Unit Tests Do Cost More To Write...

One common objection I hear to TDD and writing unit tests is that it creates a lot of extra work to develop software. And this is true, but it does not give a complete picture. TDD and writing and maintaining unit tests do take discipline and do increase the time (thus cost) of writing code. No doubt about it. However, this cost reflects the true cost of writing quality code and lowers the total cost of developing and maintaining software.

Unfortunately on many projects, management looks only at coding milestones to determine how quickly software is progressing. By milestone 1, these ten features must be implemented. What inevitably happens, assuming a tight deadline, is that developers cram and jam to get these features implemented and checked in by the milestone. They reach this milestone, and management checks these items off their project plan and record the time to completion and consider this to be the cost of developing these features.

This is completely wrong and gives a false sense of cost. Inevitably, the more time pressure a developer is to finish a feature, the more likely he or she will introduce bugs. Bugs that may not be caught much later and are much more costly to find and fix then.

Robert Glass points out in his book Facts and Fallacies of Software Engineering that Error removal is the most time-consuming phase of the life cycle. It can account for anywhere from 30 to 40 percent of the time spent coding. However this time is often booked to the task debugging and not booked back to the original feature, giving a true sense of the cost and time involved to implement the feature.

Glass also points out that 80% of the cost to implement a project is spent in the maintenance phase. I don't know about you, but I've had the unenviable task to maintain a large creaky system whereby I would make a change to implement a new feature or fix a bug, and would hope and pray I didn't break something else in some subtle manner. I would spend way more time trying to test and verify my change than it took to implement the change.

This is where I believe that unit tests ultimately provide their cost savings. They both reduce the true cost of developing a feature and reduce the **Total Cost of Ownership** (or TCO) of a codebase.

They reduce the true cost of software development by promoting cleaner code with less bugs. They reduce the TCO by documenting how code works and by serving as regression tests, giving maintainers more confidence to make changes to the system.

Remember, as in all writing, **consider your audience**. The audience for your code is not the compiler, it is the poor maintenance programmer six months from now who has to go in and make a change to the system. Sometimes that poor soul is yourself. Believe me, that person will thank you profusely if the system has sufficient unit tests.

And before I'm misquoted, I want to remind the reader that my predilection for unit tests does not mean that it is a substitute for other forms of testing.

Author: Phil Haack
Copied from: http://haacked.com/archive/2005/12/06/11302.aspx