Software Development

A Practical Approach to Software Metrics

PUBLISHED IN IT Professional, VOLUME 2, NUMBER 1, JANUARY/FEBRUARY 2000.

Peter Kulik

Metrics programs that create meaningful change in software practice must start with business goals in mind.

What are the benefits of using metrics? How do I go about starting a new metrics program or improving current practices and what should I do with the data once itš been collected? These are some of the questions that IT managers face when they consider software metrics. There are many good reasons to keep metrics for software projects, yet many programs cause managers to lose their focus or get caught up in ffeasurement for

Software metrics are quantitative standards of measurement for various aspects of software projects. A well-designed metrics program will support decision making by management and enhance return on the IT investment. There are many aspects of software projects that can be measured, but not all aspects are worth measuring.

Starting a new metrics program or improving a current program consists of five steps:

- identify business goals,
- select metrics,
- gather historical data,
- automate measurement procedures, and
- use metrics in decision-making.

IDENTIFY BUSINESS GOALS

Of the five steps, identifying business goals is the most important. Well-defined business goals will

- enable a metrics program to enhance business results,
- reduce cost by keeping a program well-defined and focused, and
- ensure a basis for improving a business'return on investment for IT.

It can be difficult to gain support for metrics, particularly when nontechnical managers are part of the decision-making. For example, if your CEO asks why he should invest in something that isn't producing more software, it may not help to tell him that most software organizations are doing it and it seems like a good idea. Rather, you should be prepared to say something like "Software metrics will help us reduce the number of faults reported in newly-deployed software by 25 percent without increasing project schedules. The resulting savings in support costs should drive a 150 percent return on investment in the first year." Having a rationale along these lines is much more likely to convince a business manager to invest in software metrics.

This rationale is where the business goals come in. A focus on business goals will help "sell" software metrics to the chief information officer, vice president of engineering, or CEO. The first step is to identify relevant business goals. Some companies already publish annual business objectives. Some CIOs or engineering vice presidents will translate these into divisional goals. If you don't know your company's business goals and related IT goals, ask senior managers. If you find it hard to identify business goals that can be addressed by a metrics program, propose some. The sidebar, "Common IT Business Goals," offers suggestions.

Common IT Business Goals

- Improve the quality of installed software.
- Make achievable project commitments.
- Reduce postinstallation support costs.
- Prevent schedule and cost overruns.
- Do more with the same number of people.
- More quickly estimate software projects.
- Deliver new products to customers sooner.

SELECT METRICS

Once you've identified business goals, the next step is to select metrics that support them. The following commonly used metrics are a good place to start, although other metrics are certainly possible to measure and may be more appropriate on the basis of your organization's business goals.

Schedule metrics

This data can help prevent schedule and cost overruns by providing early warning of schedule problems. Schedule metrics can include measurements such as the number of tasks completed on schedule, the number of tasks completed late, and the number of tasks rescheduled.

Requirements metrics

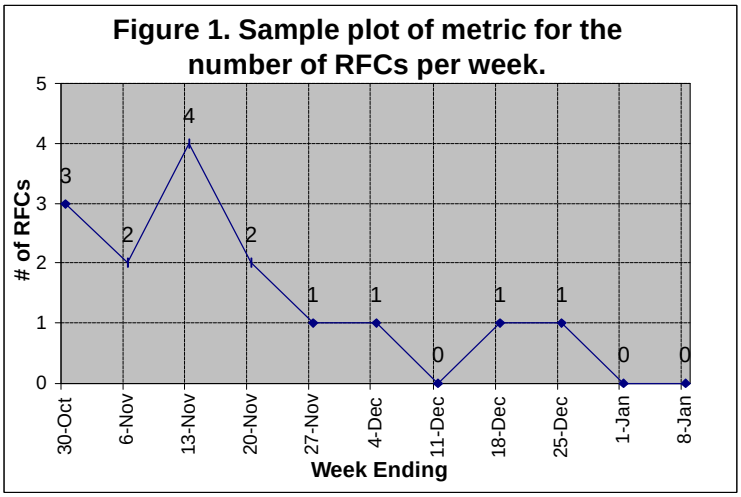
This data can help serve as a leading indicator for schedule and cost overruns. It includes measurements of the number or percentage of changed requirements and the number or percentage of new requirements. It requires a formal request-for-change procedure to capture appropriate data. See the example in Figure 1.

