Measurements that Matter

by Alan Page

Several months ago, I attended a status meeting for a medium-sized software project. The product was in its fifth release and contained about five million lines of code. The project lead, Larry, was new to the company but had years of experience shipping software. Ron, the test manager, had less experience shipping software but had been with the group for the previous two releases.

Ron began by showing us a spreadsheet filled with dozens of rows containing various criteria that his team was measuring. Each cell of data was highlighted with red, yellow, or green—imitating the colors of a traffic light. Green meant that the particular measurement met or exceeded expectations. Yellow meant there was a slight deviation from the expected output, and red meant—well, red meant trouble.

The list of metrics was enormous. His team measured two different types of code coverage, seven different views of test case reporting, more than ten different performance metrics, and too many different types of bug metrics to count. The team tracked and reported data gathered by static and runtime analysis tools. It had metrics representing data from product support, customer sites, and internal users. The amount of data gathered was enormous. Ron walked through the spreadsheet, highlighting the status of each measurement. He gave a brief explanation for each green area and explained how and when each yellow and red metric would achieve green status. Everyone seemed overwhelmed by the amount of data available, but most thought that the data was a complete representation of the product quality.

At this point Larry, the project lead, asked Ron if he knew for sure whether a particular component was meeting the expected reliability needs. Ron responded, "I'm not sure. That isn't one of the things we're tracking."

"Hmm…" replied Larry. "Marketing really wants reliability data for that component. In fact, the reliability statistics for that component and the ability of the component to meet performance criteria are the two most important goals of this release. By the way," Larry continued, "do you have performance numbers on this component?"

GETTY IMAGES

"Well…" Ron replied hesitantly. "Tracking that wasn't on our radar, but we can add some metrics for that soon."

Larry interrupted again. "Ron, now that I think about it, can you give me a quick explanation of some of the metrics you are tracking? What do they tell us about the product?"

Ron, who now was a bit rattled, stammered, "Most of these are things we've always measured. Others are things that seemed interesting to measure. We need to measure as much as we can to get a good idea of how the project is doing."

"Let's talk about this some more after the meeting," said Larry. "I have some other approaches for you to consider."

There are thousands of criteria that can be used to measure software. Software teams do their best to choose a set of metrics and determine the best way to organize these measurements in order to help them understand their progress. Despite the best intentions and sincerity of those running the metrics programs, time and time again these metrics projects fail. In fact, many metrics experts claim that 80 percent of software metrics initiatives fail. Implementing a successful metrics program is difficult, but there are some things you can do to give yourself a better chance of success.

One successful method for improving the odds of success is to determine the business and customer goals of the product first and then choose only to measure criteria that support these goals. The Goal/Question/Metric (GQM) is one popular method for developing useful metrics. The GQM system consists of three steps:

1. Generate a set of goals based on the needs of the organization or business.

2. Develop a set of questions that will let us know whether we are meeting our goals.

3. Develop a set of metrics that provides answers to these questions.

There are other metrics paradigms similar to GQM, but regardless of the method, an important step for success in any metrics program is to make sure that everything that is measured relates to and supports a business or organizational goal. Measuring the wrong things will give you a false sense of progress and probably won't give you the information that your project needs. Additionally, a top-down approach, such as GQM, guarantees that management will understand and support the set of metrics you are gathering.

For example, consider that one of the goals for your project is to improve code quality. One of the questions for this goal may be "What are indicators of code quality?" Metrics you may find supportive of this question include number of bugs, number of particular types of bugs, number of regressions, code coverage results, or test pass/fail rates. There are no magic metrics, and no set of metrics will support the same goal for everyone. In adopting a goal-oriented metrics program, chances are you won't need to define metrics that you aren't already collecting. In fact, you will inevitably end up measuring less than you are with your current program.

I also advise tracking a set of relative metrics for each goal. Even if you are measuring the right things, your metrics project may fail if it is possible to manipulate the value of a metric without providing any real progress. For example, if you are measuring code churn—the percentage of lines changed, the percentage of lines deleted, and the percentage of files changed—this group of related metrics will give you a more valuable view than any one of these measures in isolation.

I recently attended a status meeting for the same project. As in the previous meeting, Ron presented a spreadsheet decorated in stoplight colors, but he began his status presentation by reiterating the top-level goals for the project. Then he showed the status of the metrics that supported each goal. Unlike the previous meeting, everyone in the room knew the relevance of the metrics to the project goals. By the end of the presentation, everyone knew there was still work to do, but they knew exactly what deficiencies existed for each goal. Additionally, everyone viewed these metrics as important and relevant. The quality of the software didn't suddenly get better due to the improved metrics program, but the understanding of the overall quality of the project, including an accurate assessment of the risks, led to a much more successful project.

Some key points to remember as you develop your program are:

- Don't try to measure too much. Just because you *can* measure something doesn't mean that you should.
- Understand the goals of your project before you determine what to measure.
- Once you determine the goals for your project, determine which metrics support these goals. Try to choose from existing metrics rather than defining new ones. The important point to note is that now you know *why* you are using each particular measurement.
- Don't let your metrics define the behavior of your team. If the metrics you have chosen can be modified without showing an increase or decrease in quality, either change the metrics or choose a set of relative metrics that cannot be manipulated.
- Monitor the metrics from version to version throughout the project. Just as you measure the project to assess quality, you should measure the metrics program to define areas for improvement and identify trends you can use to provide better information to the project team. **{end}**

Alan Page is test architect for Microsoft's Engineering Excellence Group, where he works with product teams across the company to identify and promote best practices in testing.