A FRAMEWORK FOR THE MEASUREMENT

OF SOFTWARE QUALITY

Joseph P. Cavano
Rome Air Development Center

James A. McCall
General Electric Company

ABSTRACT

Research in software metrics incorporated in a framework established for software quality measurement can potentially provide significant benefits to software quality assurance programs. The research described has been conducted by General Electric Company for the Air Force Systems Command Rome Air Development Center. The problems encountered defining software quality and the approach taken to establish a framework for the measurement of software quality are described in this paper.

INTRODUCTION

We are all aware of the critical problems encountered in the development of software systems: the estimated costs for development and operation are overrun; the deliveries are delayed; and the systems, once delivered, do not perform adequately. Software, as such, continues to be a critical element in most large-scale systems because of its cost and the critical functions it performs. Many of the excessive costs and performance inadequacies can be attributed to the fact that "software systems possess many qualities or attributes that are just as critical to the user as the function they perform" (Ref 1). For this reason, considerable emphasis in the research community has been directed at the software quality area.

The Air Force, as well as the rest of DOD and industry, is constantly striving to improve the quality of its computer-based systems. Producing high quality software is a prerequisite for satisfying the stringent reliability and error-free requirements of command and control software. Increasingly tight budgets necessitate getting the highest quality software products at the best possible cost. A major difficulty in dealing with software, however, is that there are no quantitative measures of the quality of a software product. This affects the military Command-Control-Communications-Intelligence (C^3I) environment where the requirements for software quality far exceed the demands of the commercial world. The basic resources available for accomplishing each military mission are often specified by agencies external to the responsible organization (i.e., funding by Congress and technology by the laboratories). Thus, the organization must optimize its performance within a limited set of resources. For the development of a software system, this optimization revolves around producing software that fulfills the mission requirements. In order to know that this has been done successfully, the software development should be periodically measured in a quantitative fashion to determine whether the final system will be capable of meeting its objectives.

One problem in making this determination is the absence of a widely accepted definition of software quality. This leads to confusion when trying to specify quality goals for software. A limited understanding of the relationships among the factors that comprise software quality is a further drawback to making quality specifications for software.

A second current problem in producing high quality software is that only at delivery and into operations and maintenance is one able to determine how good the software system is. At this time, modifications or enhancements are very expensive. The user is usually forced to accept systems that cannot perform the mission adequately because of funding, contractual, or schedule constraints.

Since software testing alone does not produce or ensure good software -- it only gives an indication of error frequency that can be expected -- and since verification only shows correspondence to functional requirements, a new process is needed to measure and represent the qualities of a software system. This process should indicate which software characteristics relate directly to mission requirements and serve to define a variety of quality factors: maintainability, reliability, flexibility, correctness, testability, portability, reusability, efficiency, usability, integrity, and interoperability. The process of software quality measurement may become a new function within the domain of quality assurance. The quantification of these measurements can be compared to mission requirements to determine if those requirements are being met.

The quality measurement process must be able to be applied during the requirements and design phases of software production; this key aspect further distinguishes it from the testing and verification activities. The quality measurements are predictive in nature and oriented toward the development phases rather than toward the finished system. Early measurement will give an indication of how

well the software product will operate in relation to the quality requirements levied on it. In other words, an initial assessment will be made of the quality of the software system. By obtaining such an assessment before testing or final delivery, faults or inadequacies can be identified and corrected early enough in the development process to result in large cost savings.

The framework for the measurement of software quality was established to be useful at two different levels of application: management and quality assurance. At the management level, the software quality factors are user-oriented and can be directed toward meeting the objectives of the system. At the quality assurance level, software-oriented metrics attempt to objectively measure specific elements at both the module and the system level and relate these to the software quality objectives. This paper is concerned mostly with the latter function.

QUALITY AS A RELATIVE MEASURE

The determination of "quality" is a key factor in everyday events -- wine-tasting contests, sporting events, beauty contests, etc. In these situations, quality is judged in the most fundamental and direct manner: side by side comparison of objects under identical conditions and with predetermined concepts. The wine may be judged according to clarity of color, bouquet, taste, etc. However, this type of judgment is very subjective; to have any value at all, it must be made by an expert.

