Software Quality Metrics Explained With Examples

Patrick Ward Follow
Oct 01, 2022 · 9 mins read

What are Software Quality Metrics?

Software quality metrics are a group of performance indicators used to measure the quality of a software product. Examples include Agile metrics like velocity and QA metrics like test coverage.

If you are managing the development of new software, and you want to know how to organize everything effectively and evaluate the quality of the product before release, software quality metrics are vital tools that can help you understand and improve many different aspects of a project.

Metrics themselves do not improve development, but managers use them to get a better picture of the production process. Presenting all elements of the process with data, metrics describe what is happening throughout the project. Without this information, managers struggle to discover issues and make improvements.

However, with the valuable information software quality metrics provide, you can more effectively:

- Predict defects
- Discover and resolve bugs
- Efficiently organize development
- Increase productivity

In this article, I will go over a variety of common metrics and provide examples of how you can use them to interpret, quantify, and, most importantly, improve the quality of software and the productivity of a team.

Why Are Software Quality Metrics Used?

Software Quality Metrics (SQMs) are essential management tools; they are quantities that **express information about a piece of software or the process of development**.

Certain metrics **measure the quality of code** by analyzing its complexity or ability to pass tests. Other process-related metrics describe **the workflow and productivity of developers** by measuring the time they spend planning, programming, and repairing the software.

Using software quality metrics is standard in software development companies because metrics tell the company **how the development process is running and how it can be better**. A project manager that uses metrics can analyze the development in all stages and use the information to **find inefficiencies and improve them**. In practice, the use of metrics has been found to reduce problems in each phase of development.

Tools to Evaluate Code

Metrics can also predict how easily code will be maintained and updated. Developers can use this information to revise and restructure code to reduce complexity and improve its quality.

Tools to Evaluate Developers

Software metrics provide accurate information that quantifies the efficiency of development phases and the productivity of developers. When effectively used by project managers, metrics can help improve the organization of a project, so the right developers use their time efficiently on the tasks most suited to their skillsets.

Common Software Quality Metrics

I will discuss three categories of common metrics: agile metrics, quality assurance (QA) metrics, and production metrics. Individually, these metrics focus on specific elements of productivity and code quality; combined, they help improve the effectiveness of a team and the reliability of a product.

Agile Metrics

The following metrics are used in the popular agile method of development that focuses on quickly producing working software in short iterative cycles of planning, development, and reflection. To help us reflect on the development process and improve it, agile metrics assess the time it takes for developers to complete tasks and the velocity at which they work. Three commonly used agile metrics are:

- Lead Time
- Cycle Time
- Velocity

Lead Time and Cycle Time

Lead time is the **time period starting when the task or project is requested and ending with the completion of the assignment.** Cycle time is very similar to lead time, except it doesn't include the time the task waits before being addressed by developers; it only includes **the time from when the task is started and finished**.

By measuring the time spent on individual tasks from request to development to completion, lead time and cycle time help you understand how tasks are processed, queued, distributed, and finished.

Example

After a project's release, if you measure lead and cycle time as users report issues and the developers resolve them, you can see how long tasks are delayed before resolution. long delays in the task queue can be shortened by focusing management efforts on reducing this wait so the development team can address problems more quickly.

Velocity

Another important metric is velocity, and it measures **how quickly individual developers can finish a given task**. Understanding a developer's velocity for certain tasks tells you which developers will work more efficiently in certain areas of development and produce a better product.

Example

If velocity metrics reveal that development among certain individuals lags behind others, you can try revising your task-distribution process so developers work in areas better suited to their skillsets. Later, if velocity increases, you know your efforts succeeded.

QA Software Quality Metrics

QA (Quality Assurance) metrics primarily **describe the quality of code and its potential to result in defects or difficult maintenance**. Project managers use QA metrics to help **predict, prevent, and resolve problems**.

Test Coverage

This metric measures **how thoroughly testing probes a project**. Having thorough test coverage means using a strong suite of unit tests, as well as user tests, to evaluate the functionality of every imaginable facet of a given program and see under which conditions the software fails and where fixes are needed. The metric helps a team understand **how they can better test a program and discover defects before users do**.

Example

If you regularly calculate the test coverage during the development stage and find it low, you should task developers with making stronger tests. If test coverage remains high throughout development, you know the team is thoroughly testing the software and exposing more

defects.

