Metrics In Software Quality Assurance

J. E. Gaffney, Jr.
IBM Corporation
Federal Systems Division
Manassas, Va.

Abstract

The nature of "software quality" and some software metrics are defined and their relationship to traditional software indicators such as "maintainability" and "reliability" are suggested. Recent work in the field is summarized and an outlook for software metrics in quality assurance is provided. The material was originally presented as a tutorial at the "ACM SIGMETRICS Workshop/Symposium on Measurement and Evaluation of Software Quality" on March 25, 1981.

Key words and phrases:

software quality assurance
software metrics
software research
software quality factors

Software Quality and the Quality Assurance Organization

What is "software quality"? It is the focus of the software quality assurance organization. "Quality" is an aspect of (software) "product integrity". "Product integrity" includes other things, such as adherence to schedule and cost not covered in the present article which are, nonetheless, of great importance.(1) Most simply, (software) "quality" may be defined as "conformance to requirements". Such "conformance" means, most generally, that the product meets the needs of the user and satisfies stated performance criteria.

The "user needs" that the software product is to satisfy should be provided in written form to the developer before he begins his design. They should be expressed in terms of the functions that the software product is to provide. Such a written expression of "user needs" can be used as the basis for discussion between user and developer in clarifying whether a design appears appropriate as a step in implementing the functions desired by the user.

Software performance criteria can include a wide variety of items, such as there being less than some number of software defects being noted during a sell-off demonstration, fewer than some stated number of defects being found during design and/or code inspections, etc.

Before focusing on the subject of the application of metrics to software quality assurance, let us briefly consider the principal functions of a software quality assurance function. First, it defines the standards for the software products developed in its organizational unit. These standards may include ones established by the Government, by a higher organizational unit such as a corporate or divisional headquarters, or by the particular software quality assurance organization itself.

The second major function of the software quality assurance function is to specify and implement tools or aids for assessing software product quality. The tools may be as simple as checkoff lists or as sophisticated as ones that automatically count the occurrence of such software measureables as the number of unique instruction types in a program, the number of conditional jumps in it, or other such elements that may have a bearing on software quality.

The third major function of the software quality assurance organization is to apply the tools to assess the degree to which the software products developed by its organizational unit adhere to the standards appropriate to that product, which it has established. The assessment may be qualitative, such as certifying the adherence of the software development group to certain development approaches, such as top-down programming and other modern programming practices. The assessment may be quantitative, such as recording the number of major defects (such as a non-terminating loop) found in inspections of the software design and/or the actual code.

Software Metrics and Quality Assurance

A software metric may be defined as "an objective, mathematical measure of software that is sensitive to differences in software characteristics. It provides a quantitative measure of an attribute which the body of software exhibits." It should be "objective" in that it is not a measure of my feelings or yours but is, insofar as possible, a reproducible measure of or about the software product of interest. The number of major defects found during a sell-off test and a comparison of that figure with a pre-established threshold of "goodness" or "badness" is objective. Saying that the software "has a lot of defects" is not. The range of values of software metrics should reflect differences with respect to one or more dimensions of quality among the software products to which it is applied.

Software development is increasingly being accomplished more in line with established engineering and scientific principles and less as an art form. Quantification of the software development process and the resultant software product is mandatory in order for software engineering to truly be a scientific discipline. The use of software metrics will contribute to this desired objective of an increased level of quantification. Without such quantification, the integrity of the software product, which was considered above, cannot be what it should or otherwise has the potential to be.

> What Lord Kelvin said in 1891 applies here:
>
> "When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science."
>
> Or, the message to us in the software community is:
>
> If you can't measure it, you can't manage it.

Software metrics are of interest for several reasons. Numerical measures of the software product can be transformed to indicators, such as "reliability" and "maintainability" of interest to both users and software development management. Some of these measures are defined in the section, "Some Software Metrics". A number of indicators, as defined by McCall[2], are presented in the section, "Quality Factors and Metrics". Also, software metrics are of interest because they might suggest modification to the software development process. For example, the number of conditional jumps used should be minimized because the amount of development testing required is proportional to that figure.

The quantitative evaluation of software quality can address two principal problem types encountered in software products:

A. Those problems having to do with the static aspects of software and which are addressable (at least potentially) by software based/or "program linguistics" oriented metrics. Such metrics are the subject of the remainder of this presentation.

B. Those problems having to do with the dynamic aspects of software, such as that a program is difficult to operate and/or to integrate with other programs. Such problems are not considered further in this presentation.

Quality Factors and Metrics

Software quality focuses on the degree of correctness of an implementation of a function conceived to "meet the user's needs" (see above). It also is concerned with the "goodness" of such an implementation. Ideally, this measure of "goodness" should be quantifiable, indicating how well the software is designed and coded according to measurable, quantifiable criteria. This is where "metrics" fit into software quality assurance. They should relate to software quality "attributes" or "factors" of interest acknowledged by the community of software developers and users.