Subjectivity and specialization also apply to determining software quality. To help solve this problem, a more precise definition of software quality is needed as well as a way to derive quantitative measurements of software for objective analysis. A major question at this point is whether software can be measured at all. A number of studies indicate that the answer to this question is yes (Refs 2, 3), but it is a qualified yes. Since there is no such thing as absolute knowledge, one should not expect to measure software quality exactly, for every measurement must be partially imperfect. Jacob Bronowski described this paradox of knowedge in this way: "Year by year we devise more precise instruments with which to observe nature with more fineness. And when we look at the observations, we are discomfited to see that they are still fuzzy, and we feel that they are as uncertain as ever. We seem to be running after a goal which lurches away from us to infinity every time we come within sight of it." (Ref 4).

Consequently, any measurement of software must be somewhat imprecise. This promotes areas of uncertainty surrounding the meaurement, so a confidence level must be established to allow for tolerance in software measurement. The real goal of software measurement lies in determining what this area of tolerance might be and how it might affect the use of the measurement.

For instance, if precise results are unattainable, does one still wish to expend energy and money to make these measurements? The answer to this is not always clear, but for some applications even a slight indication is better than no indication. Or as Reichenbach states: "Every act of planning requires some knowledge of the future and if we have no perfectly certain knowledge, we are willing to use probable knowledge in its place" (Ref 5).

DIFFICULTY IN ASSESSING SOFTWARE QUALITY

Software has always been viewed as an abstraction. Unlike hardware, it has no physical presence. This concept has contributed to the difficulty in assessing the quality of software. The difficulties manifest themselves in several ways. To illustrate, a few examples will be described.

If the maintainability of a program is to be assessed (maintainability being one of the quality factors), one might construct a hypothesis which states that as the number of unconditional branches in a program increases, the more difficult it will be to maintain the program.

However, the exact form of the relationship between maintainability and the number of unconditional branches is not known (or even that it exists). There may be an isotonic increasing function, and for each unconditional branch, the degree of difficulty for maintenance increases by some delta or the relationship may be in the form of a step function where at certain threshold values the degree of difficulty takes a quantum jump. The hope is that the specific relationships can be discovered and converted into meaningful ratings for the top level qualities. For maintainability, this rating might be in terms of the average number of person-days needed to fix an error. Of course, at this time what encompasses a good number for a rating like that (is a person-day good for maintenance or should it be 2 person-days) is not well known. Baselines are desperately needed to fill this gap.

This leads us into still another problem when considering software quality. Since the quality is application-oriented (i.e., the requirement for reliability must be higher for a manned space flight than for computer-aided instruction), the user must be able to clearly state his quality objectives. This is not always easy to do. Guidelines to assist in defining these application-oriented quality requirements are needed.

For instance, consider two application programs, A and B, which were given the same problem requirement, written in the same language, and implemented on the same computer. Program A runs 10 percent faster, has 5 percent fewer errors under identical testing conditions, and costs 20 percent less than Program B and is similar in maintainability and documentation aspects. Which program has the higher quality?

An impulsive answer would be Program A. However, how can one be sure that the testing on the two programs was really identical? And what does one mean by "10 percent faster?" Perhaps Program B was developed to execute in half the core-space as Program A. Now a completely different

interpretation is possible; with a constraint like that, Program B may well be considered to have the higher "quality." Likewise, another interpretation might result if a different application was chosen. For instance, if one knew that Program A was designed to operate on only one machine while Program B was built for a distributed system, quality measurements would be interpreted differently according to the users.

It is easy to see that assessing software quality quickly becomes very difficult. One reason for this is that the same function or algorithm can be implemented in many forms and it is not always clear which form is best. Another reason is that the complexity and interactions involved in large-scale software developments increase nonlinearly with size (Ref 1). And finally, documentation must be considered an integral part of software. In fact, software can be considered to consist almost entirely as documentation. From requirements specification to the coded program, software exists primarily as a written document. There are few proven techniques for determining quality for written works.

