Guest Editors' Introduction

Software Metrics: Charting the Course

Taghi M. Khoshgoftaar, Florida Atlantic University
Paul Oman, University of Idaho

Software has become critical to commercially vital technologies that account for 12 million jobs and one trillion dollars of sales in the US alone. And it continues to become central to more products, making many new ones possible and reducing cost or increasing functionality in existing ones. In addition, software products contribute indirectly to the delivery of many other important products and services such as banking, telecommunications, and transportation.

For these reasons, software engineers face challenging times. As society becomes more dependent on software, engineers must increase customer satisfaction, boost software-development productivity rates, and reduce product-defect rates. Because software directly affects product users, the responsibility for customer satisfaction is increasing. With so many tasks dependent upon software, small increases in software development productivity can yield large economic gains, while decreases in software-defect rates can prevent large economic losses. Most importantly, where software supports life-critical tasks or life-saving research, decreasing defect rates and increasing productivity rates will save lives.

International expansion is another factor. Nations now viewed as potential software consumers will themselves become software producers once they comprehend the huge economic role of software production. These new sources of products will further broaden the choices for consumers and motivate the industry to stress customer satisfaction.

Answering the call. Software measurement is essential in our efforts to meet these challenges. Measurement enables engineers to quantify product reliability and per-
**General glossary**

The following definitions have been slightly modified for this issue from the IEEE Standard for a Software Quality Metrics Methodology, IEEE Std 1061-1992, 1993.

- **Critical range** — Metric values used to classify software into categories of acceptable, marginal, or unacceptable.
- **Critical value** — Metric value of a validated metric that is used to identify software that has unacceptable quality.
- **Direct metric** — Metric applied during development or during operations that represents a software quality factor (such as mean time to software failure for the factor reliability).
- **Factor sample** — Set of factor values drawn from the metrics database and used in metrics validation.
- **Factor value** — Value of the direct metric that represents a factor (see metric value).
- **Measure** — A way to ascertain or appraise value by comparing it to a standard. To apply a metric.
- **Measurement** — The act or process of measuring. A figure, extent, or amount obtained by measuring.
- **Metric** (see software quality metric)
- **Metrics framework** — A tool for organizing, selecting, communicating, and evaluating the required quality attributes for a software system. A hierarchical breakdown of factors, subfactors, and metrics for a software system.
- **Metrics sample** — A set of metric values that is drawn from the metrics database and used in metrics validation.
- **Metric validation** — The act or process of ensuring that a metric correctly predicts or assesses a quality factor.
- **Metric value** — A metric output or an element that is from the range of a metric.
- **Predictive assessment** — The process of using a predictive metric(s) to predict the value of another metric.
- **Predictive metric** — A metric applied during development and used to predict the values of a software quality factor.
- **Process metric** — A metric used to measure characteristics of the methods, techniques, and tools employed in developing, implementing, and maintaining the software system.
- **Process step** — Any task performed in the development, implementation, or maintenance of software (for example, identifying the software components of a system as part of the design).
- **Product metric** — A metric used to measure the characteristics of the documentation and code.
- **Quality attribute** — A characteristic of software, or a generic term applying to factors, subfactors, or metric values.
- **Quality factor** — A management-oriented attribute of software that contributes to its quality.
- **Quality requirement** — A requirement that a software attribute be present in software to satisfy a contract, standard, specification, or other formally imposed document.
- **Quality subfactor** — A decomposition of a quality factor or quality subfactor to its technical components.
- **Sample software** — Software selected from a current or completed project from which data can be obtained for use in preliminary testing of data collection and metric computation procedures.
- **Software component** — A general term used to refer to a software system or an element, such as module, unit, data, or document.
- **Software quality metric** — A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality.
- **Validated metric** — A metric whose values have been statistically associated with corresponding quality factor values.

Performance, to isolate development process and product attributes that impact reliability and performance, and to demonstrate how process and product changes impact these attributes (see the glossary for some general terms). Further, measurement lets development teams:

- set achievable goals,
- demonstrate their potential to meet these goals,
- track their progress,
- adjust processes to correct out-of-bounds conditions, and
- demonstrate the impacts of these adjustments on meeting goals.\(^3\)

**Special issue.** This month’s *Computer*
demonstrates the role of software measurement in managing projects, improving software performance and reliability, and evaluating changes for improving the software development process.

Our role as guest editors was both easy and difficult. It was easy because we received so many excellent articles. Difficult because out of 42 we could accept only seven for publication, an acceptance rate of only 17 percent.

Guiding our selection was a desire to present a balanced portrait of the metrics and measurement arena. This meant including articles on management aspects, others directed toward the “line engineer,” and still others addressing the more esoteric ideals of standards and validity.

The first article, R. Grady’s “Successfully Applying Software Metrics,” describes software product and process metrics that can be used to measure success. The major uses of software metrics are presented by examples from Hewlett-Packard software development organizations and elsewhere.

The next article, “Using Metrics to Manage Software Projects,” by E.F. Weller, discusses how the defect data from inspections and test can be used to plan and track software projects and improve product quality.

D.L. Lanning and T.M. Khoshgoftaar’s “Modeling the Relationship Between Source Code Complexity and Maintenance Difficulty” applies canonical correlation analysis to study a relationship
that might not otherwise be directly observable. The authors use product and process measures collected during the development of a large commercial real-time product for their study.

"Using Metrics in Management Decision Making" by G. Stark, R. Durst, and C.W. Vowell describes a metrics program implemented by managers of NASA's Mission Operations Directorate (MOD) at the Johnson Space Center. The authors show that the metrics program within MOD has helped managers and software engineers to better understand their processes and products.

"Case Studies of Software-Process Improvement Measurement" by D.J. Paulish and A.D. Carleton presents a basic set of measures for software-development organization performance and a software process improvement method. Some Siemens software development organizations served as case study sites for examining the impact of proposed process-improvement methods.

"Achieving Software Quality with Testing Coverage Measures" by J.R. Horgan, S. London, and M.R. Lyu describes the principles of dataflow testing and a software analysis tool that measures the effectiveness of testing. The authors present two case studies in which they analyze the relationship between software quality and test coverage metrics.

Finally, in "Evaluating Software Engineering Standards," S.L. Pfleeger, N. Fenton, and S. Page present a framework for assessing the quality of software engineering standards and methods. Using two numerical measures that differentiate levels of objectivity and areas of application, they contrast software engineering standards with other engineering standards.

We think this is a well-balanced collection of articles. We hope you find these examples applicable to your work environment.

References


Readers can contact Khoshgoftaar at the Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431.