Foundations for a measurement discipline

Measurements have traditionally played an important role in science and engineering. We expect experimentation and data collection to figure similarly in our field. The following report by Bill Curtis discusses some of the problems in measuring software engineering phenomena. His implicit advice: Not all numbers are created equal. The value of a particular set of data depends, to a large extent, on the methods used in its collection and the model used for its interpretation.

—Editor

Bill Curtis, MCC Software Technology Program

Although many companies now collect data from their software projects, debate continues over the validity of software measures and their interpretation. The scientific foundation for software measurement practice grows slowly because, in part, the empirical research methods used have progressed little since this decade began. This slow progress limits the benefits that industry can get from metrics.

Two problems. There are two issues that must be addressed to improve the discipline of software measurement: the theoretical framework and the training of empirical researchers.

The conceptual paradigm displayed in many software measurement studies reflects little growth in the theoretical models used a decade ago. For instance, software metrics are often criticized for failing to predict variations in performance that are more properly attributed to variations in the skill and experience of software engineers. When used to predict the difficulty of working with a program, software complexity should be operationally defined as a composite measure computed from an interaction between the characteristics of both a program and a software engineer.

A more theoretically complete paradigm depicting a program's psychological complexity must account for the interaction of human information processing and the software attributes. Investigators who want to predict the comprehensibility of a program should become familiar with the research describing why lexical and syntactic measures of text have not been more successful in predicting the comprehensibility of prose. In fact, many areas of software engineering need measures derived from theories developed in other disciplines that explain significant amounts of the variation in measures of software productivity, quality, comprehensibility, and so on.

To improve the theoretical formulation and analysis of software-engineering data, the training of empirical researchers must become more interdisciplinary. In particular, students performing empirical research in software engineering need better formal training in empirical methodology. At a minimum, their curriculum should include courses in the design of experiments, measurement theory, univariate and multivariate statistics, and advanced topics relevant to their research area, such as structural equation models and stochastic methods.

Study after study reports large correlations among software metrics such as lines of code and the formulas of Halstead, McCabe, Albrecht, and others. Were these data reduced to a smaller set of summary measures through factor analysis (a multivariate statistical technique), the first factor emerging from most data sets would account for a huge amount of the common variance among metrics. This factor would indicate that the software component's size is the primary attribute assessed through the operational definitions of many metrics. The pervasiveness of the effect of size on software metrics has made it difficult for most studies to demonstrate that metrics perform better than lines of code in predicting many criteria.

Better empirical training will give researchers not only a stronger methodological basis for software measurement, but it will help them propose and explore more appropriate theoretical models of software-engineering phenomena.

Current practice. If research has progressed slowly in providing better software measures, can there be any good news for measurement practice? Yes, but it is guarded. Even crude measures have been able to detect statistically significant relationships. Many skilled practitioners have been able to calibrate their measures over time to learn how they behave in a particular environment. This calibration lets many problems with existing measures be specified and accounted for when interpreting data, especially problems characteristic of extreme cases.

The use of software measurements by practitioners and managers has similarities to the use of diagnostic measures by medical and psychological clinicians. Limits to the validity and interpretability of many diagnostic measures continue to provoke scientific debate. Yet in the hands of a skilled practitioner, diagnostic indicators can yield remarkable insight.

The key point is skilled practitioner. Skilled clinicians have undergone extensive professional training and have practical experience observing how their indicators (often developed through years of rigorous empirical research) vary over both normal and pathological conditions in a variety of situations. In the clinical sense, software measures must be defined and used by skilled practitioners who are sensitive to how the measures behave under a variety of conditions across many projects.

The abuses that many fear from the unsophisticated use of software measures occur most often when inexperienced, unskilled practitioners use them as absolute, causal measures rather than as diagnostic indicators that must be interpreted in context. One such abuse is the strict limit of module sizes below some value when the metric used has only shown modest relationships with com-

Continued on p. 92
Software Engineering Opportunities

RADE™, a Micrognosis product, is an integrated workstation that is designed to put the active traders' need for fast breaking information at their fingertips. Leading banks, brokerage firms, insurance companies use Micrognosis systems as their link to the world of financial information.

We have exciting opportunities for qualified professionals in the following areas:

Software Manager
Product Engineering
You will be responsible for the supervising of project teams during all phases of software enhancements.

An ability to set and meet full project lifecycle goals and objectives in a highly technical environment is required. We expect you to have a solid knowledge of "C" and PLM operating under UNIX® in an on-line environment.

Software Manager
Research & Development
You will direct project teams involved in the development of state-of-the-art, microprocessor-driven software for the Financial Services Industry.

You must possess strong analytical, conceptual and technical management skills and a knowledge of "C", PLM and data communications under UNIX and VMS.

Group Leader
Software Systems Test
You will be responsible for the verification, validation and evaluation of product and system releases.

You must have at least 6-8 years experience with a previous supervisory background, and experience in microprocessor-based systems with knowledge of VAX and microVAX, UNIX-based with "C" experience a plus.

Senior Systems Test Engineers

Preferably, you should have 5-7 years experience in the design, implementation and test of microprocessor-based or VAX-based software systems.

A knowledge of various software systems test methodologies is required along with a knowledge of "C" and data communications under UNIX would be a plus.

Group Leader
Software Test Tools & Simulation

To design, develop and customize test tools and simulation for microprocessor-based and VAX-based real-time distribution systems.

For this senior level position, with group leader responsibilities, you must possess 7-10 years experience and a knowledge of VAX, UNIX and "C."

For all of the above positions a BSCS or BSEE or its equivalent is required or preferred.

IT'S ABOUT TIME!

Micrognosis can offer you an outstanding salary, comprehensive benefits and exceptional opportunities for professional growth.

Join a company that can provide real career momentum. It's time to send your resume, indicating position of interest, salary history and requirements in confidence to: Vito Santoro, Manager of Staffing & Development, MICROGNOSIS, Inc., Dept. IEEE 1001, 100 Saw Mill Road, Danbury, CT 06810. An equal opportunity employer.

Applying technology to trading

*UNIX is a trademark of AT&T Bell Laboratories

QUALITY TIME

Continued from p. 89

prehensibility, modifiability, defect-proneness, and the like.

The future. Improving the use of software measurements requires that the relationship between the research and industrial communities continues to grow. The research process too often suffers from little resources. Industry can enhance its future benefits from better software measurement by considering itself as a critical resource that the research community must be able to access to improve both training and the external validity of empirical research. To avoid an empirical literature dominated by studies of student programming, industry must provide researchers with access to professional programming.

Companies that have supported a serious measurements program and have understood the importance of a research component will be able to reap the benefits of greater understanding, predictability, and control of their software life-cycle process.

HUMAN FACTORS

Continued from previous page

trials in a controlled experiment to determine that one way of presenting information to users results in faster performance than another arrangement.

Experiments and task analysis usually require strict controls. Questionnaires and field studies are, by nature, less robust. Iterative testing can be more or less robust depending on the degree of experimental control exercised during the test. A major advantage of iterative design is that the large problems are revealed early in the development process and smaller refinements are discovered after those have been corrected.

Working together. Human factors issues crop up at all stages of development. Therefore, the best results are obtained when system designers and human factors professionals collaborate throughout a design project. With human factors research, models, standards, guidelines, principles, methods, techniques, and tools, designers can make better, more objective decisions about how the system should work.

Looking at the full range of human factors inputs makes it clear that the fit between human factors engineering and software development certainly is a tight one.