The progress being made in the measurement of software quality is due primarily to the use of new software engineering techniques. As more disciplined, engineering approaches, tools, and methodologies are developed and followed in the production of software, the software products themselves become more orderly and rigorous. As a result of this, certain aspects of quality can be measured in more objective, quantifiable ways. By breaking down the quality of software into its component factors, one can arrive at several aspects of software that can be analyzed quantitatively. This decomposition has been the primary research interest in quality measurement.

Research sponsored by the Air Force has led to a proposed software measurement model which contains a comprehensive, hierarchical definition of software quality (figure 1). At the highest level, quality factors are defined that are appropriate for software acquisition managers to use

as an aid in specifying quality objectives for their software systems. These high level factors are then broken down into criteria and subcriteria that are more software-directed until specific metrics (actual, quantifiable measurements) are proposed that relate to the factors. These metrics are based on suggested programming practices in the literature. By making these measurements, it is believed that a corresponding measurement or rating will be obtained for the quality factor. The current state of this research is that few of the metrics have been either proven or disproven. The current state of work in this area is completely described in reference 2.

Based on these discussions, the following three points must be considered in measuring software quality:

    1. To determine the quality of software, predetermined attributes must be measured in a consistent fashion.

    2. A relationship must be developed between the product to be measured and the application that will use it.

    3. A predictive rating of software quality is not absolute. It is an indication of the quality of the end product.

AN APPROACH TO QUANTIFICATION

The framework established (figure 1) is conducive to the quantitative measurement of software quality. The approach to quantitatively measuring software quality utilizing this framework will be discussed in this section of the paper.

At the highest level, the major aspects (factors) of software quality are identified. In identifying and defining these factors, the user and use of these factors has to be considered.

The user is the program manager or acquisition manager, the customer of the software system

FACTOR

MANAGEMENT-ORIENTED VIEW OF
PRODUCT QUALITY

CRITERIA

SOFTWARE-ORIENTED ATTRIBUTES
WHICH PROVIDE QUALITY

METRICS

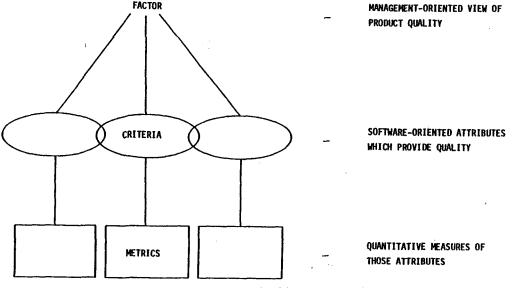QUANTITATIVE MEASURES OF
THOSE ATTRIBUTES

Figure 1. Software Quality Framework

developer. The user requires a defined set of factors in order to identify what qualities are desired in the software product being developed. To satisfy this use, the definitions of the factors must lend themselves to quantification (measurement) that is meaningful to the user.

The approach taken to satisfy these two requirements was to evaluate how a program manager views the end product of a software development. The orientations or viewpoints identified relate to life cycle activities involving the software product. These activities and the quality factors associated with them are shown in figure 2.

The questions in parentheses briefly indicate the relevancy of the factor to the user. The formal definitions of these factors are in table 1.

Underlying these user-oriented quality factors is a set of attributes which, if present in the software, provide the characteristics represented by the factors. For each factor then, a set of criteria has been established and defined.

A key point in the approach should be noted here. The measurements are to be taken during the development effort. These measurements are not post-implementation assessments of software quality. They are not test-like measurements. Their purpose is to provide an indication of the progression toward a desired level of quality. The set of attributes, or criteria, established for each quality factor then represents attributes which can be measured during the software development.
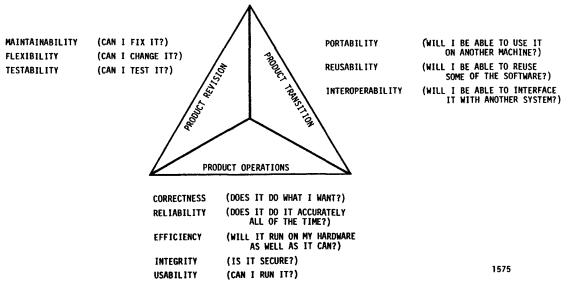
MAINTAINABILITY (CAN I FIX IT?)
FLEXIBILITY (CAN I CHANGE IT?)
TESTABILITY (CAN I TEST IT?)