Test coverage percentage

This metric describes the fraction of lines of code "covered" by testing. Various industry studies have shown that testing without measuring code coverage covers only 50 to 60 percent of code~~le~~aving a significant amount of code that can harbor latent faults. Testing with measurement and tracking of test coverage can motivate development of additional test cases that will typically drive test coverage to 90 percent or more of the code. As a result, more of the faults in the code will be discovered by testers rather than by users. Fixing these faults prior to deployment can dramatically improve the quality of installed software and reduce support costs.

Software size

Software size is a fundamental driver of schedule duration, effort, and cost for a software development project. *Lines of code* and *function points* are the most commonly used software size metrics, although there are also several proprietary, object-oriented software size measures. The lines-of-code measure allows managers to make reasonably accurate top-down projections, and is recommended for most IT organizations. Function points allow more precise top-down projections of effort and cost, but are much more difficult to calculate and require specialized expertise.



In this project, developers prepared an early prototype and reviewed it with users over the first three weeks shown. At the end of the period shown, requirements had stabilized and the group froze them, preventing submission of further requests for comment.

Code lines should be counted excluding comment lines; this measure is called *non-comment source statements* (NCSS), or *thousands of noncomment source statements* (KNCSS). Many industry rules of thumb describe

programmer productivity in terms of lines of code, for example 350 NCSS per engineering-month of effort. Top-down estimates for a new project can be completed quickly using an estimate for the NCSS to be developed and such rules of thumb. The NCSS to be developed can be estimated based on historical data for previous projects that are similar to the new project. Measuring programmer productivity~~and~~ putting practices in place to improve it~~can~~ help an organization do more with the same number of people.

For example, the size of the project shown in Figures 2 and 3 was approximately 19 KNCSS. At an average staffing level of five, this was an approximately 10-month project.

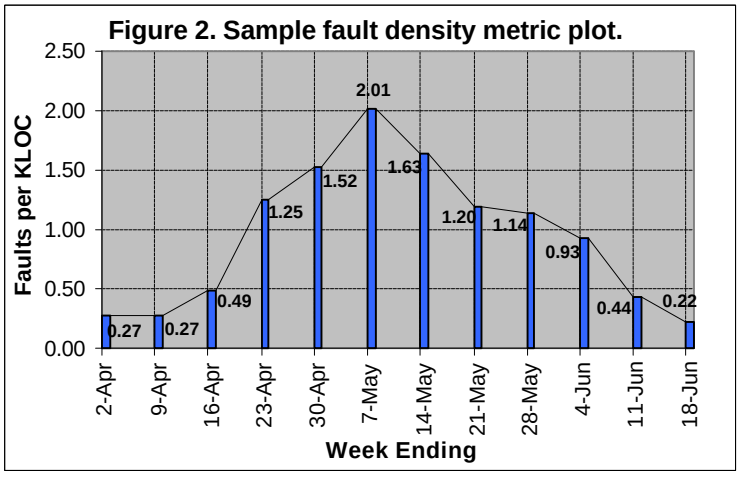
Fault density

The number of unresolved faults per KNCSS is a basic measure of software quality. Some organizations define standards for software release based on fault density~~for~~ example, no more than 0.25 faults per 1 KNCSS. Industry data shows that most software organizations discover around seven faults per 1,000 lines of code during testing. This measure can be used as a benchmark of quality, as illustrated by the example in Figure 2.

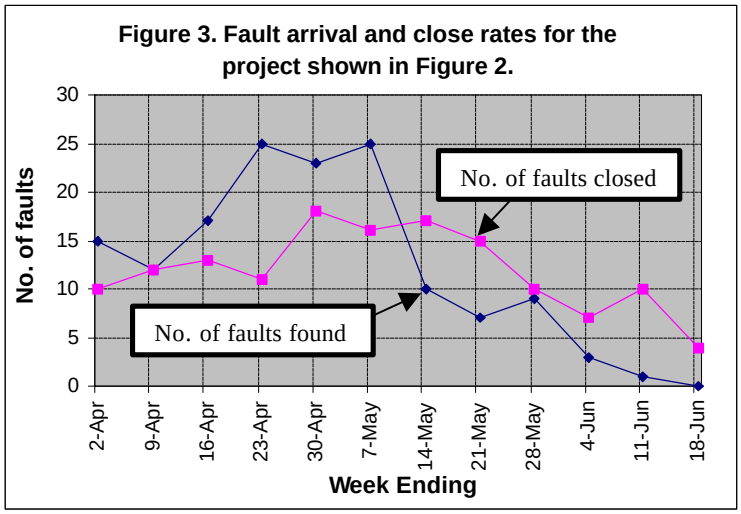
For example, one IT organization was able to reduce postrelease support costs by establishing deployment standards based on fault density. With measurably higher software quality at the start of deployment, users discovered fewer latent faults in operation, and less maintenance effort was required.

