool I would recommend to a co-worker for automated testing in PHP.

An example testing class can be seen below. These are some tests showing the validation of a password:

```
<?php

class PasswordUtilsTest extends PHPUnit_Framework_TestCase {

	function test_confirmPasswordsTrue() {
		$result = PasswordUtils::checkMatchingPasswords("test", "test");
		$this->assertTrue($result);
	}

	 function test_confirmPasswordsFalse() {
		$result = PasswordUtils::checkMatchingPasswords("test", "fail");
		$this->assertFalse($result);
	 }

	function test_validPassword() {
		 $result = PasswordUtils::testPassword("longPasswordlongPassword");
		$this->assertEquals($result, "Password cannot be longer than 20 characters.");
		$result = PasswordUtils::testPassword("password");
		$this->assertEquals($result, "Password must have at least one number.");
		$result = PasswordUtils::testPassword("123456789");
		$this->assertEquals($result, "Password must have at least one letter.");
	}
```