Test first is fun!

Monday, September 08, 2008

Posted by Philip Zembrod

So the Test-Driven-Development and Extreme-Programming people tell you you should write your tests even before you write the actual code. "Now this is taking things a bit too far," you might think. "To the extreme, even. Why would I want to do this?"

In this post, I'll tell you my answer to this question. I now really do want to write my tests first...and here's why!

After many years of writing code without using or writing unit tests, I took a colleague's advice and read Kent Beck's "Extreme Programming Explained." I picked "write tests first" as the first XP practice to try out in my daily coding.

The practice is: **Write a failing test for each feature** you plan to implement. Run the test and see it fail. **Then implement the feature until the test succeeds.** Refactor now and begin again.

Why write the test first? The obvious reason, I thought, was to make it more likely that tests will get written at all. But I heard the promise that this was not just a way to ensure tests aren't overlooked, but **a way to higher productivity**. I tried it, and found that **getting tests written was indeed one of the less important reasons to write tests first!**

Writing tests firsts **leads you to think about the interface first**. Of course, you do that anyway when you write the header file with the C++ class definition or when you write a Java interface before you implement any methods. However, writing a test lets you **focus on how the new interface will be *used*** before even writing the interface. You could call writing the interface the supply side and writing the test the demand side of the deal. Writing the test first, you set out with the customer's or user's view of the new class.

Another way of seeing the same thing is to **regard the test as a coded specification**. In the test, you specify what service the new class or feature should provide, and you specify, by example, the syntax with which this service will be requested. In contrast to specifications written in natural language, a specification written into a test contains **a technical safeguard against growing stale: if it does, the test will probably fail**.

These two aspects of unit tests are enough to make me feel excited about writing them first. **Tests are no longer a necessary chore, but the place and time where I start to design something new.** That's what I love to do. How soon can I get started writing my next test?

But this is still not the best part: If I write a test first, run it to see it fail (often even fail to compile), and write the code to satisfy the test, then I have everything in place to ***see my code running the minute it is written and compiled!*** No more dread of strange behaviour or system crashes the first time I launch the system with my new code! No more laborious navigating through the application to my code's feature! No more wondering: Did my code actually get executed or not?

Just a quick run-the-testcase, and I know how my code runs: green - good. Red - not yet good. Read failure message and fix code until green. Debugging sucks, testing rocks, indeed!

Of course, all the complex issues of integration and system testing remain. Good unit testing gives me a good head start for integration, but I might still be in for unpleasant surprises there.

The point I want to make here, though, is about my state of mind when I write new code. For me, **writing new code for complex systems was always accompanied by fear**: fear of crashes I'd have to debug, fear of creating bugs I might not discover, fear of the dreary work of searching for bugs I might have created. Fear that took up a considerable amount of my mind space and **slowed me down**.

Now, this fear is gone! I happily go about writing my code because I know the tests are already in place. **It will cost me just a few keystrokes to run my finished code, and I will immediately see what it does.** Hooray, I wrote a program, and it works, and it's easy to prove it!

It's the same old enthusiasm that I felt more than 20 years ago when I wrote and ran my first programs. Many of you have felt it, too - the joy of inducing some life into this dead piece of hardware through our written word. And now this joy sits in my mind again where

fear of crashes was before. You'd better believe that speeds up my coding! Want to give it a try yourself?

Labels: Philip Zembrod