SOFTWARE QUALITY METRICS EXPLAINED

"If you don't collect any metrics, you're flying blind. If you collect and focus on too many, they may be obstructing your field of view." Scott M. Graffius

Fundamentally understood as the measurements used to ensure an end result is delivered with quality, metrics are important to daily life because they transform requirements and performance for the better. But no matter your industry, what defines "quality" is entirely subjective.

That's particularly true when it comes to software. Does quality rely on the app working, on clean code, or least amount of bugs? In this article, we'll look at metrics for measuring your software quality. We will discuss:

- Why you want to use quality metrics
- The aspects of measuring quality
- Some common metrics
- How to choose metrics that suit your needs

What are software quality metrics?

The exact steps to quality control and what qualifies as a good measurement tool or aspect varies from person to person, business to business. What is important to one may not be important to another. And, as the quote from project management expert Scott M. Graffius touches on, applying too many metrics not only takes up time, it can confuse the entire result.

When we think of software quality metrics, the same concept applies. As companies are increasingly moving toward an agile and extreme release cycle of software, you need to ensure all your products and code are of quality. Therein lies the problem.

Deploying and pushing software to the market faster means often "flying blind," potentially allowing quality to suffer due to lack of time. As metrics are intuitive and depend entirely on who you are talking to:

- One software company may focus on number of bugs found
- Another focuses on failure rates.
- Like Facebook, your metric focus might simply be how many users—humans—return to use the product again

Overall the purpose is not to allow your software to suffer. To do that, you must find a software quality metric system that works.

The software industry is sometimes lacking in the ability to measure and control what is being done. For many organizations, finding a good formula to align software quality metrics is confusing because there are so many metrics! However, most software quality metric experts do agree on a few things, which we'll look at shortly.

In this video, David Rizzo, VP Product Engineering, and David Kennedy, Solutions Architect, share how Compuware evolved to DevOps to increase innovation, reduce escaped defects, and improve MTTR. The KPIs they measure prove that Agile and DevOps truly provide a business advantage:

Why software quality metrics matter

If we look at the definition of metrics in relation to value, it is ideal to then focus on end-user requirements and what value they are getting from the software. Helping your team to achieve ultimate value with higher quality, you can use metrics to evaluate, modify, and improve the process over time. Creating a cycle of value from start to finish in relation to quality will ultimately:

- Improve revenue
- Cut costs and time
- Promote agile software development environments

In short, it solves the problem of coders only looking at the software while in development and not addressing further measurement of the finished work or how the user sees it.

Software requirements

Defined by experts as a "description of features and functionalities of the target system," software requirements can be broken down into either Functional or Nonfunctional, often used to describe how users expect the software product to perform.

- **Functional requirements.** The simple inputs, behaviors, and outputs of a software system. These requirements are used to understand the intended behavior—calculation, data manipulation, user interaction, and more. If these functions do not happen, then the software is not working.
- **Nonfunctional requirements.** The representation of standards that are measured to ensure

the effectiveness of a software system (aka quality). Nonfunctional requirements like rate of recovery, privacy, usability, and more outline how a system should operate while covering most aspects used to define software quality.

(*Learn more about Software Requirement Specifications.*)

