CHAPTER 5

About Management

Fallacy 1 **You can't manage what you can't measure.**

Discussion

The purpose of this saying is to point out that measurement is invaluable to managers. Clearly, managers need to know the answer to questions such as how much, when, and how well. There is a whole software engineering subfield devoted to the topic of software metrics, and proposals of new things to count—and how to count things long understood—are rampant.

What is interesting about the field of software metrics is that it is used little in practice. In surveys of tools and techniques used by software managers, metrics generally come in close to last. There are exceptions, of course—some enterprises (for example, IBM, Motorola, and Hewlett-Packard) place heavy emphasis on metric approaches. But for the most part, metric approaches are soundly ignored. Why is that? Perhaps managers are unconvinced of the value of metrics. Perhaps some of the necessary data is too hard to collect.

But there have been lots of studies of both the value and the cost of metrics, most of which have positive findings. At NASA-Goddard, for example, studies have shown that the ongoing cost of collecting the necessary metrics should be no more than 3 percent (data collection and analysis) + 4 to 6 percent (processing and analyzing the data) = 7 to 9 percent of the total cost of the project (Rombach 1990). NASA-Goddard considers that to be a bargain, given the value of their results.

Some of the history of metrics approaches has been tainted, however. Originally, managers all too often collected data that didn't matter or that cost too much to obtain. Such helter-skelter metrics collection was expensive and, as it turned out, pointless. It wasn't until the notion of the GQM approach (originally proposed by Vic Basili)—establish Goals to be satisfied by the metrics, determine what Questions should be asked to meet those goals, and only then collect the Metrics needed to answer just those questions—that there began to be some rationality in metrics approaches.

There was also the problem of software science. Software science was an attempt by the brilliant computing pioneer Murray Halstead to establish an underlying science for software engineering (Halstead 1977). He defined factors to measure and ways of measuring them. It seemed a worthy and, at the time, an important goal. But study after study of the numbers obtained showed neutral or negative value to the software science data. Some even likened software science to a form of astrology. The collection of "scientific" data about software projects eventually fell into disrepute and has, for the most part, been abandoned. Those who remember the software science debacle tend to taint all software metrics activities with the same brush.

Nevertheless, the collection of software metric data now happens often enough that there is even a "top 10" list of software metrics, the ones most commonly used in practice. To present an answer to the question "what are software metrics?" we present that list here.

Software Metrics	**% Reported Using**
Number of defects found after release	61
Number of changes or change requests	55
User or customer satisfaction	52
Number of defects found during development	50
Documentation completeness/accuracy	42
Time to identify/correct defects	40
Defect distribution by type/class	37
Error by major function/feature	32
Test coverage of specifications	31
Test coverage of code	31

Perhaps equally interesting is the list of the bottom 5 metrics:

Software Metrics	**% Reported Using**
Module/design complexity	24
Number of source lines delivered	22
Documentation size/complexity	20
Number of reused source lines	16
Number of function points	10

(This data comes from Hetzel [1993]. There is no reason to believe that the intervening years since 1993 would have changed this list a great deal, although advocates of function points claim a recent rise in their usage.)

Controversy

The problem with the saying "you can't manage what you can't measure"—what makes it a fallacy—is that we manage things we can't measure all the time. We manage cancer research. We manage software design. We manage all manner of things that are deeply intellectual, even creative, without any idea of what numbers we ought to have to guide us. Good knowledge worker managers tend to measure qualitatively, not quantitatively.

The fact that the saying is a fallacy should not cause us to reject the underlying truth of the message it brings, however. Managing in the presence of data is far better and easier than managing in its absence. In fact, it is the nature of managers—and human beings in general—to use numbers to help us understand things. We love batting and fielding and earned run averages. We love basket and rebound and assist counts and invent terms like *triple double* to accommodate combinations of them. We even invent data to cover subjects when there is no natural data, such as ice skating and diving (where judges assign scores to performances).

This is a case in which the fact is that measurement is vitally important to software management, and the fallacy lies in the somewhat-cutesy saying we use to try to capture that.

Source

The saying "you can't manage what you can't measure" appears most frequently in books and articles on software management, software risk, and (especially)

software metrics. An interesting thing happened when I set out to track down where the saying originally came from. Several metrics experts said that it came from *Controlling Software Projects* (DeMarco 1998), and so I got in touch with Tom DeMarco himself. "Yes," said DeMarco, "it's the opening sentence in my book, *Controlling Software Projects.* But," he went on to say, "the saying is actually 'you can't control what you can't measure.'" Thus the fallacy version of the saying is actually a corruption of what DeMarco really said!

References

- DeMarco, Tom. 1998. *Controlling Software Projects: Management, Measurement, and Estimation.* Englewood Cliffs, NJ: Yourdon Press.
- Halstead, M.H. 1977. *Elements of Software Science.* New York: Elsevier Science.
- Hetzel, Bill. 1993. *Making Software Measurement Work.* Boston: QED.
- Rombach, H. Dieter. 1990. "Design Measurement: Some Lessons Learned." *IEEE Software,* Mar.

Fallacy 2 You can manage quality into a software product.

Discussion

This is a reprise of an idea presented previously in this book. In the section About Quality, I asked the question "whose responsibility is quality? My answer, as you may remember, was that no matter how many people believed" that management was responsible for product quality, there was too much technology to the subject of software quality to leave it up to management. I then went on at that point to say that nearly every one of the quality "-ilities" had deeply technical aspects, aspects that only a technologist could work with.

Not only is the achievement of quality a technical task, but those who believe that it is a management task often go about it in the wrong way. Over the years, managers have tried to use motivational campaigns to instill a quality viewpoint, as if the average technologist would be interested only in quality if he or she were pushed to do so. Sloganeering—"Quality Is Job One"—and methodologizing—"Total Quality Management"—seem to be management's chief approaches to achieving software product quality. Far from accepting these approaches, technol-