David Chappell

THE THREE ASPECTS OF SOFTWARE QUALITY: FUNCTIONAL, STRUCTURAL, AND PROCESS

Sponsored by Microsoft Corporation

Our world runs on software. Every business depends on it, every mobile phone uses it, and even every new car relies on code. Without software, modern civilization would fall apart. Given this reality, the quality of that software really matters. Because it's so widely used and so important, low-quality software just isn't acceptable.

But what exactly is software quality? It's not an easy question to answer, since the concept means different things to different people. One useful way to think about the topic is to divide software quality into three aspects: functional quality, structural quality, and process quality. Doing this helps us see the big picture, and it also helps clarify the trade-offs that need to be made among competing goals.

Before we do this, however, it's worth taking a moment to think about what software quality isn't. It's tempting to view software quality through the same lens as other kinds of quality, such as quality in a manufacturing process. Doing this is misleading, however. In manufacturing, a primary goal is to minimize defects in products created through a repeatable process. Methodologies such as Six Sigma were created to help do this, and they've been quite effective. Yet every software development project requires some innovation—if this isn't true, you should be buying rather than building the software—and so the project isn't executing an exactly repeatable process. Because of this, views of quality rooted in manufacturing aren't the best approach to thinking about software quality. A broader perspective is required.

Who Cares About Software Quality?

With software or anything else, assessing quality means measuring value. Something of higher quality has more value than something that's of lower quality. Yet measuring value requires answering another question: value to whom? In thinking about software quality, it's useful to focus on three groups of people who care about its value, as Figure 1 shows.

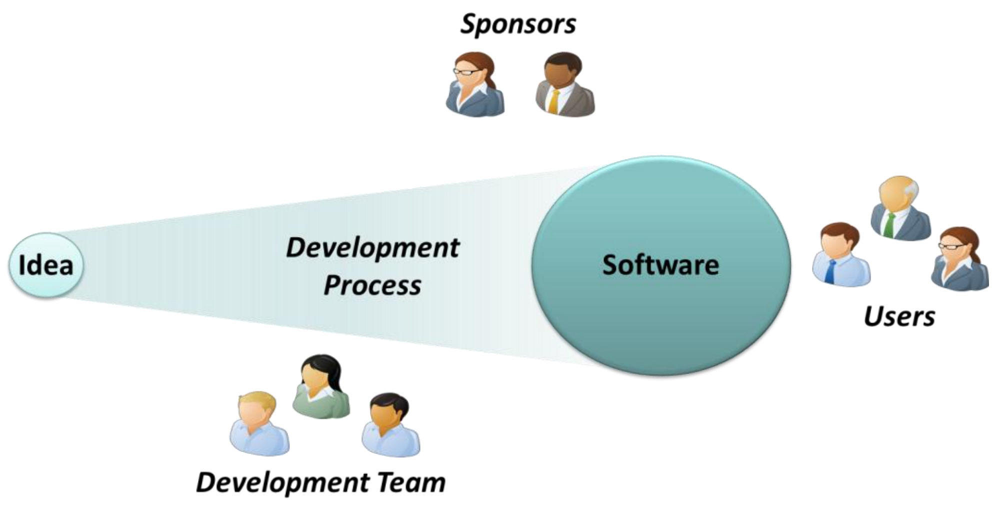

Figure 1: As a development process transforms an idea into working software, three main groups of people care about the software's quality.

As the figure illustrates, a development process converts an idea into usable software. The three groups of people who care about the software's quality during and after this process are:

- The software's *users*, who apply this software to some problem.
- The *development team* that creates the software.
- The *sponsors* of the project, who are the people paying for the software's creation. For software developed by an organization for its own use, for example, these sponsors are commonly business people within that organization.

All three of these groups care about software quality. The aspects of quality that each finds most important aren't the same, however. Understanding these differences requires taking a closer look at what software quality really means.

Defining Software Quality: Three Aspects

There's no one right way to think about software quality—it's a complicated area. It is useful, however, to group its various components into three broad aspects. Figure 2 illustrates this idea.

