



Software Metrics – A Tool for Measuring Complexity

Dr. Sanjeev Dhawan

Faculty of Computer Science & Engineering
University Institute of Engineering and Technology,
Kurukshetra University, Kurukshetra, Haryana, INDIA.

E-mail: rsdhawan@rediffmail.com

Kiran

Research Scholar

University Institute of Engineering and Technology,
Kurukshetra University, Kurukshetra, Haryana, INDIA.

E-mail: kiran.dhawan54@gmail.com

Abstract: Software measurement is very important and complex issue, especially when dealing applications into the practice. This paper discusses the comparison of various procedural oriented metrics which are used for the measurement of several properties of source code such as size, complexity, and error. It is widely accepted that sizing or predicting the volume of software process is one of the most dominant aspects of cost estimating. Generally cost estimation model are based upon the strong assumption that size and complexity influences development effort and still there is no unique metric which can work during all of the development phases. In order to improve the software quality and the maintainability, it is necessary to control the software complexity by measuring the related aspects.

Keywords: Software Measurement, Software Metrics, Size metrics, Complexity Metrics.

I. Introduction

Software engineering differs from other engineering disciplines in a number of aspects that have important consequences on software measurement. First, software engineering is a young discipline, so its theories, methods, models and techniques still need to be fully developed and assessed. However, the very nature of software engineering makes measurement a necessity. Software Complexity is defined as the degree to which a system or component has design and implementation that is difficult to understand [10], i.e. complexity of a code is directly depend upon understandability. Software complexity is an estimate of effort that is required to develop, maintain and execute the code [11]. Higher effort and time is required when the code is complex. Cyclomatic and Halstead are two common metrics to measure the complexity of code. Cyclomatic Complexity based on control flow path and Halstead metric based on mathematical relationship between number of variables like operators and operands.

II. Literature Review

The area of software measurement or software engineering measurement is one of the areas in software engineering where researchers are active since more than thirty years. Several approaches for predicting the software size have been proposed since 1970. Complexity and size are strongly related to the efforts. Basili defines complexity as a measure of the resources expended by a system while interacting with a piece of software to perform a given task. If interacting system is computer, then complexity is defined by execution time and storage required [8]. McCall [7] identified factors or characteristics of which reliability was one, which comprise the quality of computer software and then discussed their impact over the life of a software product. This paper emphasized those factors of software quality which have greatest importance at the later stages of a software product's life. The first software quality factors that received attention were correctness, efficiency, and reliability, probably in that order. Martin R. *et al.* [14] in their research paper discussed the need for measures of complexity and unstructuredness of programs. The proposed metric was compared with other metrics, the most notable of which was the Cyclomatic complexity measure. However, they also felt that the knot count provides a much clearer indication of program readability.

The further research in this area was also performed by Gill and Kemerer [16]. In their paper they presented the need for Software metrics to aid in the assessment of software complexity for maintenance. McCabe's Cyclomatic complexity metric, a measure of the maximum number of linearly independent circuits in a program control graph, has been widely used in research. Another similar finding was reported by Weyuker [15] described a set of properties of software complexity measures. Four complexity measures were evaluated and compared. Author also checked whether such well-known complexity measures as McCabe's Cyclomatic number, Halstead's programming effort, statement count, and data flow complexity satisfied their properties.

These three complexity measures were further described by Sylvia *et al.* [10]. In their performed research they compared these three complexity measures to programmer performance. Halstead's software science has provided some important directions for investigating psychological complexity. Subsequent work by McCabe also proved relevant to this purpose. Assessing the psychological complexity of software appeared to require more than a simple count of operators, operands, and basic control paths.

III. Software Measurement

Measurement is introduced by information technology organizations to better understand, evaluate, control and predict software processes. Software measurement is essential to achieving the basic management objectives of prediction, progress, and process improvement [9]. Measurement as the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules [6]. Measurement can be categorized in two ways: - (a) Direct measures: It includes software process (e.g., cost and effort applied) and product (e.g., lines of code (LOC) produced, execution speed, and defects rate), and (b) Indirect measures: It includes the product functionality, complexity, efficiency, reliability, maintainability and many others [1]. Measurement is used to determine and predict the quality of the current product or process. It is also used to improve the quality of product/process.

IV. Software Metrics

Metrics used for evaluating the software processes, products and services is termed as Software metrics. Appropriately selected metrics can help both management and engineers to maintain their focus on their goals. Software metrics programs must be designed to provide the specific information necessary to manage software projects and improve software engineering processes and services. Software metrics are techniques/formulas to measure some specific property or characteristics of software. In software engineering, the term 'Software metrics' is directly related to the measurement. Software metrics is a measurement based technique which is applied to processes, products and services to supply engineering and management information and working on the information supplied to improve processes, products and services, if required [2].

