CODEWORKS 2010

SEATTLE • PORTLAND • AUSTIN • BALTIMORE • ORLANDO

Unit Testing Strategies

D. Keith Casey Jr

Chief Stuff Breaker/Blue Parabola

Overview

What is Unit Testing?

What's the point of Unit Testing?

Our Holy Grail

Useless Unit Testing

Deciding where to write Unit Tests

Some Useful Unit Tests

So.. who are you?

- D. Keith Casey, Jr.
  - Chief Stuff Breaker, Blue Parabola
  - I break stuff with the underlying goal of understanding and making it better
  - Help organize php|tek, former contrib to the DCPHP, now with AustinPHP & Flash
  - Web2project Head Custodian

What is Unit Testing?

- Unit Testing is the process where individual blocks, functions, methods, or “units” of code are tested individually
- A pre-determined set of input is used to generate an output which is compared against an expected output
- Different from “Integration Testing” where the interaction between structures is tested

What is Unit Testing?

- In plain terms:
  - Unit Testing lets you know *when* your code breaks. *Not if, when.*
  - Great when you're refactoring and the guts change but the result shouldn't
  - Great for reproducing errors when you know particular inputs break the code

What is our goal?

- 100% code coverage
  - It means that we have a test for every line of code and all of our code works exactly as designed
- For lack of a better term, it's “perfect”
- Right?

Bzzt.

Wrong Bozo.

It doesn't mean that at all.

Code Coverage

- 100% code coverage means that when the entire sum of all your tests are run, every line is executed at least once
- And it means nothing else*

** Except in TDD*

```
class Calc {

public function add($a, $b){

  return $a+$b;
}

}
```

```
class TestCalc extends
  PHPUnit_Testcase
{

public function testAdd() {
$calc = new Calc();

assertEquals(5, $calc->add(2, 3));
assertEquals(0, $calc->add(1,-1));
}

}
```

```
class Calc {

public function add($a, $b){

  $_SESSION['count']++;
  $_SESSION['doh'] = true;

  return $a+$b;
}

}
```

```
class TestCalc extends
  PHPUnit_Testcase
{

public function testAdd() {
$calc = new Calc();

assertEquals(5, $calc->add(2, 3));
assertEquals(0, $calc->add(1,-1));
}

}
```

So what's the point?

- Code Coverage in itself is a useless metric
- It does not mean your code is done, perfect, bug-free, good, bad, or anything else..
- No matter what your PHB (or the Rails community) says
- We need to move past code coverage..

What do we replace it with?

- Useful Tests
  - A test is “useful” if it tests a new set of conditions not previously covered
  - A test that reproduces a previous bug or error state and demonstrates its resolution is ++good
  - It must test something non-trivial

```
class Calc {

public function add($a, $b){

  return $a+$b;
}

}
```

```
class TestCalc extends
  PHPUnit_Testcase
{

public function testAdd() {
$calc = new Calc();

assertEquals(5, $calc->add(2, 3));
assertEquals(0, $calc->add(1,-1));
}

}
```

```
class Calc {

public function add($a, $b){

  return $a+$b;
}

}
```

```
class TestCalc extends
  PHPUnit_Testcase
{

public function testAdd() {
$calc = new Calc();

assertEquals(5, $calc->add(2, 3));
assertEquals(0, $calc->add(1,-1));
}

}
```

How in the world is this non-trivial!?

What should we avoid?

- Trivial Tests
  - Trivial Tests contribute to code rot, bad inertia, and generally make you and your team test-resistant
  - If you're writing these tests, ***just stop it*** or tigers will eat you
  - *Seriously, I'll arrange it*

How should we represent this?

- Useful Tests / Total Tests (higher is better?)
- Trivial Tests / Total Tests (lower is better?)
- Useful Tests / Trivial Tests (higher is better, but divide by zero is ideal?)
- *Don't ask me*

So which tests do we need?

- Code is Communication
  - Every line you type communicates Instructions to the computer, intentions to your team, and explanations to the you of six months from now
- Unit Tests must be treated the same

When we write tests..

- We have to balance many requirements
  - The customer wants results
  - The boss just wants it done
  - We want confidence in current and future changes

So we have to choose carefully

- First, look towards ***new*** functionality
  - A scary codebase is no reason to let your problem grow
  - Start testing the new pieces now, figure it out and even if you never get to the backlog, things are getting better
  - We ***communicate*** a new expectation

In web2project..

- Helper & Formatting functions
  - These were completely new to the system and designed to generate specifically defined blocks of html & data
  - Building tests for these let us prototype, expand, and use them throughout the system with confidence

Testing ***new*** functionality

- This offers some interesting options
  - You can make sure new code adheres to new QA processes like coding standards, reviews, etc
  - As you figure out how to test the new pieces, something else will emerge..

So we have to choose carefully

- Next, look towards ***core*** functionality
  - If you have classes or functions used constantly and all over the place, add tests for those next
  - We ***communicate*** stability and intention
  - The most important reason is eliminating the pseudo-bogus bug reports

In web2project..

- Core security, filtering, and formatting
  - As we built our Helpers, we found that most were using filtered input data & combining the results of core functions
  - Testing core functions supported both the new code (Helpers) and numerous places throughout the system

Testing ***core*** functionality

- Gives us some interesting perspective
  - New modules & functionality benefits immediately, and even more as refactoring occurs
  - We get “free” testing all over the system with a relatively small amount of effort and a new pattern will emerge..

So we have to choose carefully

- Finally, test ***problematic*** functionality
  - Your tests in other areas will highlight the "annoying" parts of the system
  - "Annoying" can be where the most bugs are or it could be what is changing the most, it doesn't matter

In web2project..

- Trevor Morse – *Canadian but still an okay guy*
  - By sorting our issue reports by module, two modules stood out as having nearly 50% of the bugs, the next closest ~5%
  - With the confidence from the other core functions, adding complex tests became possible

But we have another opportunity

- "To report a bug, we need a test"
  - While bug reports without tests aren't ignored, they are considered after the testable ones
  - The Zend Framework has this "recommendation"
  - The team ***communicates*** problems

Broken Window Theory

- James Q Wilson & George L Kelling
  - You set the standard for your team, group, project, neighborhood
  - As that standard lower, everyone's expectations drop & behavior changes
  - As that standard raises, expectations improve & behavior change
  - http://en.wikipedia.org/wiki/Broken_windows_theory
  - http://pragprog.com/the-pragmatic-programmer/extracts/software-entropy

Recap

We defined Unit Testing

We criticized the Holy Grail of 100% Code Coverage

We talked about the difference between Useful and Trivial Tests

We covered that code – whether project or tests – is communication

We talked about implementing tests on a project first for new functionality, then core, then pain points.

I threatened you with tigers

Questions?

D. Keith Casey Jr, Chief Stuff Breaker

keith@blueparabola.com

Twitter/Skype/AIM/IRC: caseysoftware