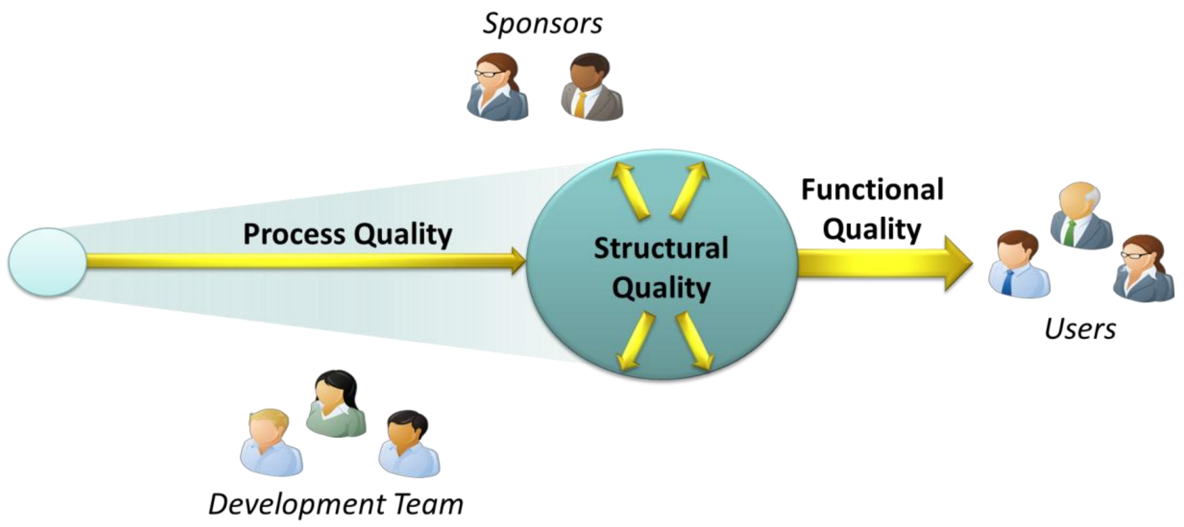

Figure 2: Software quality can be seen as having three aspects: functional quality, structural quality, and process quality.

The three aspects of software quality are *functional* quality, *structural* quality, and *process* quality. Each one is worth looking at in more detail.

Functional quality means that the software correctly performs the tasks it's intended to do for its users. Among the attributes of functional quality are:

- *Meeting the specified requirements*. Whether they come from the project's sponsors or the software's intended users, meeting requirements is the *sine qua non* of functional quality. In some cases, this might even include compliance with applicable laws and regulations. And since requirements commonly change throughout the development process, achieving this goal requires the development team to understand and implement the correct requirements throughout, not just those initially defined for the project.

- *Creating software that has few defects*. Among these are bugs that reduce the software's reliability, compromise its security, or limit its functionality. Achieving zero defects is too much to ask for most projects, but users are rarely happy with software they perceive as buggy.
- *Good enough performance*. From a user's point of view, there's no such thing as a good, slow application.
- *Ease of learning and ease of use*. To its users, the software's user interface *is* the application, and so these attributes of functional quality are most commonly provided by an effective interface and a well-thought-out user workflow. The aesthetics of the interface—how beautiful it is—can also be important, especially in consumer applications.

Software testing commonly focuses on functional quality. All of the characteristics just listed can be tested, at least to some degree, and so a large part of ensuring functional quality boils down to testing.

The second aspect of software quality, structural quality, means that the code itself is well structured. Unlike functional quality, structural quality is hard to test for (although there are tools to help measure it, as described later). The attributes of this type of quality include:

- *Code testability*. Is the code organized in a way that makes testing easy?
- *Code maintainability*. How easy is it to add new code or change existing code without introducing bugs?
- *Code understandability*. Is the code readable? Is it more complex than it needs to be? These have a large impact on how quickly new developers can begin working with an existing code base.
- *Code efficiency*. Especially in resource-constrained situations, writing efficient code can be critically important.
- *Code security*. Does the software allow common attacks such as buffer overruns and SQL injection? Is it insecure in other ways?

Both functional quality and structural quality are important, and they usually get the lion's share of attention in discussions of software quality. Yet the third aspect, process quality, is also critically important. The quality of the development process significantly affects the value received by users, development teams, and sponsors, and so all three groups have a stake in improving this aspect of software quality.