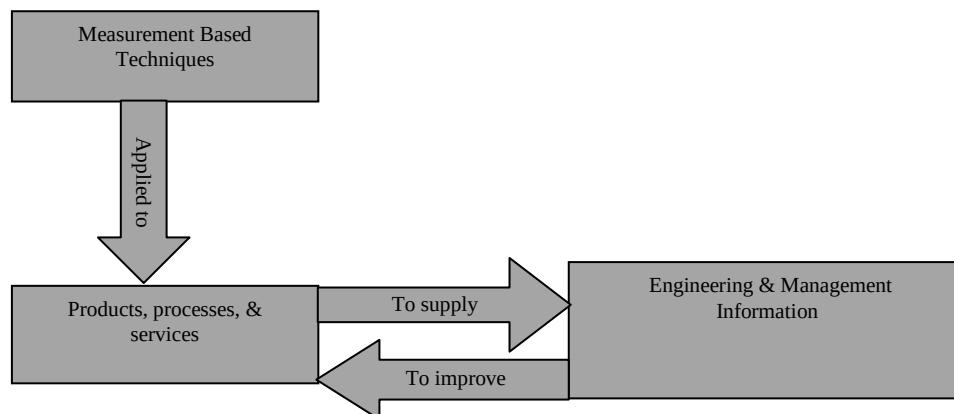


Figure 1 Software metrics.

V. Size Metrics

The most important product metric is the size metric. A number of different software size metrics have been proposed and used. Size metric is direct count of selected characteristics of software product. It is used to describe the volume of a software product [13]. Size metrics can be expressed in terms of physical size of the program, function and services it provides, and in terms of logical size. The following metrics measures the size of the code.

(a) Lines of code (or LOC): It is possibly the most widely used metric for program size. This metric is used to measure the quantitative characteristics of program source code. This metric is based on counting the lines of the source code [6]. The most common definition of LOC seems to count any line that is not a blank or comment line, regardless of the number of statements per line. Levitin [18] concluded that LOC is a poorer measure of size than Halstead's program length, N .

(b) Function Point (FP): This metric was developed by Albrecht [Albrecht 79] [3]. He has proposed a measure of software size that can be determined early in the development process. Function points are intended to be a measure of program size and, thus, effort required for development. The approach is to compute the total function points (FP) value for the project, depends on the counts of distinct (in terms of format or processing logic) types in the following five classes input, output, external interface files, internal files, and external inquiries[6].

$$FP = \text{count total} * [0.65 + 0.01 * (Fi)]$$

Where count total is the sum of all FP entries. The Fi ($i = 1$ to 14) are "complexity adjustment values".

VI. Complexity Metrics

Complexity is correlated to the size of code. Complex program need more time to develop and test. This includes both the internal complexity of a single procedure and the complexity of the data flow in and out of a procedure [9]. High complexity may result in bad understandability and more errors. Therefore, excessive complexity should be avoided. Too complex procedures should be simplified by rewriting or splitting into several procedures. The following metrics measure the complexity of executable code within procedures.

(a) Cyclomatic Complexity: Thomas McCabe introduced a metric in 1976 based on the control flow structure of a program [3]. This metric is known as McCabe Cyclomatic complexity (CC) and it has been famous code complexity metric throughout since it was first introduced. Measuring CC of a code is like making a basis path testing. In other words, it has an advantage of measuring the flow of a program. Thus, it can measure the complexity of an algorithm. Given any computer program, McCabe method maps a program to a directed, connected graph. The nodes of the graph represent decision or control statements. The edges indicate control paths that define the program flow [4].

Cyclomatic complexity is calculated as:

$$v(G) = e - n + 2$$

Where, e is the number of edges, and n is the number of nodes in the graph. Cyclomatic Complexity can also be calculated by just counting the number of predicates (binary nodes), and add one to this.

$$v(G) = p + 1$$

Where, $v(G)$ is the Cyclomatic Complexity, and p is the number of binary nodes or predicates.

(b) Halstead Metrics: This metrics was introduced by Halstead in 1977 and this metrics is known as Halstead software science or as Halstead metrics. It is the best known measures of software complexity. Most of the product metrics typically apply to only one particular aspect of a software product [Halstead77]. In contrast, Halstead set of metrics applies to several aspects of a program, as well as to overall production effort [5]. Halstead uses some primitive measures to develop expressions for program vocabulary (n), length (N), and volume (V) metrics.

$$\text{Program Vocabulary } (n) = n_1 + n_2$$

$$\text{Program Length } (N) = N_1 + N_2$$

$$\text{Program volume } (V) = N \log_2 n$$

$$\text{Program level } (PL) = 1 / [(n_1/2) * (N_2/n_2)]$$

$$\text{Effort } (e) = V/PL$$

VII. Observation

The table below presents advantages and disadvantages of various software metrics discussed in above section.

