International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:8, 2009

Static and Dynamic Complexity Analysis of Software Metrics

Kamaljit Kaur, Kirti Minhas, Neha Mehan, and Namita Kakkar

***Abstract*—**Software complexity metrics are used to predict critical information about reliability and maintainability of software systems. Object oriented software development requires a different approach to software complexity metrics. Object Oriented Software Metrics can be broadly classified into static and dynamic metrics. Static Metrics give information at the code level whereas dynamic metrics provide information on the actual runtime. In this paper we will discuss the various complexity metrics, and the comparison between static and dynamic complexity.

***Keywords*—**Static Complexity, Dynamic Complexity, Halstead Metric, Mc Cabe's Metric.

I. Introduction

SOFTWARE Complexity is defined as "the degree to which a system or component has a design or implementation that is difficult to understand and verify"[8] i.e. complexity of a code is directly dependent on the understandability. All the factors that makes program difficult to understand are responsible for complexity.

Software complexity is an estimate of the amount of effort needed to develop, understand or maintain the code. It follows that more complex the code is the higher the effort and time needed to develop or maintain this code [9]. Results based on real life projects show that there is a correlation between the complexity of the system and the number of faults [10] [11]. The McCabe metric and Halstead's are two common code complexity measures. The McCabe metric determines code complexity based on the number of control paths created by the code. Halstead bases his approach on the mathematical relationships among the number of variables, the complexity of the code and the type of programming language statements.

II. Literature Review

Six design metrics are based on measurement theory. In evaluating these metrics against a set of standard criteria, they are found to possess both a number of desirable properties and suggest some ways in which the Object Oriented approach may differ in terms of desirable or necessary design features from more traditional approaches [1].

Kamaljit Kaur is working with Department of Information technology at Shaheed Udham Singh College of Engineering and Technology, Mohali, India.

Kirti Minhas, Neha Mehan and Namita Kakkar are associated with Department of Computer Science & Engineering of Rayat-Bahra Institute of Engineering & Bio-Technology, Sahauran, Distt. Mohali, India.

An adequate complexity metrics set for large-scale OO software systems is still a challenge. In traditional OO software engineering methodologies, OO metrics such as CK and MOOD have been recognized in practical software development. In order to measure a system's complexity at different levels, they propose a hierarchical complexity metrics set that integrates with these metrics and parameters of complex networks [2].

Yacoub et. al. [3] defined two metrics for object coupling (Import Object Coupling and Export Object Coupling) and operational complexity based on state charts as dynamic complexity metrics. The metrics are applied to a case study and measurements are used to compare static and dynamic metrics.

Munson and Khoshgoftaar [4] showed that relative complexity gives feedback on the same complexity domains that many other metrics do. Thus, developers can save time by choosing one metric to do the work of many.

Munson and Hall [5] examined the static complexity of a program together with the three dynamic measures of functional, fractional, and operational complexity. The eminent value of the dynamic metrics was shown in their role as measures of test outcomes.

Mayo et. al. [6] explained the automated software quality measures: Interface and Dynamic Metrics. Interface metrics measure the complexity of communicating modules, whereas Dynamic metrics measure the software quality as it is executed.

Hays in [7] has examined the testing of object-oriented systems. Then compare and contrast it with the testing of conventional programming language systems, with emphasis on fault-based testing.

III. Halstead's Metric

Halstead has proposed metrics for length and volume of a program based on the number of operators and operands. In a program he defined the following measurable quantities:

- *n1*= the number of distinct operators
- *n2* = the number of distinct operands
- *N1* = the total number of operators
- *N2* = the total number of operands

From them, he defined the following entities:-

Vocabulary (n) = n1 + n2

Length (N) as $N = N1 + N2$

Volume (V) as $V = N \log_2 n$ (the program's physical size)

Potential Volume(V)* as $V^* = (2 + n2^*) \log_2 (2 + n2^*)$ (the smallest possible implementation of an algorithm).

Program level (L) as $L = V^* / V$ (The closer L is to 1, the tighter the implementation.) Starting with the assumption that code complexity increases as vocabulary and length increase, Halstead observed the following:

1. Code complexity increases as volume increases.
2. Code complexity increases as program level decreases.

IV. Problems with Halstead's Metric

Halstead's work is criticized on several fronts:

1. Fault with his methodology for deriving some mathematical relationships and fault with some of his assumptions. For example, Halstead used a number called the Stroud number (ranging from five to 20), which represents how many elementary discriminations the human brain can perform in a "moment." Another "fact" is that Halstead postulated the human brain can handle up to five chunks of material simultaneously.
2. Halstead metrics are difficult to compute. How do we easily count the distinct and total operators and operands in a program? Counting these quantities every time we made a significant change to a program is difficult. A metric is useless if you can't compute it quickly and easily.
3. The computation of the Halstead metrics for the bubble sort suggests that the bubble sort, as implemented, is very complex. The problem is that the computation for the potential volume mandates the number of input and output parameters. For the bubble sort, only the array to be sorted is needed. The low number for the potential volume skews the program and language levels.

