Chapter 12

A Software Metrics Primer[1]

In this chapter:

My friend Nicole is a quality manager at a large company recognized 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 engineering process to make such statements. Nicole's organization spent years establishing effective processes, measuring critical aspects of its work, and using those measurements on 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. This chapter describes some basic software measurement principles and suggests some metrics that can help you understand and improve the way your organization and its projects operate (Jones 1996; Wiegers 1996). Project initiation is a good time to choose the appropriate measures that will help you assess project performance and product quality. Plan your measurement activities carefully because they can take significant initial effort to implement and the payoff will come over time.

Why Measure Software

Software projects are notorious for running over schedule and budget and having quality problems to boot. Software measurement lets you quantify your schedule and budget performance, work effort, product size, product quality, and project status. 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 begin recording key information about your project.

1 This chapter was originally published in *Software Development,* 1999, 7(7): 39–42. It is reprinted here, with modifications, with permission of CMP Media Inc.

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 assess 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 better understand how 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 effective technique for selecting appropriate metrics to meet your needs (Daskalantonakis 1992; Basili and Rombach 1988). 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 targets such as the following:

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

For each goal, think of ***questions*** you would have to answer to judge whether your team is reaching that goal. If your goal was "reduce maintenance costs by 50 percent 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 do we currently spend on adaptive maintenance (adapting to a changed environment), perfective maintenance (adding enhancements), and corrective maintenance (fixing defects)?

Finally, identify ***metrics*** that will provide answers to each question. Some of these will be simple items you can count directly, such as the total budget spent on maintenance. These are called *base measures*. Other metrics are *derived measures*, which are calculated from one or more base measures, typically as simple sums or ratios. To answer the final question in the previous list, you must know the hours spent on each of the three maintenance activity types and the total maintenance cost over a period of time.

Note that 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 to establish relative improvement goals ("reduce maintenance by 50 percent") rather than absolute goals ("reduce maintenance to 20 percent of total effort"). You can probably reduce maintenance to 20 percent of total organizational

effort within a year if you are currently at 30 percent, but not if you spend 80 percent of your effort on maintenance today.

Your balanced metrics set should eventually include items relating to the six basic dimensions of software measurement: size, time, effort, cost, quality, and status. Table 12-1 suggests some metrics that individual developers, project teams, and development organizations should consider collecting. Some of these metrics relate to products, others to projects, and the remainder to processes. You should track most of these metrics 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 at once, but I recommend including at least the following measurements early in your metrics program:

1. *Product size:* Count lines of code, function points, object classes, requirements, user stories, or GUI elements. Adjust raw counts for complexity.
2. *Estimated and actual duration:* Count in units of calendar time.

Table 12-1 Appropriate Metrics for Software Developers, Teams, and Organizations

Level	Appropriate Metrics
Individual Developer	1. Work effort distribution 2. Estimated and actual task duration and effort 3. Code covered by unit testing 4. Number and type of defects found by unit testing 5. Number and type of defects found by peer reviews
Project Team	1. Product size 2. Estimated and actual duration between major milestones 3. Estimated and actual staffing levels 4. Number of tasks planned and completed 5. Work effort distribution 6. Requirements status 7. Requirements volatility 8. Number of defects found by integration and system testing 9. Number of defects found by peer reviews 10. Defect status distribution 11. Percentage of tests passed
Development Organization	1. Overall project cycle time 2. Planned and actual schedule performance 3. Planned and actual budget performance 4. Schedule and effort estimating accuracy 5. Released defect levels 6. Reuse effectiveness

3. *Estimated and actual effort:* Count in units of labor hours.
4. *Work effort distribution:* Record the time spent in various development and maintenance activities (Wiegers 1996).
5. *Defects:* Track the number, type, severity, and status of defects found by testing, peer reviews, and customers.
6. *Requirements status:* Track the number of requirements that are proposed, approved, implemented, verified, deleted, and rejected.

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 on building good software. Creating a software measurement culture and overcoming such resistance will take diligent, congruent steering by managers who are committed to measurement and sensitive to these concerns.

To help your team overcome the fear, 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 you don't. A competent software manager does not need metrics data from individuals to distinguish the effective team contributors from those who are struggling.

Respect the privacy of the data (Grady 1992). It's harder to abuse the data if managers don't know who the data came from. Classify each base measure you collect into one of these three privacy levels:

- **Individual** Only the individual who collected the data about his own work knows it is his 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 may be shared throughout the organization.

As an example, if you're collecting work effort distribution data, the number of hours each individual spends working on every development or maintenance 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 need not consume a great deal of time. Commercial tools are available for measuring code size in many programming languages. Activities such as daily time tracking constitute a habit each developer gets into, not 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, consider using GQM to first select a basic set of initial metrics. Once your team becomes used to the idea of measurement and you've gained some momentum, you can introduce new metrics that will provide 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 instead of focusing on the real organizational goals (Austin 1996). As an illustration, if you're measuring productivity but not quality, expect some developers to change their programming style to visually expand the volume of code they produce, or to code quickly without regard for defects. I can write code very fast if it doesn't have to run correctly. The balanced set of measurements helps prevent dysfunctional behavior by monitoring the team's performance in several complementary dimensions that collectively lead to project success.

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. Encourage team members to use the data themselves and to identify other measures that will help them improve their work. When my small software group at Kodak established a work effort metrics program in 1990 we found it to be more helpful than we expected. One team member expressed it well: "It was interesting to compare how I *really* spent my time with how I *thought* I spent my time and how I was *supposed* to spend my time."

Share the Data Your team will be more motivated to participate in the measurement program if you tell them 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 saying. Let team

members 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's more difficult and time consuming to precisely define the base measures and derived measures than you might think. However, if you don't pin these definitions down, participants are likely to 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 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. Measurement 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 reward desired behaviors.

Practice Activities

1. Select several appropriate metrics for your organization at the individual level, the project level, and the organizational level. You could use Worksheet 12-1 to identify candidate metrics in the six key dimensions of software measurement.
2. Alternatively, you could use Worksheet 12-2 to try the goal-question-metric strategy to select metrics that will align with your project or organizational goals. Use a separate copy of Worksheet 12-2 for each goal.

Worksheet 12–1: Your Project Metrics

Category	Possible Metrics
Size	
Effort	
Time	
Cost	
Quality	
Status	

Worksheet 12–2: Goal-Question-Metric

Goal:		
Questions:		
Metrics:		