1990

Using Metrics to Quantify Development

Harlan D. Mills

P. B. Dyson

USING METRICS TO QUANTIFY DEVELOPMENT

Harlan D. Mills, *Florida Institute of Technology*
Peter B. Dyson, *Software Productivity Solutions*

You can't control what you can't measure. That fundamental reality underlies the importance of software metrics, despite the controversy that has surrounded them since Maurice Halstead put forth his ideas on software science. Skeptics claim metrics are useless and expensive exercises in pointless data collection, while proponents argue they are valuable management and engineering tools.

But what *are* metrics? What benefits do they provide? What are their limitations? Should — and can — I use them on *my* project? Which should I use? How? The six theme articles in this issue explore these questions and provide some answers.

What metrics are. Metrics are simply quantitative measures of certain characteristics of a development project. They may measure any of several things:

- products (like code and documentation),
- the development process (aspects of development activities),
- the problem domain (like telecommunications, management information systems, and process control), or
- environment characteristics (like people, the organization, and tools).

From the software engineer's or manager's perspective, metrics can represent a tool that, when properly used, enhances management control over the development process and product quality. Only in its first human generation, software engineering is an adolescent activity that needs metrics to record and transmit information about what is humanly possible and practical as software engineering becomes a real engineering discipline.

This conceptual view of metrics is both simple and intuitively appealing. In practice, of course, there are many difficult areas that must be addressed.

How to use metrics. First, you must define the framework for your metrics. You cannot apply metrics without first understanding what you want to measure and how you will measure what you want to know about. In "Design Measurement: Some Lessons Learned," H. Dieter Rombach describes how to do this by explaining what he has learned in applying metrics to software design for several projects.

With a framework in place, you must focus your metrics to achieve project-specific results. To do so requires getting the metrics results in a form you can use to monitor and modify your development effort. The earlier in the life cycle you do this, the more control you have on quality, in terms of functionality, reliability, cost, and schedule. Three articles explore different approaches to using metrics to build quality in early during development.

Based on real projects at Hewlett-Packard, Bob Grady describes in "Work-Product Analysis: The Philosopher's Stone of Software?" how to make metrics a useful part of your development process. He focuses on metrics for software complexity, which you can use to flag potentially nettlesome problems early in the life cycle, before they become expensive to undo or patch.

Similarly, in "Predicting Source-Code Complexity at the Design Stage," Sallie Henry and Calvin Selig describe how to use metrics to gauge the quality of your source code at the detailed-design stage, before coding begins. They also evaluate

0740-7459/90/0300/0015/$01.00 © 1990 IEEE

several types of metrics' predictive abilities and present statistics you can use to adapt these metrics types to your projects.

In "Empirically Guided Software Development Using Metrics-Based Classification Trees," Adam A. Porter and Richard W. Selby describe how to identify high-risk components early. These classification trees use data from previous projects and combine several types of metrics, giving you the advantages of building a corporate project memory and being able to use the appropriate metrics for the project at hand.

Metrics let you do more than predict potential problems during development. You can also use them to gauge qualities like reliability. In "Applying Reliability Measurement: A Case Study," Willa K. Ehrlich, S. Keith Lee, and Rex H. Molisani show how one metrics model can accurately predict the number of defects remaining in a system. With such a model, you can know when your software has reached your quality target and stop testing.

These articles show that metrics *are* worthwhile. But implementing a metrics program is not something you can do overnight, especially if your development efforts are the large-scale, mission-critical applications for which quality is a major concern. In "Implementing Management Metrics: An Army Program," Stewart Fenick describes how a US Army organization is instituting a metrics program for mission-critical software development. In explaining the plans, he defines the issues that any development organization must face when implementing a metrics program.

While metrics remain controversial, that controversy is diminishing as research results show their usefulness, as issues of software reliability and safety become part of the public consciousness, and as government and industry realize the growing dependence on software and the concomitant need to measure software's functionality, reliability, cost, and schedule. Remember: You can't control what you can't measure. ❖

Acknowledgments

We thank those who made this metrics theme section possible: the 53 authors who submitted articles and the 38 referees who did a terrific job reviewing them.

Harlan D. Mills is a professor of computer science at Florida Institute of Technology and president of Software Engineering Technology. His interests include mathematical foundations for software engineering and software engineering under statistical quality control.

Mills received a PhD in mathematics from Iowa State University. He has been named a fellow by IEEE, IBM, Association of Computer Programmers of America, and Wesleyan University, and he has received the Data Processing Management Association's Distinguished Information Scientist Award and the Warnier Prize.

Peter B. Dyson is vice president of Software Productivity Solutions. His interests include metrics, software quality, software management, and software-engineering environments and tools.

Dyson received a BS in computer science from Washington University in St. Louis and an MBA from Florida Institute of Technology.

Address questions to Mills at Software Engineering Technology, 2770 Indian Harbour Blvd., Vero Beach, FL 32960 or to Dyson at Software Productivity Solutions, 122 N. Fourth Ave., Indialantic, FL 32903; Milnet dyson@radc-softvax.arpa.