What is Code Quality?

Of course, you want quality code, who wouldn't? But to improve code quality, we have to define what it is.

A quick Google search yields many results defining code quality. As it turns out, the term can mean many different things to people.

One way of trying to define code quality is to look at one end of the spectrum: high-quality code. Hopefully, you can agree on the following high-quality code identifiers:

- It does what it is supposed to do.
- It does not contain defects or problems.
- It is easy to read, maintain, and extend.

These three identifiers, while simplistic, seem to be generally agreed upon. In an effort to expand these ideas further, let's delve into why each one matters in the realm of software.

Why Does Code Quality Matter?

To determine why high-quality code is important, let's revisit those identifiers. We'll see what happens when code doesn't meet them.

It does **not** do what it is supposed to do

Meeting requirements is the basis of any product, software or otherwise. We make software to do something. If in the end, it doesn't do it… well it's definitely not high quality. If it doesn't meet basic requirements, it's hard to even call it low quality.

It **does** contain defects and problems

If something you're using has issues or causes you problems, you probably wouldn't call it high-quality. In fact, if it's bad enough, you may stop using it altogether.

For the sake of not using software as an example, let's say your vacuum works great on regular carpet. It cleans up all the dust and cat hair. One fateful night the cat knocks over a plant, spilling dirt everywhere. When you try to use the vacuum to clean the pile of dirt, it breaks, spewing the dirt everywhere.

While the vacuum worked under some circumstances, it didn't efficiently handle the occasional extra load. Thus, you wouldn't call it a high-quality vacuum cleaner.

That is a problem we want to avoid in our code. If things break on edge cases and defects cause unwanted behavior, we don't have a high-quality product.

It is **difficult** to read, maintain, or extend

Imagine this: a customer requests a new feature. The person who wrote the original code is gone. The person who has replaced them now has to make sense of the code that's already there. That person is you.

If the code is easy to comprehend, you'll be able to analyze the problem and come up with a solution much quicker. If the code is complex and convoluted, you'll probably take longer and possibly make some wrong assumptions.

It's also nice if it's easy to add the new feature without disrupting previous features. If the code is *not* easy to extend, your new feature could break other things.

No one *wants* to be in a position where they have to read, maintain, or extend low-quality code. It means more headaches and more work for everyone.

It's bad enough that you have to deal with low-quality code, but don't put someone else in the same situation. You can improve the quality of code that you write.

If you work with a team of developers, you can start putting into place methods to ensure better overall code quality. Assuming that you have their support, of course. You may have to win some people over (feel free to send them this article