Fault arrival and close rates

A robust mechanism to determine when software is "ready to deploy," fault arrival and close rates measure the number of faults found and closed in a given period, usually per day or per week. Measuring fault arrival and close rates can help deliver software sooner by indicating lapses in testing when the fault discovery rate prematurely drops to zero and by focusing on improving the close rate.



This plot shows the fault density metric for the twelve-week system test of an approximately 19-KNCSS project. Fault density in the first two weeks was very low; however, the test team had not yet executed most of the test cases. In the twelveth week, fault density fell below 0.25 faults/KNCSS, all test cases had been executed successfully, and the team determined that the system was ready to deploy.



These fault arrival and close rates are for the same project as in Figure 2. For the last six weeks of testing, the close rate was greater than the arrival rate, and the arrival rate converged to zero while testing continued through the end of the 12th week. As in most software devlopment projects, close rate paced the duration of system test.

Close rate~~ra~~ther than fault arrival~~of~~ten paces completion of the test phase because it is generally easier to find faults than to fix them. So thatš measures.

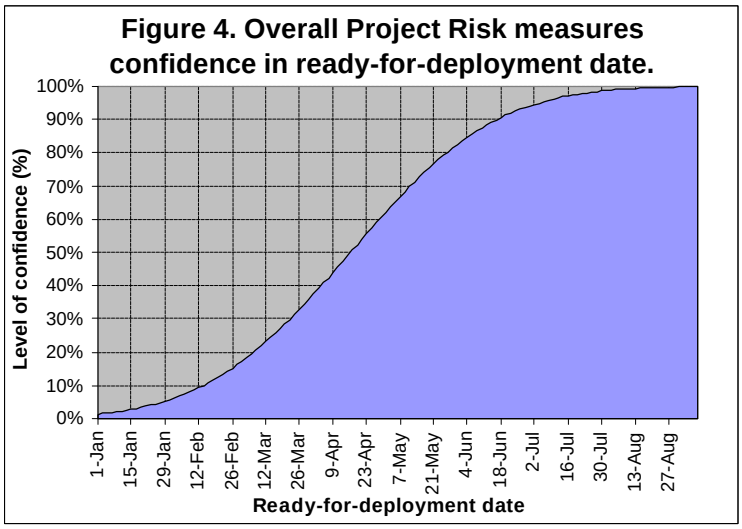
Measuring fault arrival and close rates also helps prevent over-testing. Both fault arrival and close rates will converge to zero when the software is ready to deploy; see the example in Figure 3.

For example, one IT organization found themselves in the midst of system test for a large software project with more than 200 faults open and no way to predict when the system would be ready for deployment. By measuring and tracking arrival and close rates, they were able to make a commitment for deployment readiness. Then, by taking action to improve the close rate, they were able to meet this commitment despite some surprises in testing.

Overall Project Risk metric

The likelihood of achieving a particular schedule date is expressed as a percentage, or level of confidence. Calculating this metric during project planning, as shown in Figure 4, for example, is a powerful top-down estimation method and can prevent bad commitments (such as those only 10 percent likely to be achieved). For example, one IT organization has used this method to calculate "schedule contingency" to ensure that commitments are achievable.

While a project is in progress, the Overall Project Risk metric can highlight projects most in need of management attention. For example, projects with less than a 50 percent level of confidence likely need management attention to prevent cost and schedule overruns.



Overall Project Risk describes the level of confidence for various ready for deployment dates. In this example, the project is 10 percent likely to be ready for deployment on 12 February. However, the level of confidence rises to 90 percent by 18 June.

GATHER HISTORICAL DATA

The third step in establishing or improving a metrics program is to gather historical data for the selected metrics. Many software metrics do not have intrinsic value, but can offer insights only when normalized within the context of historical data. You can glean some historical data from records of completed projects while other data requires building a database over time.

For example, lines of code is quite easy to measure for previous projects, using readily available source code measurement tools. With historical data for lines of code, predicting the number of lines of code for new projects will be much easier. Also, with data about the amount of effort that went into completing previous projects, managers can estimate historical programmer productivity and use that information as an in-house rule of thumb for schedule estimating.