Code Coverage

Code coverage is similar to test coverage, except that it **measures the amount of the codebase that is tested by unit tests**. Unit tests should cover every line of code in a product, as well as every branch of a condition and every path through the program, to ensure it all works well in production.

Example

Many tech giants rely on code coverage metrics to find bugs and ensure their code is strong and reliable. Google, for example, uses code coverage to assess code quality and employs techniques to continue improving code coverage for new developments.

Code Complexity Metrics

When trying to understand how difficult code will be to comprehend, test, repair, and maintain, code complexity metrics can **express the complexity of a piece of code**.

Cyclomatic Complexity

Managers normally calculate this metric known as cyclomatic complexity with an automatic software tool that constructs a graph with nodes and edges from the code and uses an equation to find the maximum number of independent paths in a program. The more independent paths, the higher the metric and the more complex the code.

Example

If during development you notice the productivity (expressed in the agile metrics above) decreasing and the code complexity increasing, it means the code is too complex. Generally, any cyclomatic complexity value above 10 suggests that the code will be difficult to maintain. So, if the complexity approaches or exceeds this value, you should focus on simplifying the program to allow changes to be made more easily.

Mean Time to Detect and Mean Time to Repair

Mean Time to Detect (MTTD) refers to **the average time it takes for the software company to detect an issue**. This metric helps give a number quantity to the ability of a company to discover problems. You want this metric low because it means you are discovering problems sooner. If this metric is high, it means you could improve your defect discovery methods.

Mean Time to Repair (MTTR) is similar to MTTD except that it **quantifies a team's ability to repair defects instead of discovering them**.

Example

After product release, you should monitor these metrics and ensure they remain low; this means developers are repairing and improving the software quickly. If these numbers are high, you should devote more of the development team's effort to finding and repairing issues.

Production Metrics

Production information reveals how developers use their time, how efficiently they work, and how often they need to revise their code, and it can help you evaluate a developer's skill and assign them to suitable tasks to increase efficiency.

To know specific information about how developers produce software, you can use these metrics:

Active Days

This metric simply measures the number of days developers spend **actively coding**. It does not include time spent planning. In this way, it is very similar to lead time and helps us understand the ratio between time used for coding and time spent on other activities.

Example

The active days metric can be used to analyze and organize how time is allocated during a project. If a developer spends few active days on a task and allocates time to other activities that unnecessarily slow progress, you can address this issue directly and help the developer better manage their time.

Productivity

Typically based on the volume of code produced by a developer, productivity in software development should be measured if you want to see how and when developers work best. Also, productivity measurements help when deciding how many developers are needed to work on a project.

Example

If at the beginning of a project you want to know how to organize development teams and distribute tasks to developers, you can look at past productivity metrics to determine how many engineers are needed in which areas and which developers are best suited for the various tasks.

Code Churn

Code Churn refers to the amount of **code that is changed in a piece of software**. Although modifications are a natural aspect of the development and maintenance of a program, excessive changes can result in unpredicted defects. Code churn also describes the amount of code modified due to defects; thus, higher code churn often equates to faultier code.

Example

If code churn is low after publication, then you can assume that the code was well produced, and you can learn from the development practices used. However, if code churn is high after publication, the codebase may have a defect-prone structure and need changes to its design.

Conclusion

Metrics can greatly help in the management of software development by providing information about the quality of the code, the productivity of the team, and the areas that need improvement. Without data, a project manager may find it difficult to interpret the effectiveness of a development team and discover issues.

However, with the help of metrics, you can continuously find ways to improve the product and facilitate efficient development.

When making a soup, a chef must taste the dish regularly throughout the entire process to judge whether it is cooking properly, well-seasoned, or, perhaps, missing an ingredient. Similarly, you use metrics to judge development and know what works well and what needs improvement.

Written by Patrick Ward Follow

Hi, I'm Patrick. I made this site to share my expertise on team augmentation, nearshore development, and remote work.

Featured Articles

01 How to Demonstrate Willingness to Learn [Resume, Interview]

02 How to Get an MBA Without a Bachelors Degree in 2022

03 How to Keep Employees Happy Without a Raise

04 How to Outsource in the Philippines (For Tech Companies)

05 BPO Meaning: Business Process Outsourcing Explained

Business Glossary
@NanoGlobals