J. A. McCall[2] has listed some "software qualify factors", some of which can be related to "software metrics", as is done for two of then, "maintainability" and "testability", in the section, "Some Software Metrics". McCall's "software quality factors" (using his definitions) are:

1.	Correctness	Extent to which a program satisfies its specifications and fulfills the user's mission objectives.
2.	Reliability	Extend to which a program can be expected to perform its intended function with required precision.
3.	Efficiency	The amount of computing resources and code required by a program to perform a function.
4.	Integrity	Extent to which access to software or data by unauthorized persons can be controlled.
5.	Usability	Effort required to learn, operate, prepare input, and interpret output of program.
6.	Maintainability	Effort required to locate and fix an error in an operational program.
7.	Testability	Effort required to test a program to insure it performs its intended function.
8.	Flexibility	Effort required to modify an operational program.
9.	Portability	Effort required to transfer a program from one hardware configuration and/or software system environment to another.

10.	Reusability	Extent to which a program can be used in other applications--related to the packaging and scope of the functions that programs perform.
11.	Interoperability	Effort required to couple one system with another.

G. J. Myers[3] has defined some other items which certainly can be considered "software quality factors" in the same sense that the eleven cited above are. They are:

Coupling - The degree of interconnectedness of modules
Strength - The degree of cohensiveness of a module

These measures related to the degree of propagation of changes, and hence, "maintainability". Cruickshank and Gaffney[4] have developed quantitative measures of these items.

Some Software Metrics

In this section, several important metrics are defined and an example of the relationship between then and some of the qualitative "software quality factors" (defined above) is provided. Some of the basic work done by the late Professor Maurice Halstead[5] of Purdue University in software metrics is briefly summarized.

Among the metrics developed by Halstead are these four:

1.	Potential Volume or Intelligence	The minimum amount of "information" an algorithm; function of conceptually unique number of inputs and outputs to a software procedure or module. Its unit is "bits" or "binary digits".
2.	Volume	The actual amount of information a program; a function of the unique number of operators (instructions) and a unique number of operands (data labels) used. Its unit is "bits".
3.	Difficulty (or What Might be Called, "Expansion Ratio")	Volume/Intelligence; the "size" of the program relative to its minimum "size", a measure of redundancy.
4.	Effort	Volume times difficulty; relates to the difficulty a person finds in understanding a program; relates to the degree of difficulty one may find in modifying a program; also relates to error proneness of a program.

Among the metrics developed by others are these three:

5.	Division	Proportion or number of conditional jumps; relates to testing effort; inversely proportional to overall productivity; a measure of control complexity. Various workers, including McCabe[6], Chen[7], Paige[8], and Gaffney[9], have indicated its significance.
6.	Information Flow Complexity	Length of procedure (number of instructions) times the square of the number of possible combination of an input source to an output destination. Kafura et al.[10] have developed this metric.

A metric of particular significance to questions relating to software maintenance is:

7.	Proportion of Modules Changed in an Update	A measure of complexity of the software; relates to difficulty in modifying the software (maintenance). This metric was developed by Belady[11]. It appears to be related to "coupling" and "strength" (see above), perhaps quantifying them to some extent as they deal with questions about propagation of changes.

Various metrics of the group defined above can be related to "software quality factors," such as listed earlier. One can think of an hierarchy in increasing order of detail and quantifiability. An example of such an hierarchy is where the complex concept of "maintainability" is repetitively decomposed until it is described by various metrics, in this case, 'Effort' and 'Division'. Gordon[12] showed a relationship between 'effort' and the 'understandability' of a program, which is a major attribute of 'maintainability'.

In the remainder of this section, mathematical definitions of the four Halstead metrics qualitatively defined earlier in this section are provided.

Consider a program to be composed of a sequence of 'operators' and 'operands'. For example, in the instruction "ADD A", 'ADD' would be an operator and 'A' an operand. Halstead[5] made the following definitions.

n_1 = No. of operator types used.
n_2 = No. of operand types used.
n^* = minimum no. of operand types (= the conceptually unique no. of inputs and outputs)
$n = n_1 + n_2$ = Total 'vocabulary' size.
N_1 = Total no. of operators used.
N_2 = Total no. of operands used.
$N = N_1 + N_2$

Then, he mathematically defined the first of the metrics given above as:

1. Potential volume, $V^*=(2+n_2^*)\log_2(2+n_2^*)$, the minimum program size, a measure of the instrinsic 'size' of the algorithm to be programmed.

2. Volume, $V=N\log_2 n$ A measure of program size.

3. Difficulty (D) or expansion ratio = V/V^*.

4. Effort = V x D = No. of decisions required to develop the program.

Some Recent Metrics Work

A great deal of work is being done with software metrics in various universities and industrial organizations. Some of this work as applied to software quality assurance is summarized below:

1. Fitsos and Smith, IBM, GPD, Santa Teresa - Demonstrated a strong relationship between 'Difficulty' metric and number of defects in code.

2. Kafura, Iowa State[10] - Developed 'information flow' metric, found high correlation with number of defects.

3. Ottenstein, Michigan Technological Univ.[13] - Estimated number of defects in code as function of 'program volume' metric.

4. Bailey and Dingee, Bell Labs., Denver[14] - Applied 'software science' metrics to some switching system software, found defects not to be a linear function of 'program volume' metric.

