Empirical Software Engineering, 6, 201–205, 2001.

Knowledge Requirements for Software Quality Measurement

NORMAN F. SCHNEIDEWIND nschneid@nps.navy.mil
Division of Computer and Information Sciences and Operations, Naval Postgraduate School, Monterey, CA 93943

1. Introduction

Because measurement of quality is the key to achieving high quality software, it is important for software engineers to be knowledgeable in this area. We identify the body of knowledge in software quality measurement that is required of the software engineer. One approach is to develop a set of related issues, functions, and body of knowledge. Issues determine the functions performed by the software engineer and these functions, in turn, determine knowledge requirements. The second approach identifies knowledge requirements by keying the knowledge set to the phases and metrics used in each phase. The two approaches are compatible but different views of achieving the same objective: the former indicates *why* a need for measurement arises and the latter shows *when* the need arises.

2. Issue Oriented Approach

Issues arise because there are important considerations in achieving software quality goals at acceptable cost. An example of the integration of Issue, Function, and Knowledge is shown in Table 1. This table shows the functions that would be performed by the software engineer in executing a life cycle quality management plan, oriented to the issues in the first column.

The following is a brief example of the issues, functions and knowledge requirements listed in Table 1.

Issue 1: *Quality Engineering* is a systematic approach to establishing quality requirements and identifying, implementing, analyzing, and validating the process and product software quality metrics for a software system (Standard for Software Quality Metrics Methodology, 1998). Metrics are identified and data collection plans are developed for satisfying quality goals. Criteria are identified for measuring and interpreting conformance with quality requirements during inspection and testing.

Table 1. Knowledge requirements in software quality measurement.

Issue	Function	Knowledge
1. *Goals:* What are the software quality goals of the system?	*Analyze* quality goals and *specify* quality requirements.	Quality engineering Requirements engineering
2. *Cost and risk:* What is the cost of achieving quality goals and the risk of not achieving them?	*Evaluate* economics and risk of quality goals.	Economic analysis Risk analysis
3. *Context:* What application and organizational structure is the system and software to support?	*Analyze* the application environment.	Systems analysis Software design
4. *Operational profile:* What are the criticality and frequency of use of the software components?	*Analyze* the software environment.	Probability and statistical analysis
5. *Models:* What is the feasibility of creating or using an existing quality model for assessment and prediction? How can the model be validated?	*Model* quality and *validate* the model.	Probability and statistical models
6. *Data requirements:* What data are needed to support quality goals?	Define data type, phase, time, and frequency of collection.	Data analysis
7. *Types and granularity of measurements:* What measurement scales should be used? What level of detail is appropriate to meet a given goal? What can be measured quantitatively, qualitatively, or judgmentally?	*Define* the statistical properties of the data.	Measurement theory
8. *Product and process test and evaluation:* How can product quality measurements be fed back to improve process quality?	*Analyze* the relationship between product quality and process stability.	Inspection and test methods
9. *Product and process quality prediction:* What types of predictions should be made to assess and predict and process quality?	*Assess* and *predict* software quality.	Measuremet tools

In *Requirements Engineering*, quality requirements are identified that are applicable to the software system. User expectations, organizational experience, required standards, regulations, or laws are used to create this list. In addition, contractual requirements and acquisition concerns, such as cost or schedule constraints, and warranties, are considered.

3. Life Cycle Oriented Approach

A simplified life cycle measurement process is shown in Figure 1. Placing the knowledge requirements in the context of a time line provides visibility of several aspects of software measurement that are not revealed in Table 1. One aspect is that the artifacts of the development process are included in this approach, as shown in Figure 1. In addition, it is important to understand that measurements obtained during the early part of the life cycle are less quantitative than those obtained later. This fact constrains our ability to make meaningful assessments and predications of quality for either the product or the process. The reason is that both the product and the process evolve over time and life cycle phases such that the objects we measure during test and operation are not the same objects that were measured during requirements analysis. This is due not only to the objects being different (e.g., requirement specification versus code) but is also the result of changes to requirements and design approaches during the life cycle. Also, note that early measurements are made on static artifacts (e.g., design documents) and later measurements are made on dynamic artifacts (e.g., code in execution).

