Analysis and Comparison of Software Complexity Model

Hai-Yan Chen

Abstract. This paper analyses four common models of current software complexity as LOC, McCabe Cyclomatic, Halstead Volume, and Information Flow Complexity. It also compares those four models with experiments, summarises strengths and weaknesses of each model, and lays the foundation of setting up some new complexity models.

Keywords: LOC (lines of code), McCabe Cyclomatic, Halstead Volume, Information Flow Complexity.

Introduction

As early as 1975, the Father of IBM mainframe computers, Fred Brooks, published a book, The Mythical Man-Month [1]. In the book, he recollected gains and experience of developing broad scale software system which he led more than one thousand people in 1960s. He gathered that in the process of developing broad scale software system, it would be rather difficult for the designers to cover all designing ideas and then convey a consistent designing idea. Therefore, there would be high complexity of the software, usual lags of the broad scale software system, soaring cost and countless mistakes. That is called software crisis [2].

Notwithstanding people commonly realise the complexity is the basic feature of the software, nevertheless there have been very limited related literature towards the complexity of the software. What is more, yet, there has been no any generally acknowledged adequate definition of it. Based on
current literature, most researchers hold this belief: “Complexity of software is the level of difficulty to analyse, design, test, maintain and change the software”, and the complexity will increase with time running.

Famous measures of software include: McCabe Structure Complexity Metric [3], Halstead Software Science [4], LOC Statement Line Measure [5], and Metric based on FPA Function Point Analysis. Those measures and metrics have been refined to more detailed ones. To summarise those measures of software complexity which have been put forward, they are categorised to four sorts:

1) Measure of Structure Complexity: this researches the topological structure, control path and the influence of control path of all parts in the system.

2) Measure of Algorithm Complexity: this takes relative difficulty to achieve the algorithm into account, which contains complexity of time and space. Nonetheless this complexity is not a direct property of the software, but an indirect one of the algorithm.

3) Measure of Data Structure Complexity: This focuses data structure and the influence of data flow to the software.

4) Measure of Text Complexity: This gives the source programme a static analysis, and throughout counting the operational character and operand, reckons on the potential mistake number and development effort in the software.

**Model of Complexity Parameter**

We choose following four algorithms as a complexity measure, for they are entirely independent from the code format and basically, have nothing to do with the used language. Those four are: LOC(lines of code), McCabe Cyclomatic, Halstead Volume, Information Flow Complexity. Here to briefly introduce those four algorithms.

- **LOC**
  
  LOC refers to the scale of the software, the number of the executable source code lines, which include deliverable working control language statements, data definitions, data type declarations, equivalent declarations, and input / output format declarations. This is a method which reckons from technics and the line of code (LOC) is used as an estimation unit for software complexity. The advantage is easy to count and the disadvantage is the ambiguity of the number of lines of code and the difficulty of a piece of work and the efficiency of the code cannot be accurately reflected

- **McCabe Cyclomatic**
  
  Cyclomatic complexity, also known as loop complexity, was proposed by McCabe in 1976 for structural testing of software, based on the following
assumptions: the complexity of the programme depends largely on the complexity of the programme control flow. That is to say, the simplest single order structure, the more loops and loops that are formed, the more complex the programme.

Its method of computation is that the node is the smallest unit of code in the programme, and the edge represents the programme flow between the nodes. If a module flow chart had e sides and n nodes, then its loop complexity would be as:
\[ V(G) = e - n + 2p. \]

\( V(G) \) - the number of turns in the graph G;
\( E \) - edge number;
\( N \) - node number;
\( P \) -- the number of disconnected parts of the graph.

Halstead Volume

Halstead (1997) distinguished software science from computer science. The premise of software science is that any programming task consists of the selection and arrangement of a finite number of programme symbols (token), which are the basic grammatical units that can be distinguished by the compiler.

Its basic idea is to calculate the complexity of the programme according to the number of operational code of executable code lines and operands in the programme. The more of operational code of executable code lines and operands, the more complex the programme structure. Halstead measured the complexity of a programme based on the total number of operators and operands in a programme.

Typically, Halstead defined the length as:
\[ H_v = (N_1 + N_2) \log_2(n_1 + n_2) \]

Amongst them,
\( N_1 \) = the number of different action-code (operators) that appear in the programme, as: =, +, etc.
\( N_2 \) = the number of action-code (operands) that appear in the programme, such as constants, variables, etc.
\( N_1 \) = the total number of occurrences of the operator, namely, occurrence number.
\( N_2 \) = the total number of occurrences of operands, namely, occurrence number.

