

A Software Metrics Primer¹

Karl E. Wieggers

Process Impact

www.processimpact.com

My friend Nicole is a quality manager at Motorola, a company widely known for its software process improvement and measurement success. Once I heard her say, “Our latest code inspection only found two major defects, but we expected to find five, so we’re trying to figure out what’s going on.” Few organizations have enough insight into their software development and project management efforts to make such precise statements. Nicole’s organization spent years making its processes repeatable and defined, measuring critical aspects of its work, and using those measurements to set up well-managed projects that develop high-quality products.

Software measurement is a challenging but essential component of a healthy and highly capable software engineering culture. In this article, I describe some basic software measurement principles and suggest some metrics that can help you understand and improve the way your organization operates. Plan your measurement activities carefully because they can take significant effort to implement and the payoff will come over time.

Why Measure Software

Software projects are notorious for running over schedule and budget, yet still having quality problems. Software measurement lets you quantify your schedule, work effort, product size, project status, and quality performance. If you don’t measure your current performance and use the data to improve your future work estimates, those estimates will just be guesses. Because today’s current data becomes tomorrow’s historical data, it’s never too late to start recording key information about your project.

You can’t track project status meaningfully unless you know the actual effort and time spent on each task compared to your plans. You can’t sensibly decide whether your product is stable enough to ship unless you’re tracking the rates at which your team is finding and fixing defects. You can’t quantify how well your new development processes are working without some measure of your current performance and a baseline to compare against. Metrics help you better control your software projects and learn more about the way your organization works.

What to Measure

You can measure many aspects of your software products, projects, and processes. The trick is to select a small and balanced set of metrics that will help your organization track progress toward its goals. Goal-Question-Metric (GQM) is an excellent technique for selecting appropriate metrics to meet your needs.. With GQM, you begin by selecting a few project or organizational goals. State the goals to be as quantitative and measurable as you can. They might include things like:

¹ This paper was originally published in *Software Development*, July 1999. It is reprinted (with modifications) with permission from *Software Development* magazine.

- Reduce maintenance costs by 50% within one year
- Improve schedule estimation accuracy to within 10% of actual
- Reduce system testing time by three weeks on the next project
- Reduce the time to close a defect by 40% within three months.

For each goal, think of questions you would have to answer to see if you are reaching that goal. If your goal was “reduce maintenance costs by 50% within one year,” these might be some appropriate questions:

- How much do we spend on maintenance each month?
- What fraction of our maintenance costs do we spend on each application we support?
- How much money do we spend on adaptive (adapting to a changed environment), perfective (adding enhancements), and corrective (fixing defects) maintenance?

Finally, identify metrics that will let you answer each question. Some of these will be simple data items you can count directly, such as the total budget spent on maintenance. Other metrics you will compute from two or more data items. To answer the last question listed previously, you must know the hours spent on each of the three maintenance activity types and the total maintenance cost over a period of time.

Notice I expressed several goals in terms of a percentage change from the current level. The first step of a metrics program is to establish a current baseline, so you can track progress against it and toward your goals. I prefer relative improvement goals (“reduce maintenance by 50%”) rather than absolute goals (“reduce maintenance to 10% of total effort”). You can probably reduce maintenance to 10% of total effort within a year if you are currently at 20%, but not if you spend 80% of your effort on maintenance today.

Your balanced metrics set should eventually include items relating to product size, product quality, process quality, work effort, project status, and customer satisfaction. Table 1 suggests metrics that individual developers, project teams, and development organizations should consider collecting. You would track most of these over time. For example, your routine project tracking activities should monitor the percentage of requirements implemented and tested, the number of open and closed defects, and so on. You can’t start with all of these, but I recommend including at least the following measurements early in your metrics program:

- *Product size*: count lines of code, function points, object classes, number of requirements, or GUI elements
- *Estimated and actual duration (calendar time) and effort (labor hours)*: track for individual tasks, project milestones, and overall product development
- *Work effort distribution*: record the time spent in development activities (project management, requirements specification, design, coding, testing) and maintenance activities (adaptive, perfective, corrective)
- *Defects*: count the number found by testing and by customers and their type, severity, and status (open or closed)

Creating a Measurement Culture

Fear is often a software practitioner’s first reaction to a new metrics program. People are afraid the data will be used against them, that it will take too much time to collect and analyze the data, or that the team will fixate on getting the numbers right rather than building good software. Creating a software measurement culture and overcoming such resistance will take diligent, consistent steering by managers who are committed to metrics and sensitive to these concerns.

Table 1. Appropriate Metrics for Software Developers, Teams, and Organizations.

