



Surprise! Software Rots!



Is software designed to be simple and elegant more valuable than software that is complex and hard to maintain? An Agile process accepts this as an important fact.

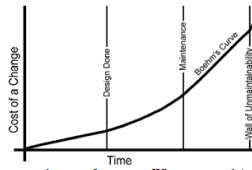
It may surprise you to learn software rots. Intellectually we know it really doesn't, but it might as well. Software rot is caused by improper design and limited project resources. Complexity creeps in as easy code changes are made instead of difficult design changes. Code duplication accumulates rapidly during maintenance tasks.

After a while we notice that fixing one bug causes several even more subtle (and expensive) bugs to occur and the cost of maintenance goes up significantly. Eventually the cost of maintenance exceeds resources. It seems as if our code has decayed on its own in spite of our best efforts.

Barry Boehm found that as software proceeded through its life cycle the cost of making a change became larger. Ratios of 100:1 for making a change after delivery versus project start are common. But ratios as low as 5:1 can be found. How flat this curve stays depends on many individual project factors.

This cost curve is commonly interpreted to mean that you must create infallible requirements documents followed by complete, detailed, and error-free designs or pay a huge price later. That isn't what it means. Boehm's findings are not a condemnation of change but rather a caution to be prepared when changes occur. A well run project keeps the cost of changes lower longer.

There are three life cycle events that seem



to accelerate software rot. When we proclaim the design is done and accept no more changes. When we move the system into maintenance and change the team's process. Last when the cost of making vital changes exceeds our resources we reach the wall of unmaintainability.

To stay Agile you must fight software rot. **Refactoring** is the art of making design changes over time to keep the software fit for its purpose and ready for more changes. **Unit** and **acceptance** tests can almost eliminate the fear that drives inappropriate easy changes on a delivered system. **Early delivery** of partial systems helps detect big changes before they become expensive. An Agile process accepts that requirements, analysis, and design are never truly done. An Agile process runs equally well in maintenance or development easing the transition.

We know that some changes will cost much more than others. Boehm's findings were that 20% of the changes make up 80% of the effort. We accept that. We counter it with honest estimates and use a planning process that requires the customer to guide our spending decisions and

cost compromises. We fight software rot from start to finish so it can never dominate our decisions.

One common misconception about testing software is the cost. Most developers only know two data points; no tests, and too many tests. Finding bugs cost you something. If you don't test your software you are essentially moving that cost to your users/customers. If you only pay for fixing bugs but not finding them you save money.

On the other hand if you set out determined to test every possible input to your software you will find the cost of testing rises exponentially as you get close to 100% tested. Of these two data points not testing is obviously cheaper.


We know now that you can not effectively increase software quality at the end of the project by testing and fixing bugs. It remains low quality but with fewer bugs. Second, designing code to be testable lowers the cost of testing it. There is a sweet spot between no tests and too many tests that minimizes the total cost of development. It lies some where close to testing everything that could actually break.

Agile processes are a re-evaluation of the way software is created. The quality of the source code is much more important than you may think. Just because customers don't see code doesn't mean we are excused from the effort needed to be ready for changes by keeping quality up, complexity down, and full test coverage.

Next

[Agile-Process.org home](#) | [Most Important Features First](#) | [XP guided tour](#) | [About the Author](#)

Copyright 2009 Don Wells all rights reserved.



Don't face retirement alone.

[LEARN MORE](#)