Managing quality

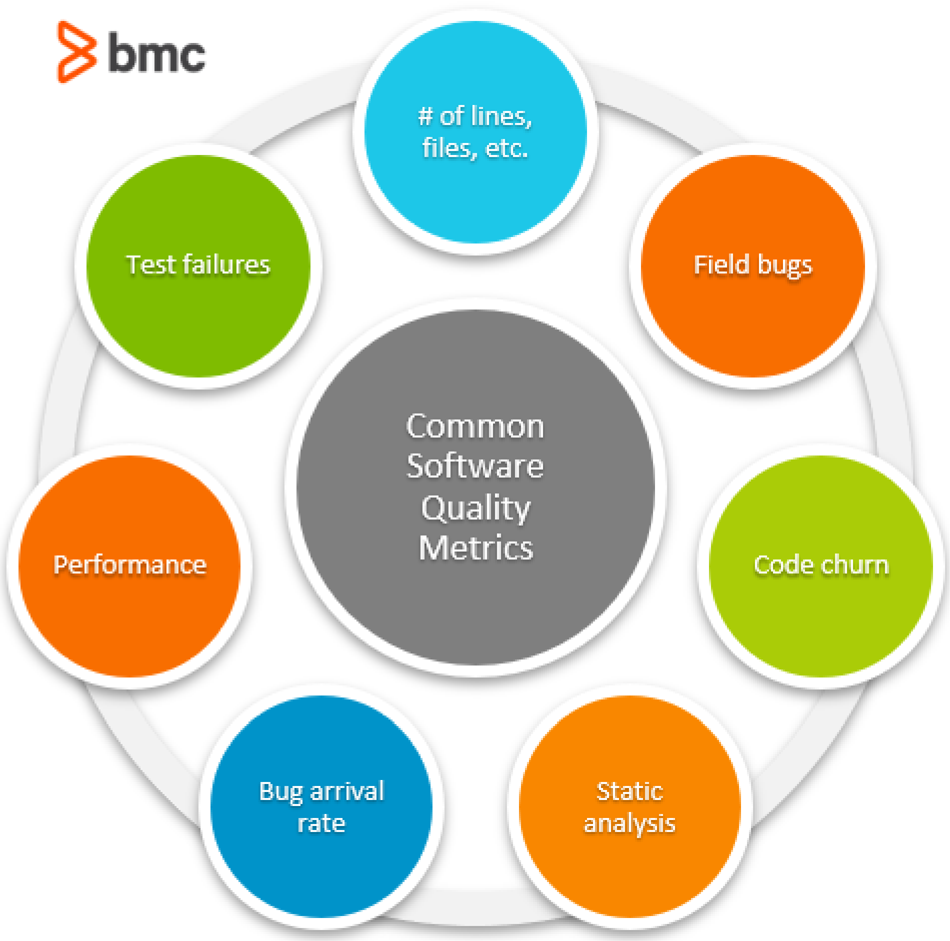
When it comes to managing software quality, there are many factors that come into play. With aspects that often can not be measured with numbers or graphs, this is the part of "quality" that is subjective. To begin, you will want to identify the aspects of software quality. To do this, you need to address the 5 following questions:

1. **What is my quality?** This looks at the maintainability of software. Is it easy or difficult to work with?
2. **What is my risk?** Reliability of the software. How likely is it to fail and do the tests work?
3. **Where is my risk?** Security issues and vulnerabilities of the software. Can you identify spots that could result in breaches?
4. **Is it good enough?** Performance and usability quality of the software. What does it look like across its entire lifecycle?
5. **Is it done?** Rate of bug delivery and testability of the software. Can you ensure all results meet the criteria of "doneness"?

Focus on one aspect of the software at a time. This will help in "not obstructing your field of view".

7 metrics to ensure software quality

The following principles or factors are things that can be measured. Then use the results to test the quality of your software as it applies to the above quality aspects trying to be achieved.



- **Number of lines, files, etc.** How do your file sizes affect your software? What is the function of the code lines? Are your numbers **maintainable**?
- **Field bugs.** What are the problems in the already running software? How many bugs did you find in production? Is your software **reliable**? How many times did it fail over a set period of time?
- **Code churn.** Why is one part of the code churning more than others? Why is it error-prone? Is anything in the completion rate standing out? Is the code **usable** once it has churned?
- **Static analysis findings.** What is consistent about the software? How long does it take to fix code? Is the software **secure** in its current standing vs. what has changed?
- **Bug arrival rate.** How are you finding bugs? When did a bug show up? Why did you think the software was ready? What are the **rates** at which bugs are coming in? How many software releases happened during a period?
- **Performance.** Does the software code last during updates? Is it performing as it should? Why isn't it **performing** in load, stress, or response testing? Do users enjoy it?
- **Test failures.** Automated and manually, what tests are failing? Was a test working and now it is failing? What is your failure balance? How can **testability** be improved with technologies?

Choosing your metrics

The only way to maximize the chances of releasing high-quality software and of creating a highly agile software development environment is to adapt some formula of the above aspects and metrics combined. Comprehensively testing and managing quality will ultimately lead to the greater overall value of the software to its users.

Many companies over time have chosen to focus on just a few to determine quality. For example, look at Facebook's practice of weighing greatly on performance and user return. No matter what is tested and determined the goal remains the same—to deliver quality software that is worth releasing and valuable to the user.

Related reading

- BMC DevOps Blog
- DevOps Guide, a series of 30+ articles
- Choosing IT Metrics That Matter
- DevOps Metrics & KPIs
- Testing Automation Explained: Why & How To Automate Testing