A common debate in software development projects is between spending time on improving the quality of the software versus concentrating on releasing more valuable features. Usually the pressure to deliver functionality dominates the discussion, leading many developers to complain that they don't have time to work on architecture and code quality.

Betteridge's Law of headlines is an adage that says any article with a headline or title that ends in a question mark can be summarized by "no". Those that know me would not doubt my desire to subvert such a law. But this article goes further than that - it subverts the question itself. The question assumes the common trade-off between quality and cost. With this article I'll explain that this trade-off does not apply to software - that high quality software is actually cheaper to produce.

Although most of my writing is aimed at professional software developers, for this article I'm not going to assume any knowledge of the mechanics of software development. My hope is that this is an article that can be valuable to anyone involved with thinking about software efforts, particularly those, such as business leaders, that act as customers of software development teams.

We are used to a trade-off between quality and cost

As I mentioned in the opening, we are all used to a trade-off between quality and cost. When I replace my smart phone, I can choose a more expensive model with faster processor, better screen, and more memory. Or I can give up some of those qualities to pay less money. It's not an absolute rule, sometimes we can get bargains where a high quality good is cheaper. More often we have different values to quality - some people don't really notice how one screen is nicer than another. But the assumption is true most of the time, higher quality usually costs more.

Software quality means many things

If I'm going to talk about quality for software, I need to explain what that is. Here lies the first complication - there are many things that can count as quality for software. I can consider the user-interface: does it easily lead me through the tasks I need to do, making me more efficient and removing frustrations? I can consider its reliability: does it contain defects that cause errors and frustration? Another aspect is its architecture: is the source code divided into clear modules, so that programmers can easily find and understand which bit of the code they need to work on this week?

These three examples of quality are not an exhaustive list, but they are enough to illustrate an important point. If I'm a customer, or user, of the software, I don't appreciate some of the things we'd refer to as quality. A user can tell if the user-interface is good. An executive can tell if the software is making her staff more efficient at their work. Users and customers will notice defects, particularly should they corrupt data or render the system inoperative for a while. But customers and users cannot perceive the architecture of the software.

I thus divide software quality attributes into external (such as the UI and defects) and internal (architecture). The distinction is that users and customers can see what makes a software product have high external quality, but cannot tell the difference between higher or lower internal quality.

At first glance, internal quality does not matter to customers

Since internal quality isn't something that customers or users can see - does it matter? Let's imagine Rebecca and I write an application to track and predict flight delays. Both our applications do the same essential function, both have equally elegant user interfaces, and both have hardly any defects. The only difference is that her internal source code is neatly organized, while mine is a tangled mess. There is one other difference: I sell mine for $6 and she sells hers for $10.