PORTABILITY (WILL I BE ABLE TO USE IT ON ANOTHER MACHINE?)
REUSABILITY (WILL I BE ABLE TO REUSE SOME OF THE SOFTWARE?)
INTEROPERABILITY (WILL I BE ABLE TO INTERFACE IT WITH ANOTHER SYSTEM?)

PRODUCT REVISION
PRODUCT TRANSITION
PRODUCT OPERATIONS

CORRECTNESS (DOES IT DO WHAT I WANT?)
RELIABILITY (DOES IT DO IT ACCURATELY ALL OF THE TIME?)
EFFICIENCY (WILL IT RUN ON MY HARDWARE AS WELL AS IT CAN?)
INTEGRITY (IS IT SECURE?)
USABILITY (CAN I RUN IT?)

1575

Figure 2. Software Quality Factors

Table 1. Definition of Software Quality Factors

CORRECTNESS	Extent to which a program satisfies its specifications and fulfills the user's mission objectives.
RELIABILITY	Extent to which a program can be expected to perform its intended function with required precision.
EFFICIENCY	The amount of computing resources and code required by a program to perform a function.
INTEGRITY	Extent to which access to software or data by unauthorized persons can be controlled.
USABILITY	Effort required to learn, operate, prepare input, and interpret output of a program.
MAINTAINABILITY	Effort required to locate and fix an error in an operational program.
TESTABILITY	Effort required to test a program to ensure it performs its intended function.

Table 1. Definition of Software Quality Factors (Continued)

FLEXIBILITY	Effort required to modify an operational program.
PORTABILITY	Effort required to transfer a program from one hardware configuration and/or software system environment to another.
REUSABILITY	Extent to which a program can be used in other applications - related to the packaging and scope of the functions that programs perform.
INTEROPERABILITY	Effort required to couple one system with another.

Software quality metrics, when established, provide measures of the software attributes. The metrics may be in the form of a checklist used to "grade" a document produced during the development or particular count of specific attributes such as the number of paths through a module or the number of unconditional branches in a program. Many of the metrics incorporated in the framework have resulted from efforts of the research community in recent years (Refs 6, 7, 8, 9).

Formal relationships between the set of metrics related to a quality factor and a rating of the quality factor have been established via regression analyses performed on empirical data. These relationships take the form of a linear equation. An example is shown here:

$$r_f = c_1 m_1 + c_2 m_2 + c_3 m_3 + \ldots$$

where:

r_f is a rating of a quality factor, f

c_i are the regression coefficients

m_i are the various measurements identified as relating to the quality factor, f.

This relationship, once established, is then used as a predictor. The measurements, m_i, are applied at specific times during the development. The major aspects of this approch are:

- User-oriented at highest level
- Software-oriented at lower levels
- Provides quantification of the attributes
- Can be applied periodically during software development
- Additional metrics, criteria, or even factors can be added as the technologies of producing software change and as research efforts identify better measures

This approach avoids several pitfalls encountered by other efforts in this area in recent years. It does not attempt to utilize a single measurement to quantify quality. It does not rely on measures applied only to the source code, but also to the documentation associated with the software which adds signficantly to its quality, especially for such factors as maintainability, testability, or portability. The metrics have been established as language-independent measures. And lastly, a rule was used in choosing the units of the metric. The rule, the units of the metric will be the number of occurrences of an attribute divided by the total possible occurrences of that

attribute, essentially normalizes the measurements, and discounts biases introduced by size.

The formal relationships established are based on several large-scale Air Force software developments. They are not claimed as generally applicable relationships. Continuing efforts are underway to establish the metrics applicability in other environments such as support software and management information software. The concern in this initial effort was to establish that the concept was viable. The fact that some metrics were found to exhibit significant correlation to qualities demonstrated during the operational history confirmed our hypothesis (Ref 10). Future research efforts are planned to further extend these concepts and pursue the true or more accurate relationships. The fact that the framework has been established .and is conducive to the introduction of new findings facilitates these research efforts.

