UNIT TESTING IN PRACTICE

SCOTT GRANT
ABOUT ME

- Scott Grant (hi!)
- Startup CTO in Kingston, Ontario
- Formerly @ Electronic Arts, Google
- Formerly Adjunct Prof. @ Queen's University

- Prof. James R. Cordy (CISC327: Software Quality Assurance)
WHAT IS SYSTEMATIC TESTING?
WHAT IS SYSTEMATIC TESTING?

• An explicit discipline or procedure (a system) for:
  • choosing and creating test cases
  • executing the tests and documenting the results
  • evaluating the results, possibly automatically
  • deciding when we are done (enough testing)
UNIT TESTS

• Units are, preferably, the smallest testable part of an application

• Unit tests are designed to test small pieces of the code (units) independently

```php
function sum( $a, $b ) {
    return $a + $b;
}

sum( 1, 1 ) == 2

$this->assertEquals( 2, sum( 1, 1 ) );
```
Assertions

• In many ways, the heart of a unit test

• Checks for a predefined constraint about the program state
  • "After sum is called with the arguments 1 and 1, it will return the value 2."

• If the constraint does not hold, the test fails

• The test asserts that the code is correct for a particular context if the assertion holds
PHPUNIT

- A programmer-oriented testing framework for PHP
- Provides a framework for creating and executing unit tests easily, quickly, and reproducibly
- Based on atomic unit tests with assertions
- Open-source!

- https://github.com/sebastianbergmann/phpunit
PHPUnit 4.2.2 by Sebastian Bergmann.

Configuration read from /Users/scott/Sites/arcadia/test/phpunit.xml.dist

...............................................................  63 / 110 ( 57%)
.............................................

Time: 404 ms, Memory: 6.00Mb

OK (110 tests, 166 assertions)
1. CHOOSING AND CREATING TEST CASES
WHAT MAKES A GOOD TEST?

- **Independent:** Each test needs to run independently from other tests and environments.
- **Fast:** To have useful tests and to be able to run them as often as possible, tests need to be fast.
- **Repeateable:** You should be able to run a test as many times as you want with the same result.
WHAT MAKES A GOOD TEST?

• **Up to date**: Tests are written once, but code can be changed or extended

• **Short**: Tests should be just a few lines--easy to read and understand

• **Resilient**: Once written, tests shouldn't change until the behaviour of tested class/method changes
TESTS IN ISOLATION

• Before each test, initialize the appropriate state

• Next, perform the test

  • One test! One!!

• After each test, clean up after yourself

  • There are more tests after this one, we don't want to corrupt them with our test data
function is_even( $x ) {
    if ( 0 == $x % 2 ) {
        return true;
    }
}

return false;

function is_odd( $x ) {
    // ??
}
class Test_Even extends WP_UnitTestCase {

    public function test_even_with_even() {
        $this->assertTrue( is_even( 2 ) );
    }

    public function test_even_with_odd() {
        $this->assertFalse( is_even( 1 ) );
    }

}
CHOOSING & ORGANIZING TESTS

• The experimental model of software testing gives us two important principles for our test plan:

  • Test inputs should be chosen to carefully isolate different causes of failure (the experimental variables)

  • Test cases should be ordered such that each test only assumes features to be working that have already been tested by a previous test
function is_odd( $x ) {
    return ! is_even( $x );
}
function is_odd( $x ) {
    $remainder = $x % 2;
    if ( $remainder = 1 ) {
        return true;
    }
    return false;
}
2. EXECUTING THE TESTS AND DOCUMENTING THE RESULTS
TEST REPORTS

• Output of test execution should be summarized in a readable report

• Test reports should be designed to be concise, easy to read, and to clearly point out failures or unexpectedly changed results

• Test results should be in a standardized form for easy comparison with future test executions
PHPUnit 4.5.0 by Sebastian Bergmann and contributors.

Configuration read from /tmp/wordpress/src/wp-content/plugins/two-factor/phpunit.xml

....................................
..........................

Time: 11.37 seconds, Memory: 71.50Mb

OK (53 tests, 90 assertions)
PHPUnit 4.4.0 by Sebastian Bergmann.

Configuration read from /home/travis/build/scotchfield/arcadia/test/phpunit_travis.xml.dist

............................
............................
............................
F
............................
............................

Time: 2.52 seconds, Memory: 9.50Mb

There was 1 failure:

1) TestArcadiaUser::test_get_user_by_name_simple
Failed asserting that false is not false.

/home/travis/build/scotchfield/arcadia/test/tests/include/test_user.php:43

FAILURES!
3. EVALUATING THE RESULTS, POSSIBLY AUTOMATICALLY
TEST RUN SUCCESS

• When has a test run successfully completed?
  • When all of the unit tests pass!

• When does a unit test pass?
  • When all of the assertions hold!

• When does an assertion hold?
  • When that individual piece of expected behaviour captured in the assertion holds!
SAFETY BLANKET

• When a programmer makes a change to the code (any change), they can run the test suite

• If all of the tests pass, and we believe the test suite covers enough functionality, we're fairly confident that the code works

• If a test fails, we might have broken something (or exposed a bug!), and can fix it before checking in
AUTOMATED TESTING

• A continuous build machine can run unit tests after every commit
  • Useful even if a programmer isn't proactively running unit tests

• Catch bugs as they're introduced
  • Or provide confidence that the code isn't going to cause problems
This pull request can be automatically merged by project collaborators.