V. Mc Cabe's Metric

Thomas J. McCabe developed Cyclomatic complexity, is a software metric and is used to measure the complexity of a program. His fundamental assumption was that conditions and control statements add complexity to a program. Given two programs with same size, the program with the larger number of decision statements is likely to be more complex. It directly measures the number of linearly independent paths through a program's source code. Cyclomatic complexity is computed using the control flow graph of the program as show in figure 1. McCabe's Cyclomatic complexity, best known measure is based on the control structure. The metric can be defined in two equivalent ways:

a) The number of decision statements in a program + 1

b) For a graph *G* with *n* vertices, *e* edges, and *p* connected components,

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

A code fragment and its corresponding flow graph are shown below:

```
if (x < 0)
      do {
            if (y)
                  b();
            m = c() * m;
      }
      while (m < k);
else if (x == 0)
      do{
            if (y ==0)
                  b();
            c();
      }
      while (x == 0)
else
      do {
            if (j)
                  b();
            m = c() * 2m;
      }
      while (m <= k);
```

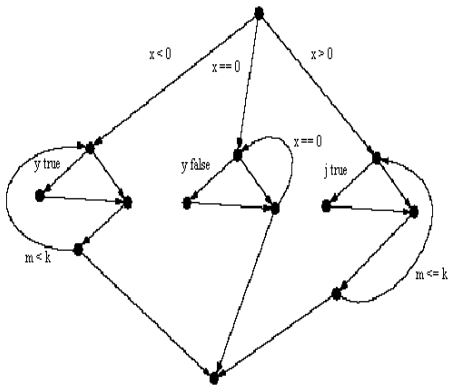

Fig. 1 Flowgraph of the Example Program

The following are the merits of the Mc Cabe's metric:

1. The complexity measure is simple
2. There is no doubt that a large class of programming errors occurs around conditions and loops and adds to complexity.

VI. Comparison of Static and Dynamic Complexity

Static metrics measure what may happen when a program is executed. A dynamic measure would provide a means of measuring what is actually happening. Static Metrics are derived from an analysis of non-executing code. Dynamic metrics are derived from an analysis of code while it is executing. They provide an indication of what calls are actually taking place, the number of statements executed and what paths are being executed. Dynamic metrics include both complexity measures and measures useful in reliability modeling. Dynamic metric values are dependent on the input or test data with which system software is run.

VII. Conclusion

We have to take into the consideration both static as well as Dynamic Complexity Metrics, so as to find out the variation. After comparing Static and Dynamic Complexity Metrics we

can come to the conclusion that whether the Software is good in quality or not from coding as well as execution point of view. If complexity is higher, then we have to change the code. Since each user have got different requirement, we can also suggest according what changes to make it more understandable and simple. Hence, this will lead to a good software.

REFERENCES

[1] Shyam R. Chidamber and Chris F. Kemerer," A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, Vol. 20, NO. 6, JUNE 1994.
[2] Yutao Ma, Keqing He, Dehui Du, Jing Liu, and Yulan Yan , "A Complexity Metrics Set for Large-scale Object-oriented Software Systems", IEEE International Conference on Computer and Information Technology (CIT'06).
[3] Sherif Yacoub, Tom Robinson, and Hany H. Ammar , "Dynamic Metrics for Object Oriented Designs", Software Metrics Symposium, 1999. Proceedings. Sixth International Volume , Issue , 1999 Page(s):50 – 61
[4] John C. Munson, Taghi M. Khoshgoftaar, "Measuring Dynamic Program Complexity," IEEE Software, vol. 9, no. 6, pp. 48-55, Nov./Dec. 1992, doi:10.1109/52.168858
[5] John C. Munson, Gregory A. Hall, "Estimating test effectiveness with dynamic complexity measurement", Empirical Software Engineering Journal, ISSN- 1382-3256.
[6] Kevin A Mayo, Steven A Wake, Sallie M. Henry," Static and Dynamic Software Quality Metric Tools", Department of computer Science, Virginia Tech, Blacksburg.
[7] Jane Huffman Hayes, "Testing of Object-Oriented Programming Systems (OOPS): A Fault-Based-Approach", Science Applications International Corporation, 1213 Jefferson-Davis Highway, Suite 1300, 22202 Arlington, Virginia.
[8] IEEE Std. 1061-1998 IEEE Computer Society: Standard for Software Quality Metrics Methodology, 1998.
[9] Li, W.; Henry, S.: "Object Oriented Metrics that predict Maintainability", Journal of Systems and Software, Vol. 23, No. 2, 1993, pp 111-122
[10] Ammar, H. H., Nikzadeh, T., Dugan, J., "A Methodology for Risk Assessment of Functional Specification of Software Systems Using Colored Petri Nets", Proc. Of the Fourth International Software Metrics Symposium, Metrics'97, Albuquerque, New Mexico, Nov 5-7, 1997, pp108-117.
[11] Munson, J., Khoshgoftaar, T., "Software Metrics for Reliability Assessment", in Handbook of Software Reliability Engineering, Michael Lyu (edt.), McGraw-Hill, 1996, Chapter 12, pp 493-529.