The most obvious attributes of process quality include these:

- *Meeting delivery dates*. Was the software delivered on time?
- *Meeting budgets*. Was the software delivered for the expected amount of money?
- *A repeatable development process that reliably delivers quality software*. If a process has the first two attributes—software delivered on time and on budget—but so stresses the development team that its best members quit, it isn't a quality process. True process quality means being consistent from one project to the next.

There are many connections among these three aspects of software quality. For example, improving process quality with agile development methods increases the odds of getting the project's requirements right, which also improves functional quality. There are trade-offs as well, where improving quality in one area can lower quality in another. An organization might speed up a project's development process to meet a deadline—improving process quality—only to find that the number of bugs in the software has gone up, hurting functional quality. Similarly, cutting features can lower functional quality, since users get less of what they're looking for, but improve process quality by increasing the odds of meeting a release date. In general, each development project weighs the interests of all three groups—and all three aspects of quality—against one another. Different projects make different trade-offs.

There are many connections among these three aspects of software quality.

Unsurprisingly, everybody involved in a software project cares most about the aspects of quality that directly impact them. Users care primarily about functional quality, since that's what they see. They're also likely to care about some aspects of process quality, such as the delivery date of the final software. Users typically don't care at all about structural quality, even though its absence might well impact them over the software's lifetime.

A development team certainly does care about structural quality, however, since they're the people who will be affected by the problems caused by low quality here. They also care about functional quality, although perhaps a bit less than users do—cutting features that users want can make life easier for developers. Development teams also care about process quality, in part because it provides many of the metrics by which they're measured.

The third group, sponsors, cares about everything: functional quality, structural quality, and process quality. If they're smart, the people paying for the project know that slacking off in any area is a poor long-term strategy. In the end, sponsors are striving to create business value, and the best way to do this is by taking a broad view of software quality. They must also understand the connection between quality and risk. The risk of accepting lower software quality in, say, a community website, is much less than the risk of allowing lower quality in an airplane's flight control system. Making the choice appropriately commonly requires trade-offs among competing goals.

Tools for Improving Software Quality

Viewing software quality as having three distinct aspects is useful. It implies, however, that tools for improving software quality need to address all three parts. Functional quality is important—testing certainly matters—but tools focused on structural and process quality are needed, too.

Tools for improving functional quality include manual testing tools that let a tester explore the software through its user interface, along with tools for automated testing, such as frameworks for unit testing. Tools for load testing and performance testing can also help measure and improve those components of functional quality.

Tools that help improve the structural quality of software provide services such as refactoring, which lets a developer improve how code is organized without changing what that code does. Structural quality tools can also provide static code analysis, examining code for security problems (such as the potential for SQL injection attacks) and other problems, along with dynamic code analysis, which might include performance profiling, measures of test coverage, and more. These tools can also provide various code metrics, such as measurements of cyclomatic complexity.

Tools for improving process quality help monitor and manage the development process. They include support for tracking the status of the process, perhaps by mapping requirements against progress measurements for the developer responsible for each one. Process quality tools can also provide insight into code churn, i.e., the number of lines added or modified each week, progress in finding and fixing bugs, test plan progress, and other measures of project health.

As with every other aspect of software development, using good tools certainly helps.

Whatever they do, it’s important to realize that unlike tools for functional quality and structural quality, which are typically used solely by the development team, tools for process quality are also used by the project’s sponsors (and maybe even by the software’s users). This means that these tools should be accessible through less technically focused interfaces, such as spreadsheets and collaboration software. Making them available only through developer tools isn’t enough.

Tools aren’t the whole story, of course. Activities such as group code reviews and effective management can also have a big impact on various aspects of software quality—people matter. Yet as with every other aspect of software development, using good tools certainly helps.

Conclusion

There’s no single best way to view software quality—different perspectives emphasize different things. The simple three-part breakdown described here is meant to provide one useful way to think about this topic.

By providing an equal emphasis on functional quality, structural quality, and process quality, this approach helps broaden our view to include the things that matter to all three stakeholders: users, development teams, and sponsors. Including all three areas also helps us think about the kinds of tools we need to improve software quality.

Given how important software has become to the world, it’s hard to overemphasize the importance of software quality. Recognizing the broad scope this idea encompasses is a useful step along this path.

About the Author

David Chappell is Principal of Chappell & Associates (www.davidchappell.com) in San Francisco, California. Through his speaking, writing, and consulting, he helps people around the world understand, use, and make better decisions about new technologies.