Group	Appropriate Metrics
Individual Developers	<ul style="list-style-type: none"> • Work effort distribution • Estimated vs. actual task duration and effort • Code covered by unit testing • Number of defects found by unit testing • Code and design complexity
Project Teams	<ul style="list-style-type: none"> • Product size • Work effort distribution • Requirements status (number approved, implemented, and verified) • Percentage of test cases passed • Estimated vs, actual duration between major milestones • Estimated vs. actual staffing levels • Number of defects found by integration and system testing • Number of defects found by inspections • Defect status • Requirements stability • Number of tasks planned and completed
Development Organization	<ul style="list-style-type: none"> • Released defect levels • Product development cycle time • Schedule and effort estimating accuracy • Reuse effectiveness • Planned and actual cost

To help your team overcome the fear, you must educate them about the metrics program. Tell them why measurement is important and how you intend to use the data. Make it clear that you will never use metrics data either to punish or reward individuals (and then make sure that you don't). A competent software manager does not need individual metrics to distinguish the effective team contributors from the slackers.

Respect the privacy of the data. It is harder to abuse the data if managers don't know who the data came from, Classify each data item you collect into one of these three privacy levels:

- Individual: only the individual who collected the data about his or her own work knows it is his or her data, although it may be pooled with data from other individuals to provide an overall project profile
- Project team: data is private to the members of the project team, although it may be pooled with data from other projects to provide an overall organizational profile
- Organization: data can be shared among all members of the organization.

As an example, if you are collecting work effort distribution data, the number of hours each individual spends working on every development or maintenance phase activity in a week is private to that individual. The total distribution of hours from all team members is private to the project team, and the distribution across all projects is public to everyone in the organization.

View and present the data items that are private to individuals only in the aggregate or as averages over the group.

Make It a Habit

Software measurement doesn't need to be time consuming. Commercial tools are available for measuring code size and complexity in many programming languages. Activities like daily time tracking are more of a habit than a burden. Commercial problem tracking tools facilitate counting defects and tracking their status, but this requires the discipline to report all identified defects and to manage them with the tool. Develop simple tracking forms, scripts, and web-based reporting tools to reduce the overhead of collecting and reporting the data. Use spreadsheets and charts to track and report on the accumulated data at regular intervals.

Tips for Metrics Success

Despite the challenges, many software organizations routinely measure aspects of their work. If you wish to join this club, keep the following tips in mind.

Start Small. Because developing your measurement culture and infrastructure will take time, use GQM to first select a basic set of initial metrics. Once your team becomes used to the idea of measurement and you have established momentum, you can introduce new metrics that will give you the additional information you need to manage your projects and organization effectively.

A risk with any metrics activity is dysfunctional measurement, in which participants alter their behavior to optimize something that is being measured, rather than focusing on the real organizational goal. For example, if you are measuring productivity but not quality, expect some developers to change their coding style to expand the volume of material they produce, or to code quickly without regard for bugs. I can write code very fast if it doesn't actually have to run correctly. The balanced set of measurements helps prevent dysfunctional behavior by monitoring the group's performance in several complementary aspects of their work that lead to project success. Never attempt to use metrics to motivate performance.

Explain Why. Be prepared to explain to a skeptical team why you wish to measure the items you choose. They have the right to understand your motivations and why you think the data will be valuable. Use the data that is collected, rather than letting it rot in the dark recesses of a write-only database.

Share the Data. Your team will be more motivated to participate in measurement activities if you inform them about how you've used the data. Share summaries and trends with the team at regular intervals and get them to help you understand what the data is telling you. Let them know whenever you've been able to use their data to answer a question, make a prediction, or assist your management efforts.

Define Data Items and Procedures. It is more difficult and time-consuming to precisely define the data items and metrics than you might think. However, if you don't pin these definitions down, participants may interpret and apply them in different ways. Define what you mean by a "line of code," spell out which activities go into the various work effort categories, and agree on what a "defect" is. Write clear, succinct procedures for collecting and reporting the measures you select.

Understand Trends. Trends in the data over time are more significant than individual data points. Some trends may be subject to multiple interpretations. Did the number of defects per function point found during system testing decrease because the latest round of testing was ineffective, or did fewer defects slip past development into the testing process? Make sure you understand what the data is telling you, but don't rationalize your observations away.

Measurement alone will not improve your software performance, but it provides information that will help you focus and evaluate your software process improvement efforts. Link your measurement program to your organizational goals and process improvement program. Use the process improvement initiative to choose improvements, use the metrics program to track progress toward your improvement goals, and use your recognition program to motivate and reward desired behaviors.

Further Reading

My book, *Creating a Software Engineering Culture* (Dorset House, 1996) includes an overview of software metrics and a case study on software work effort measurement. Robert Grady's *Practical Software Metrics for Project Management and Process Improvement* (PTR Prentice Hall, 1992) describes a world-class metrics program developed at Hewlett-Packard. *Applied Software Measurement, Second Edition*, by Capers Jones (McGraw-Hill, 1996) is a comprehensive treatise on metrics that includes industry averages for productivity and quality.