These characteristics add up to opportunities for early assessment of quality but using less accurate measures of quality than are obtained later in the life cycle. This is analogous to assessing an automobile's performance and quality by reading the manufacturer's brochures as opposed to taking a test drive of the vehicle. Thus, we have identified an important principle of software measurement: the use of *surrogate measures* that are available early in an evaluation process and are hypothesized to be

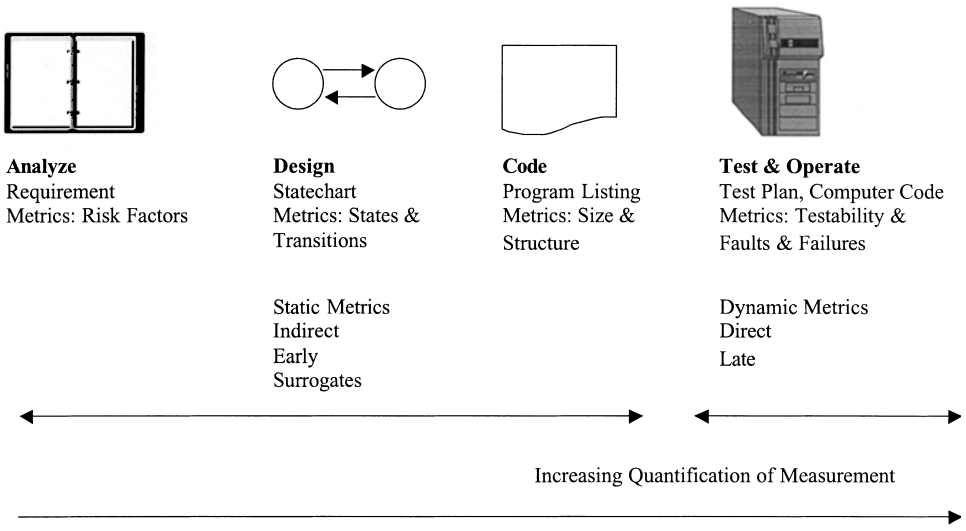

Figure 1. Life cycle measurement attributes.

representative of the measures of real interest (e.g., code complexity as a surrogate for reliability; automobile specifications as a surrogate for driving experience).

4. Summary and Recommendations

We have briefly identified the core body of knowledge in software measurement by using two approaches—issue oriented and life cycle phase oriented. The former addresses major measurement problems in the software industry and the latter puts measurement in a project time perspective. The measurement functions that have been identified are: analyze, specify, evaluate, model, validate, define, assess, and predict. The knowledge areas that have been identified are: quality engineering, requirements engineering, economic analysis, risk analysis, systems analysis, software design, probability and statistical analysis, probability and statistical models, data analysis, measurement theory, inspection and test methods, and measurement tools. We recommend that this body of knowledge becomes a part of the core required for certification and licensing of software engineers.

Reference

Standard for a Software Quality Metrics Methodology, Revision, 1998. *IEEE* Std 1061–1998, 31.

Dr. Norman F. Schneidewind is Professor of Information Sciences and Director of the Software Metrics Research Center in the Division of Computer and Information Sciences and Operations at the Naval Postgraduate School, where he teaches and performs research in software engineering and computer networks. Dr. Schneidewind is a Fellow of the IEEE, elected in 1992 for "contributions to software measurement models in reliability and metrics, and for leadership in advancing the field of software maintenance". He is the developer of the Schneidewind software reliability model that is used by NASA to assist in the prediction of software reliability of the Space Shuttle, by the Naval Surface Warfare Center for Trident Software reliability prediction, and by the Marine Corps Tactical Systems Support Activity for distributed system software reliability assesment and prediction. This model is one of the models recommended by the American National Standards Institute and the American Institute of Aeronautics and Astronautics Recommended Practice for Software Reliability. In addition, the model is implemented in the *Statistical Modeling and Estimation of Reliability Functions for Software* (SMERFS), software reliability modeling tool. He has published widely in the fields of software reliability and metrics.

In 1993 and 1999, he received an award for outstanding research achievements by the Naval Postgraduate School. He was Chairman of the Working Group that produced the IEEE Standard 1061–1992, Standard for a Software Quality Metrics Methodology and its revision in 1998. In 1993 he was given the IEEE Computer Society's Outstanding Contribution Award "for work leading to the establishment of IEEE Standard 1061–1992". In addition, he was given the IEEE Computer Society Meritorious Service Award "for his long-term committed work in advancing the cause of software engineering standards". He was recognized for his contributions to the IEEE Computer Society by being named to the "Golden Core" of volunteers. He serves as Associate Editor, IEEE Transactions on Software Engineering.