The framework is essentially a model of software quality. It potentially extends the scope of quality assurance activities. It quantifies the definition of software quality. It supports the collection of data, the documentation of lessons learned, and therefore its own evolution as an up-to-date software quality assurance tool and methodology.

IMPACT ON SOFTWARE QUALITY ASSURANCE

The framework established has several potentially significant impacts on quality assurance activities during large-scale software developments.

First, the framework provides a mechanism for a program manager to identify what qualities are important. These qualities are attributes of the software in addition to its functional correctness and performance which have life cycle implications. Such factors as maintainability and portability have been shown in recent years to have significant life cycle cost impact. Software quality assurance personnel therefore receive better direction. They are made aware of what qualities are considered important and therefore should be checked.

Second, the framework provides a means of quantitatively assessing how well the development is progressing relative to the quality goals established. This augments current techniques used by quality assurance personnel which may range from

only testing to testing and standards enforcement
to testing, standards enforcement, participation in
walkthroughs, and so forth. The advantage pro-
vided by the framework over current techniques
such as code inspections or the use of source code
auditors is the quantification it introduces. The
same techniques can be used (e.g., a code auditor
is a good source of metric data) and translation
to the metric values can be made. Thus the qual-
ity assurance personnel have an additional tool
with which to assess the quality of the software
being produced. In fact, the tool provides
assessment of quality in a different dimension
(i.e., according to the quality factors which
relate to the program manager's view of the
required quality).

Thirdly, the framework provides for more interac-
tion by the quality assurance personnel throughout
the development effort. The metrics have been
established so that subsets are applied during the
three phases of development: requirements analysis,
design, and implementation. The quality assurance
personnel is not only checking a requirements
specification for compliance in format with regu-
latory requirements but is also taking measurements
which can identify poor quality in the require-
ments document which could eventually lead to a
poor end product. This early indication provided
by the framework gives more leverage to the qual-
ity assurance personnel participating in the early
phases of development.

Lastly, indications of poor quality in early phases
may for one reason or another not be acted upon;
that is, corrective actions may be postponed due
to higher priority activities, such as a delivery.
However, they are indicators that potential prob-
lems could exist. Quality assurance personnel
can utilize these indications for identifying new
standards to be enforced in the future, or for
identifying modules to be emphasized during test
testing.

UTILITY OF CONCEPT

In order to assess the impact this framework or
model of software quality will or could have on
software quality assurance, the framework itself
must be evaluated. To evaluate the framework, the
following characteristics must be investigated:

- Definition
  - What is the model measuring?
  - Is it detailed enough?

- Fidelity
  - Will different quality assurance per-
    sonnel get similar results?
  - Are the actuals close to the
    predictions?

- Constructiveness
  - Does it help in understanding soft-
    ware quality?
  - Are the measures derived explainable?

- Stability
  - Can the model be manipulated to obtain
    desired results?

- Usability
  - Can the methodology be cost-effectively
    implemented in a quality assurance
    program?

Each of these characteristics will be discussed in
the following paragraphs.

Definition

The metrics are quantitative measures of the char-
acteristics of the software which provide certain
qualities. The hierarchical structure of the
framework provides relevancy to management at one
level and to software developers at the other
level. The detail is substantiated by the fact
that currently there are approximately 25 charac-
teristics being measured during requirements
analysis, 100 during design, and 150 during
implementation.

Fidelity

The quantification provided by the metrics pro-
vides consistency in its application. Unlike
standards and conventions or inspections, in which
subjectivity is introduced, the metrics utilize
objective quantitative measures. Validations to
date have shown signficant correlations between
the predictions based on measurements and the
actuals based on operational history.

Constructiveness

The fact that the metrics relate to specific
characteristics in the software means the concept
lends itself to understanding software quality.
In most cases, the measures are intuitively asso-
ciated with the related qualities.