5. Gaffney, IBM, FSD, Manassas[15] - Applied 'software science' metrics to some signal processing and control software, found potential for 'goodness/badness' relations, found relation between 'volume' and 'potential volume' metrics and number of conditional jumps and hence, testability (per McCabe[6] and Paige[8]).

6. Belady, IBM, Research, Yorktown Hts.[11] - Developing measures of software complexity and progressive deterioration based on propagation of changes.

7. Basili, Univ. of Maryland[16] - Determining relationships among various software metrics with software engineering laboratory data.

Outlook for Software Metrics In Quality Assurance

Metrics are promising and are potentially very valuable for providing a basis for objective comparison of software products and possibly for providing a basis for establishing standards of 'goodness/badness'. They should prove useful in supplementing the 'software defect count' models, such as developed by Musa[18].

Much work is yet to be done for metrics to be extensively applied on a practical basis in the software development and maintenance environments. Areas in which work needs to be done include:

1. Refining the metrics and selecting the most valuable ones; standardizing a set to be used, if possible.

2. Establishing the validity of the metrics used in a particular environment in which they are employed.

3. Establishing 'goodness/badness' thresholds for the metrics used.

4. Applying metrics at the software design stage if possible (a greater payoff is potentially possible due to less expense for earlier modifiability).

5. Building up a base of metrics application experience on a set of programs of application size (not just 'toy programs').

6. Conducting cost-benefit analyses of metrics applications.

7. Obtaining acceptance of the utility of metrics by the software development community.

Some 'social' issues in the practical application of software quality metrics are:

1. There is no accepted measurement practice

2. Analysts and programmers are often relatively autonomous

3. Management has a need for control

4. There is resistance to quantitative measure of one's work product by most software professionals

5. Often, there is organizational inertia and resistance to change in the way in which business is done

6. Who should do the measuring?

7. Who should use the measures?

8. How should the measures be used?

Summary

Software "quality" is an aspect of "product integrity" definition of a "software metric" was given and several "metrics" were defined. The utility of "metrics" including the quantification of certain attributes of software "quality factors", such as "maintainability" was outlined Mathematical definitions of four of the metrics developed by Halstead were provided. Some recent work in the software metrics field related to software quality was summarized. An outlook for the application of software metrics in quality assurance was provided; their use is promising but much work needs to be done if their full potential is to be realized.

Bibliography

1. Bersoff, E., Henderson, V., Siegel, S. "Software Configuration Management, An Investment in Product Integrity," Prentice-Hall, 1980.

2. McCall, J.A., "An Introduction to Software Quality Metrics," In J. D. Cooper and M. J. Fisher (Eds), "Software Quality Management," Petrocelli, 1979.

3. Myers, G. J., "Reliable Software Through Composite Design," Petrocelli/Charter, 1975.

4. Cruickshank, R. and Gaffney, J., "Measuring the Development Process: Software Design Coupling and Strength Metrics;" The Fifth Annual Software Engineering Workshop, November, 1980, NASA Goddard Space Flight Center.

5. Halstead, M., "Elements of Software Science," Elsevier, 1977.

6. McCabe, T., "A Complexity Measure," "IEEE Transactions on Software Engineering," December, 1976, pg. 308.

7. Chen, E., "Program Complexity and Programmer Productivity," "IEEE Transactions on Software Engineering," May, 1978, pg. 187.

8. Paige, M., "An Analytical Approach to Software Testing," Proceedings of the "IEEE Computer Software and Applications Conference," October, 1978, pg. 527.

9. Gaffney, J., "Program Control Complexity and Productivity," Proceedings of the "IEEE Workshop on Quantitative Software Models," October, 1979, pg. 140.

10. Kafura, D., Harris, K., Henry, S., "On the Relationship Among Three Software Metrics," Proceedings of the "1981 ACM Workshop/ Symposium on Measurement and Evaluation of Software Quality," March, 1981 (ACM SIGMETRICS, Volume 10, Number 1; Spring, 1981), pg. 81.

11. Belady, L., "An Anti-Complexity Experiment," IEEE Workshop on Quantitative Software Models, October, 1979, pg. 128.

12. Gordon, R., "A Measure of Mental Effort Related to Program Clarity," Ph.D. Thesis, Purdue University, 1977; University Microfilms International.

13. Ottenstein, L., "Predicting Software Development Errors Using Software Science Parameters," 1981 ACM Workshop, op. cit., pg. 157.

14. Bailey, C., and Dingee, W., "A Software Study Using Halstead Metrics," 1981 ACM Workshop, op. cit., pg. 189.

15. Gaffney, J., "Software Metrics: A key to Improved Software Development Management," "Computer Science and Statistics", 13th Symposium on the Interface" (at Carnegie-Mellon University), March, 1981, to be in proceedings published by Springer-Verlag.

16. Basili, V., and Phillips, T-Y, "Evaluating and Comparing Software Metrics in the Software Engineering Lab," 1981 ACM Workshop, op. cit., pg. 95.

17. Musa, J., "Software Reliability Measurement," "The Journal of Systems and Software I, 1980, pg. 223.