

Methods to Improve Quality

Pieter van den Hombergh

13 september 2005

- 1 Unit testing
- 2 Why should I bother with Unit testing
 - Collateral damage
- 3 Sorry excuses
 - It ...
 - But I already got...
- 4 First Unit tests
 - simple assertions

Motivation for Unit Testing

Or what excuses are there to not do unit testing.

Does my code do what I want?

- Or rather, does the code (under test) fulfil my intent?
- Does it do what I want all of the time?
 - Every program has more than one path of execution, some of them leading to the success scenario, others to failure
Programmers who say they test, very often only test with the **right** input. (By the way: *[h|cr]ackers tend to test with **wrong** input*)
- So test with non existing file, invalid input, (simulated) disk full situations etc.

What do I want to accomplish?

- Do not get carried away
 - Once you get the hang of it, you might think its fun doing unit test. Do not get carried away. E.g. do not test setters and getters if setting a value does not change an object's state.

Test your exceptions

Typical code fragment by novices

```
catch(Exception e) {  
    // silenty ignore  
}
```

Having code like this drops information on the floor. Either really handle the exception or log the information somewhere.

What is collateral damage?

Collateral damage is what happens when a new feature bug fix in one part of the system causes a bug (damage) to another, possibly unrelated part of the system. It's an insidious problem that, if allowed to continue, can quickly render the entire system broken beyond anyone's ability to fix.

What is collateral damage?

We sometime call this the “Whac-a-Mole” effect.



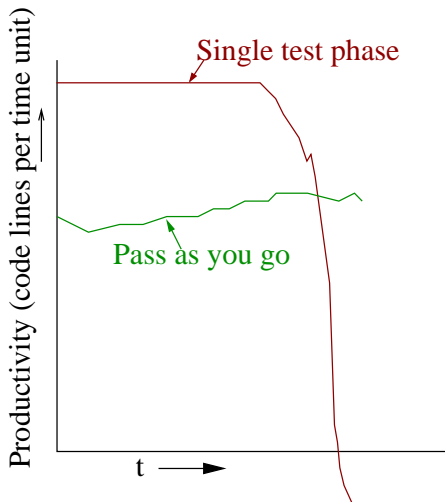
In the carnival game of Whac-a-Mole, the player must strike the mechanical mole heads that pop up on the playing field. But they don't keep their heads up for long; as soon as you move to strike one mole, it retreats and another mole pops up on the opposite side of the field. The moles pop up and down fast enough that it can be very frustrating to try to connect with one and score. As a result, players generally flail helplessly at the field as the moles continue to pop up where you least expect them. Widespread collateral damage to a code base can have a similar effect. From Pragmatic unit testing

Can I depend on my code?

Code you cannot depend on is useless. Worse, If you do not know it is (un)reliable, you might looking for bugs at the wrong places. Testing helps you to assess validity of the code. So if it blows up, you know what caused it and how to signal it.

How do I do unit Testing?

- 1 Write the unit test from the specifications of the code.
- 2 Run it.
- 3 It might not compile or fail because it cannot find the class under test.
- 4 The write the class to test.
- 5 Then cycle through
 - 1 Run the tests
 - 2 Correct the failure
 - 3 Until all test pass.



- It takes to much time to write the tests
 - *It takes more time to find bugs, even if you intend not to write any. Unit test lets the bugs find you. Unit testing is a method to keep your productivity constant and reliable*

It takes too much to run the tests

It should n't. Most unit test run quickly. For tests that take a long time to run, run them less often.

It is not my job to test my code

Then what is your job? Hand out buggy code?

I don't really know how my code is supposed to behave, so I cannot test it

Maybe it is not the right time writing your code yet. A (testable) prototype might help clarify the exact requirements.
If you do not know what the code is supposed to do, then how do you know it does?

But it compiles!

Your compiler might catch this

```
public static void main(Strings args[]) {  
    // more code left out  
}
```

but how about this?

```
1  public void addit( Object anObject ) {  
2      ArrayList myList= new ArrayList();  
3      myList.add( anObject );  
4      myList.add( anObject );  
5      // more code ...  
6  }
```

How should the compiler know?

I'm paid to write code, not to write tests.

By the same logic you are not paid to spend all day in the debugger.

Where that 'but...' could lead to

A consultant recently came to our team asking for help with his project. He pulled out a print-out of his Java source file to reveal the god class to end all god classes, weighing in at a whopping 9700 lines long. I flipped to a random page (one out of 87 printed pages) and saw two squelched exceptions i.e.

```
catch (SQLException ex) {  
}
```

in an if statement nested 6 levels deep. A quick shuffle through the rest of the stack revealed more of the same. I suggested to the others on my team that finding something as bad as this was a special occasion, and we should celebrate. by ChadFowler.

Test are quite simple

You can use several rather simple methods, provide by the framework. They typically look much like this:

```
public void assertTrue(boolean condition) {  
    if (!condition) {  
        abort();  
    }  
}
```

As you can see they do nothing if the test-condition is **true**.

Typical use in a unit test would be:

```
int a = 2;  
// some intermediate code  
//  
assertTrue( a == 2 );  
// more tests?
```

Or asserts like this

```
public void assertsEquals( int a , int b ) {  
    assertTrue( a == b );  
}
```

These kind of assertions exist for the standard java types and then some. You could also create your own: (Even money: whole dollars, euros, not cents)

```
class Money {
    public void assertEvenMoney( String Message,
                                Money amount ) {
        assertEquals( message, amount.asDouble(),
                    Math.floor(amount.asDouble()),
                    0.001);
    }
    public void assertEventMoney( Money amount ) {
        assertEvenMoney( "", amount );
    }
}
```

Ensuring that (the proper) Exceptions are thrown can be tested like this

```
public void testForException( SomeClass sc ) {  
    try {  
        ct.( sc );  
        fail( "exception not thrown" );  
    }  
    catch ( ExceptionThatWasExpected etwe ) {  
        assertTrue( true ); // code = documentation  
    }  
    catch ( RuntimeException re ) {  
        fail( "wrong exception thrown"  
            + re.getMessage() );  
    }  
}
```