Sources of historical data that are readily produced by your current processes can include

- source code,
- project schedules,
- request-for-change records,
- management reports and presentations, and
- new product support data (which products got the most customer inquiries and for what reasons).

AUTOMATE MEASUREMENT PROCEDURES

After you've selected metrics and gathered historical data, you can begin taking measurements. It can be as simple as requiring project leaders or project managers to report on chosen metrics, or as complex as hiring a new staff person to implement a measurement tool.

A management mandate is useful but not usually sufficient to support a sustainable metrics program. While you should identify a metrics champion within an organization, it is often not necessarily a full-time role. All IT staff should have tools to implement the program, including

- knowledge (through training and other communication efforts),
- paper-based templates (sheets to record metric data on, such as number

of faults found per day by severity)
- spreadsheets,
- predefined reports, and
- software tools.

In general, you should automate as much measurement as possible. This will reduce the effort required to gather metrics and improve return on investment. You can obtain basic automation, such as that afforded by simple spreadsheet tools, for a relatively small investment.

It is not unusual to encounter resistance to metrics by those being measured. Training and communication can minimize resistance as long as you include the affected software developers and testers. They should understand the business objectives these metrics address. Assure them that the metrics are not personal performance measures but, rather, measures of the software process. Metrics chosen based on business objectives should be accepted more readily by a software organization—especially if they meet the criteria in the "Metrics Checklist" sidebar.

Metrics Checklist

My company's 1999 survey of software metrics best practices (http://www.klci.com) found that effective metrics have five key attributes. To make an impact, metrics must be

- ✓ understood by management,
- ✓ easy to implement,
- ✓ understood by software developers,
- ✓ consistent measures across multiple projects, and
- ✓ real insights into project activities.

USE METRICS IN DECISION MAKING

Metrics will not drive a return on investment unless managers use them for decision making. Some decisions in which software metrics can play a role include

- product readiness to ship/deploy,
- cost and schedule for a custom project,
- how much contingency to include in cost and schedule estimates,
- where to invest for the biggest payback in process improvement, and
- when to begin user training.

Managers should demand supporting metrics data before making decisions such as these. For example, they can use fault-arrival-and-close-rate data when deciding readiness to deploy. Knowing the Overall Project Risk can help managers decide how much contingency to include in cost and schedule estimates.

The final aspect to consider is tracking the results of your metrics program. You should track results in terms of the original business objectives. For example, if the objective was a 25 percent reduction in the number of faults reported in installed modules, track this data and use it to measure the metrics program's success. Documented results help ensure that, as business goals evolve, metrics remain a relevant and vital part of software projects.

LOOKING BACK

Imagine that you are looking back on the first year of your software metrics program. You identified key business goals and implemented three metrics that supported them: code coverage; fault arrival and close rates; and Overall Project Risk.

You presented your report to management: System test was shortened by two weeks on average, service calls on new deployments are down 10 percent, and your idea about risk-based contingency reduced schedule slippage by 50 percent. The president and chief financial officer congratulate you. Your boss calls you into her office to talk about that promotion you've wanted.

While software metrics is not a "silver bullet," choosing metrics based on business goals and implementing them properly can have a measurable impact on business results. Following the five steps described in this article will help you gain new insight into software development processes, as well as find and eliminate root causes of recurring problems. The potential bottom line is enhanced return on IT investment for your company. ■

Peter Kulik *is managing partner of KLCI Inc., a software risk management and metrics firm. Contact him at pkulik@klci.com.*

Resources

Recent articles

- "Using Metrics to Justify Investment in IT," Elliot Chikofsky and Howard Rubin, *IT Professional*, Mar./Apr. 1999.
- "Tame Your Process with Metrics," Carol A. Dekkers, *Enterprise Development*, June 1999.
- "Software Metrics Best Practices," Peter J. Kulik, *Software Q/A Magazine*, Apr./May 1998.
- "Quality meets the CEO," Jeffery E. Payne, *Software Testing & Quality Engineering*, May/June 1999.
- "A Software Metrics Primer," Karl Weigers, *Software Development*, July 1999.

Classic texts

Software Metrics: Establishing a Company-Wide Program, Robert B. Grady and Deborah L. Caswell, Prentice-Hall, Inc., 1987. Detailed study of software metrics and their usefulness, used in many graduate-level courses on software engineering.

Measures for Excellence, Lawrence H. Putnam and Ware Myers, Prentice Hall, 1992. Excellent discusson of quantative metrics throughout the software life cycle and applications for estimation and management.