Only those with write access to this repository can merge pull requests.

```
$ phpunit
Installing...
Running as single site... To run multisite, use -c tests/phpunit/multisite.xml
Not running ajax tests... To execute these, use --group ajax.
PHPUnit 4.2.4 by Sebastian Bergmann.

Configuration read from /tmp/wordpress/src/wp-content/plugins/jetpack/phpunit.xml.dist

......................................................... 63 / 177 ( 35%)
......................................................... 126 / 177 ( 71%)
.........................................................

Time: 7.8 seconds, Memory: 45.25Mb

OK (110) tests, 0 failures, 0 errors.

The command "phpunit" exited with 0.

Done. Your build exited with 0.
```
4. DECIDING WHEN WE ARE DONE (ENOUGH TESTING)
WHEN ARE WE DONE?

- When we've achieved 100% code coverage
- Not an ideal statistic
  - Does not check for input, output, or functionality coverage
- Hard to achieve in practice
- However, it's a clear and unambiguous goal
CODE COVERAGE

• PHPUnit has a set of --coverage-* options

• Generate code coverage reports in various formats

• Checks which lines of code are covered by at least one unit test

• In addition, functions and classes

• Does not check for input, output, or functionality coverage
### Code Coverage

<table>
<thead>
<tr>
<th></th>
<th>Lines</th>
<th>Functions and Methods</th>
<th>Classes and Traits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total</strong></td>
<td>96.75%</td>
<td>83.33%</td>
<td>0.00%</td>
</tr>
<tr>
<td><strong>roller.php</strong></td>
<td>96.75%</td>
<td>83.33%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

**Legend**

- **Low**: 0% to 50%
- **Medium**: 50% to 90%
- **High**: 90% to 100%

Generated by **PHP_CodeCoverage 2.2.2** using **PHP 5.6.10** and **PHPUnit 4.8.6** at Fri Oct 2 2:48:25 UTC 2015.
$this->ensure_lists();

if (isset($_POST['new_list'])) {
    if (false === $this->get_list($_POST['list_id'])) {
        $this->update_list($_POST['list_id'], '');
    }
}

if (isset($_POST['update_list'])) {
    if (false !== $this->get_list($_POST['list_id'])) {
        $this->update_list($_POST['list_id'], $_POST['list_value']);
    }
}
<table>
<thead>
<tr>
<th>File</th>
<th>Lines</th>
<th>Percentage</th>
<th>Total Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>50.83%</td>
<td>306 / 602</td>
<td></td>
</tr>
<tr>
<td>achieve.php</td>
<td>97.06%</td>
<td>33 / 34</td>
<td></td>
</tr>
<tr>
<td>buff.php</td>
<td>4.84%</td>
<td>3 / 62</td>
<td></td>
</tr>
<tr>
<td>common.php</td>
<td>36.04%</td>
<td>40 / 111</td>
<td></td>
</tr>
<tr>
<td>cron.php</td>
<td>84.62%</td>
<td>11 / 13</td>
<td></td>
</tr>
<tr>
<td>db.php</td>
<td>90.91%</td>
<td>20 / 22</td>
<td></td>
</tr>
<tr>
<td>game.php</td>
<td>100.00%</td>
<td>3 / 3</td>
<td></td>
</tr>
<tr>
<td>game_meta.php</td>
<td>100.00%</td>
<td>35 / 35</td>
<td></td>
</tr>
<tr>
<td>heartbeat.php</td>
<td>63.33%</td>
<td>19 / 30</td>
<td></td>
</tr>
<tr>
<td>item.php</td>
<td>10.71%</td>
<td>3 / 28</td>
<td></td>
</tr>
<tr>
<td>log.php</td>
<td>0.00%</td>
<td>0 / 3</td>
<td></td>
</tr>
<tr>
<td>mail.php</td>
<td>55.56%</td>
<td>5 / 9</td>
<td></td>
</tr>
<tr>
<td>plugin.php</td>
<td>100.00%</td>
<td>31 / 31</td>
<td></td>
</tr>
<tr>
<td>predicate.php</td>
<td>38.10%</td>
<td>8 / 21</td>
<td></td>
</tr>
<tr>
<td>quest.php</td>
<td>55.56%</td>
<td>5 / 9</td>
<td></td>
</tr>
<tr>
<td>user.php</td>
<td>46.70%</td>
<td>85 / 182</td>
<td></td>
</tr>
<tr>
<td>zone.php</td>
<td>55.56%</td>
<td>5 / 9</td>
<td></td>
</tr>
<tr>
<td>File</td>
<td>Lines</td>
<td>Functions and Methods</td>
<td>Classes and Traits</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>-----------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td></td>
<td>87.97%</td>
<td>87.97%</td>
<td>68.42%</td>
</tr>
<tr>
<td>achieve.php</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>buff.php</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>common.php</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>component.php</td>
<td>25.00%</td>
<td>66.67%</td>
<td>0.00%</td>
</tr>
<tr>
<td>cron.php</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>db.php</td>
<td>96.43%</td>
<td>87.50%</td>
<td>0.00%</td>
</tr>
<tr>
<td>game.php</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>game_meta.php</td>
<td>86.05%</td>
<td>85.71%</td>
<td>0.00%</td>
</tr>
<tr>
<td>heartbeat.php</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>inventory.php</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>item.php</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>log.php</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>login.php</td>
<td>93.10%</td>
<td>80.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>mail.php</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>npc.php</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>quest.php</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>tracking.php</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>user.php</td>
<td>66.84%</td>
<td>66.67%</td>
<td>0.00%</td>
</tr>
<tr>
<td>zone.php</td>
<td>55.00%</td>
<td>50.00%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>
Unit testing is pretty cool.

Building unit tests helps you grow as a software developer.

Systematic unit testing is a valuable practical skill.

Mega Man 2 is a great game.