It is advantageous in that it does not require deep analysis of the structure of the programme, and it can predict the error rate and the maintenance workload. It is also advantageous to the project planning and measuring the
complexity of all the procedures. The calculation method is simple and has
nothing to do with the advanced programming language type which may be
used. Numerous research structures have shown that Halstead complexity is
useful for predicting programme working plans and Bug detection.

- **Information Flow Complexity**

  The complexity of the information flow was proposed by Henry and
Kafura [6] in 1981. The fan-in refers to summary of the number of parameters
sent to the function, and the number of global variables in the module and the
read operation. Whilst fan-out refers to the number of parameters returned
from the function and the number of write-operations towards the global
variables in the function.

\[ N_x = \sum [(\text{fan-in}) \times (\text{fan-out})]^2 \]

Generally speaking, there is at least one input and output variable for any
programme. Therefore, \( \text{fan-in} \neq 0 \) and \( \text{fan-out} \neq 0 \), namely, \( N_x \neq 0 \).

**Experiments**

Suppose there was a banking operating system consisting of 10 functions
(States). The programme architecture is shown in the figure below. Suppose
that the reliability of each function (state) \( R_i \) was known as follows:

\[ R_1=0.998 \quad R_2=0.995 \quad R_3=0.996 \quad R_4=0.990 \quad R_5=0.999 \]

\[ R_6=0.975 \quad R_7=0.980 \quad R_8=0.985 \quad R_9=0.990 \quad R_{10}=1 \]

Table below lists the related parameters.

| Components name | \( S_1 \) | \( S_2 \) | \( S_3 \) | \( S_4 \) | \( S_5 \) | \( S_6 \) | \( S_7 \) | \( S_8 \) | \( S_9 \) |
|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| \( N/\ln N \)  | 83/4.416 | 2115/7.666 | 984/5.949 | 407/5.989 | 91/4.531 | 94/4.542 | 575/6.354 | 356/5.896 | 84/4.429 |
| \( Hv/\ln Hv \) | 213/5.549 | 5315/7.666 | 1615/0.85 | 368/8.219 | 1147/7.045 | 921/6.834 | 4910/8.597 | 3729/7.459 | 214/5.288 |
| \( Mu/\ln Mu \) | 4/1.486 | 31/0.089 | 20/6.94 | 2/0.683 | 2/0.695 | 2/0.683 | 2/0.683 | 2/0.683 | 2/0.683 |
| \( Nx/\ln Nx \) | 16/2.775 | 2304/7.743 | 169/5.120 | 16/2.792 | 400/5.891 | 196/5.288 | 4/1.387 | 9/4.513 | 196/5.288 |

**Acknowledgements**

The corresponding author of this paper is CHEN Hai-Yan. This paper is
supported by Projects of the National Social Science Fund (No.06BFX051);
Shanghai university training and selection of outstanding young teachers in
special research fund (No.hzf05046).
Summary

The measurement of the programme size by the Halstead metric reflects the number of operators and opcode used in the programme. The measurement of the McCabe metric represents the number of loops and branches in the programme. The measurement of information flow reflects the amount of information exchanged between modules in a programme. They are important aspects that affect the complexity of programmes. In this way, more operator code will make some impact on the programmer's thinking process more and bring more defects to the programme. Similarly, if the number of modules and branches in the programme are that large, the amount of information exchange that large, and the programme is very complicated, there will be a lot of trouble to the software testers, and then make the residual and new defects of the programme increase.

References

[1] Surhone L M, Tennoe M T, Henssonow S F, et al. Software Britteness[M]. 2010.
[2] Krishnamurthi S, Vitek J. The real software crisis[J]. Communications of the Acm, 2015, 58(3):34-36.
[3] McCabe, Mark J. Journal Pricing and Mergers: A Portfolio Approach[J]. American Economic Review, 2002, 92(1):259-269.
[4] Jones, David R, Bultsma, et al. Nuclear PtdIns5P as a Transducer of Stress Signaling: An In Vivo Role for PIP4Kbeta[J]. Molecular Cell, 2006, 23(5):685.
[5] Jones C. A Short History Of The Lines Of Code (LOC) Metric[J]. 2008.
[6] Henry, S.M. and D. Kafura,” Software Structure Metrics Based on Information Flow,” IEEE Trasactions on Software Engineering, Vol.SE-7, 1981. pp.510 - 518.