Metrics	Advantages	Disadvantages
LOC	- Size Oriented Metric - Language Dependent - Easy to Compute	- LOC has no counting standards - Not useful to compare hand written and auto-generated code - Differs from person to person
FP	- Size Oriented Metric - Language Dependent - Understood by the non-technical user. - to estimate cost and resources required for software development	- Manually Counting Process - Difficult to Understand - Requires Experience
CC	- Gives complexity of various designs. - Computed early in life cycle. - Measures minimum effort. - Easy to apply.	- Measures program's control complexity and not the data complexity. - Nested conditional structures are harder to understand. - Ignores the size of program.
Halstead Metrics	- Predicts error rate. - Predicts maintenance effort. - Simple to calculate. - Measure overall quality. - Used for any language	- Depend on complete code. - Complexity increases as program level decreases. - Difficult to compute

Table 1 Comparison of different metrics.

VIII. Conclusions

Even though there are number of software metrics so far, the main focus of all these metrics on how to improve the process of developing, maintaining and managing software. From the above discussion, a strong point is raised that metrics should be used in the early phase of development life cycle so that it help to detect and correct various requirement bugs and also prevent error that may encounter in later phases of development. Metrics can identify potential problems that may lead to identify errors in the system and decreases over all development cost. The complexity of software will directly affect the maintainability, and reliability of the software. This paper shows the comparison between different size and complexity metrics. Between LOC and Function Point metrics, Function Point metric is widely used to measure the size of code because it is language dependent and easily understood by non technical users. And the Complexity metrics like Cyclomatic Complexity and Halstead metric which are used to predict the complexity of program. Cyclomatic Complexity is best known metric to compute complexity because it is easy to compute, and require minimum effort than Halstead metric.

IX. References

- [1] Gurdev Singh. *et al.*, "A Study of Software metrics", IJCEM International Journal of Computational Engineering & Management, Vol. 11, January 2011.
- [2] Linda L. Westfall Principle Software Measurement Services Plano, TX 75075, "Seven Steps to Designing a Software Metric.
- [3] Albrecht, A. J. and J. E. Gaffney. Jr. "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", IEEE Trans. Software Eng. SE-9, 6, pp. 639-648, Nov. 1983.
- [4] T. J. McCabe, "A Complexity Measure," ICSE '76: Proceedings of the 2nd international conference on Software engineering, 1976.
- [5] Everaldo E. Mills, "Software metrics", Software Engineering Institute, 1988.
- [6] R. S. Pressman, "Software Engineering, A Practitioner's Approach", Sixth Edition, Mc Graw. Hill, 2005.
- [7] Gene F. Walters, James A. McCall, "Software Quality Metrics for Life-Cycle Cost Reduction", IEEE Transactions on Reliability, vol. r-28, no. 3, August 1979.
- [8] Basili, V.R. Qualitative software complexity models: A summary, In Tutorial on Models and Methods for Software management and Engineering. IEEE Computer Society press, Los Alamitos, Calif., 1980
- [9] Lou Marco, "Measuring software complexity", Enterprise Systems Journal, 1997.
- [10] Bill Curtis. *et al.*, "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics", IEEE Transactions On Software Engineering, vol. se-5, no. 2, March 1979.
- [11] IEEE Std. 1061-1998 IEEE Computer Society: Standard for Software Quality Metrics Methodology, 1998.
- [12] Li, W. and Henry, S.: "Object Oriented Metrics that predict Maintainability", Journal of Systems and Software, Vol. 23, No. 2, 1993, pp 111-122.
- [13] June Verner and Graham Tate, "A Software Size Model", IEEE Transactions on Software Engineering, vol. 18, no. 4, April 1992.
- [14] Martin R. *et al.* "A Measure of Control Flow Complexity in Program Text", IEEE Transactions on Software Engineering, vol. se-5, no. 1, January 1979.
- [15] Elaine J. Weyuker, "Evaluating Software Complexity Measures" IEEE Transactions on Software Engineering, vol. 14, no. 9, September 1988.
- [16] Geoffrey K. Gill and Chris F. Kemerer, "Cyclomatic Complexity Density and Software Maintenance Productivity", IEEE Transactions on Software Engineering, vol. 17, no. 12, December 1991.
- [17] Albrecht, A.J., "Measuring Application Development Productivity," Proc. IBM Application Development Symposium, Monterey, CA, October 1979, pp. 83-92.
- [18] Levitin, A. V. "How to Measure Software Size, and How Not To." Proc. COMPSAC 86. Washington, D. C.: IEEE Computer Society Press, Oct. 1986, 314-318.
- [19] Halstead, M. H. Elements of Software Science. New York: Elsevier North-Holland, 1977.