Stability

A developer could ensure the software provides
high measures with considerable effort and still
not have a good product. Testing is a safeguard
against this type situation. The difficulty in
subverting the concept is also a deterrent.

Usability

The methodology for applying the framework lends
itself to proceduralization. It requires self-
checking periodic measurement and it lends itself
to automation. While formal relationships which
give statistical credibility to the model have not
been totally validated, the concepts have imme-
diate application to quality assurance activities.

SUMMARY

Thus, the framework described appears to have sig-
nificant potential as a quality assurance tool.
It enforces a life cycle management viewpoint on
quality assurance activities and provides early
indications of quality problems.

The measurement of characteristics of the software and documentation via software quality metrics lends itself to automation. Thus it can be accomplished cost-effectively. Formal relationships between the metrics and their related quality factors have not been validated to date, however, there are indications based on a limited sample that relationships can be established. There are considerable benefits derived using the techniques as they exist currently.

Future resource efforts and experience with these concepts promise to improve its application and expose its potential benefits further.

REFERENCES

(1) Kosy, Donald W., R-1012-PR, "Air Force Command and Control Information Processing in the 1980s: Trends in Software Technology," June 1974.

(2) McCall, J.; Richards, P.; Walters, G., "Factors in Software Quality," three volumes, NTIS AD-A049-014, AD-A049-015, AD-A049-055, November 1977.

(3) Boehm, B., et al, Characteristics of Software Quality, North Holland Publishing Co., NY, 1978.

(4) Bronowski, Jacob, The Ascent of Man, Little, Brown, and Co., Boston, 1973.

(5) Reichenbach, Hans, "Logic and Predictive Knowledge," Space, Time, and the New Mathematics, ed. Robert Marks, Bantam Books, 1964.

(6) Halstead, M., Elements of Software Science, Elsevier Computer Science Library, NY, 1977.

(7) McCabe, T., "A Complexity Measure," 1976 Software Engineering Conference, October 1976.

(8) Myers, G., Reliable Software through Composite Design, Petrocelli/Charter, 1975.

(9) Fagan, M., "Design and Code Inspections and Process Control in the Development of Programs," IBM TR 00.2763, June 1976.

(10) Walters, G. and McCall, J., "The Development of Metrics for Software R&M," 1978 Proceedings of the Annual Reliability and Maintainability Symposium, January 1978.

BIOGRAPHIES

Mailing Address:

  Mr. Joseph P. Cavano
  Information Sciences Division/ISI
  Rome Air Development Center
  Air Force Systems Command
  Griffiss Air Force Base, New York 13441
  (315) 330-4235

Mr. Cavano has been working at the Rome Air Development Center for the past 8 years. His current position deals with developing a methodology for software acquisition based on quality measurements. Prior to this, Mr. Cavano has worked in extending software engineering ideas to an on-line environment. This research was incorporated in his Master's thesis, "On-Line Software Engineering." Mr. Cavano has also designed and implemented a financial management system based on a model of government procurement and combining data management functions with text-processing. Mr. Cavano has received an M.S. in Industrial Engineering and Operations Research from Syracuse University and a B.S. in Mathematics from Clarkson College of Technology.

Mailing Address:

  Mr. James A. McCall
  General Electric Company
  Information Systems Programs
  450 Persian Drive
  Sunnyvale, California 94086
  (408) 734-4980

Mr. McCall has a wide range of experience in operations research, cost-benefit analysis, systems analysis, and simulation. He was principal investigator on the Factors in Software Quality contract with RADC and ESD, and is currently principal investigator on the Metrics Enhancement contract with RADC and USACSC. He has participated in research efforts involving cost estimation, modeling the software development process, and the development of an information and data system simulator and a computer network simulator. Prior to joining GE, Mr. McCall worked in the Advanced Technology Directorate, U.S. Army Computer Systems Command. He participated in a large study analyzing the most cost-effective method of networking the Army's multicommand management information system support. He was also involved in R&D efforts, including program verification and validation, data base management systems, structured programming techniques, and software reliability. He received an M.S. in Operations Research and an M.S. in Engineering-Economic Systems from Stanford University and a B.S. in Engineering from the U.S. Military Academy.