A COMPARATIVE STUDY OF SOFTWARE COMPLEXITY METRIC

ANKITA, VINOD K. BHALLA

Abstract: With the advancement of software development the demand for software is growing day by day and to meet these requirements the complexity of the software goes on increasing. So, in this paper three main metrics are proposed. Three software complexity metrics Line of code, Halstead's Measures of Complexity and Cyclomatic Complexity metrics are used but Cyclomatic Complexity metric is the best and strong metric among these three. First two metrics line of code and Halstead's Measures of Complexity ignores the complexity from the control graph and CC calculates the complexity via decision structure graph and that's why also called conditional complexity.

Keywords: Software complexity, Decision statements, Metric, Data flow,Control graph.

Introduction: The prime focus of software engineers and researchers is to get quality software. For that purpose it is very essential to measure the software complexity and high complexity software difficult to understand, read and hence, troublesome to change in future. Complex software considered to be the reason for the presence of defects, this leads to consider that software complexity is responsible for poor software quality.

Mainly three software complexity metrics LOC, Halstead's measure of complexity and cyclomatic complexity metrics are used but there are some drawbacks of LOC and Halstead's measure complexity metric and to overcome these problems strongest metric cyclomatic complexity metric was introduced among them and three methods are used to measure the cyclomatic complexity. Mainly cyclomatic complexity is calculated from the control flow graph which consists of number of edges and nodes and there is a significance of this calculated cyclomatic complexity number. This number should be in between 1-10 and should not be more than 20. Cyclomatic complexity number greater than 10 signifies that software has high complexity and there will be a chance of more errors in that software. One popular eclipse plug-in named "Metric" is used to calculate the Cyclomatic Complexity. This "Metric" Plug-in provides a number of features like number of classes, number of methods, number of overridden methods, Depth of inheritance tree, total lines of code, number of interfaces, number of packages, specialization index and many other features along with McCabe Cyclomatic Complexity.

Software Complexity Metrics:

A.LOC: LOC is one of the oldest metric and is used to measure the software complexity by counting the number of lines from the source code or via physical length. It is used to measure the productivity or man effort to develop a program. But there are various drawbacks of this metric, it is calculated at the end of the application completion, ignores the complexity from decision statements and consider the complexity of each code line same. Physical size that is lines of code (LOC) metric is not considered adequate because if there are 40 or 50 lines of code consisting of 25 consecutive "If then" constructs may have million distinct control paths, only a small percentage of which would probably ever be tested[1].

B.Halstead's Measures of Complexity:

It basically used to measure complexity by counting the number of unique operators (op1), operands (op2) and total number of operators (OP1) and operands (OP2) in a program. This metric is use to measure error rate, length, difficulty level etc. for a software development [2].

$$ESL=op1\log_2(op1) + op2\log_2(op2) \quad (1)$$
$$SL= OP1+ OP2 \quad (2)$$
$$SV= op1 + op2 \quad (3)$$
$$VOL=SL * \log_2(n) \quad (4)$$
$$V^*= (op1OP2 / 2op2)(OP1+OP2)\log_2(op1+op2) \quad (5)$$
$$LVL=V^*/VOL= (2/ op1)*(op2/ OP2) \quad (6)$$
$$DL=VOL/V^*= (op1/ 2) * (OP2/op2) \quad (7)$$
$$PE=VOL * DL \quad (8)$$
$$EE=VOL/S^* \quad (9)$$
$$PT= PE/18 \quad (10)$$

Where,

ESL: Expected software length
SL: Software length
SV: Software vocabulary
VOL: Volume
LVL: Level
DL: Difficulty Level
PE: Programming Effort
EE: Error Estimate
PT: Programming Time
V*: software ideal volume.

S* is the programmer ability's and Halstead's set this value to be 3000.

There are also some problems with Halstead's Measures of Complexity, it also ignores the complexity from the decision statements like if, loops etc. but used to calculate the complexity from the data flow of a software and it is very difficult to count the number of operators and operands from the program[3].

So, to overcome these problems another metric cyclomatic complexity was introduced.

C. Cyclomatic Complexity metric:

This metric was developed by Thomas J. McCabe, Sr. in 1976. McCabe's cyclomatic complexity is also used to measure the structural complexity of a module. It measures the complexity by counting the number of decision statements from the program and that's why also called conditional complexity [4] and also used to measure the number of independent paths through the graph.

It is considered that more complex software has more number of errors and after that more effort will be required to correct these errors and then it becomes difficult to change the software in the future [5].

McCabe's cyclomatic complexity is a software quality metric and higher the cyclomatic complexity number, the more complex the code will be.

If the cyclomatic complexity number of a module is in the range of 1 to 10 then it is considered as risk free module. If the same lies in the range of 10-20 then it is considered as a target of moderate risk. 30-40 range of cyclomatic number makes module highly risky and the range exceeding 40 exempt it from the candidate of testing. [6].

This metric has a strong correlation with LOC [7] and also along with Halstead's measure of complexity.

Instead there is no relation between control path and number of operators and operands but still with the increase of control path, number of operators and operands also get increased. So, this shows a correlation between Halstead's and Cyclomatic complexity metric.

Cyclomatic complexity metric does not consider the complexity from the data flow of software. Example if there are 1000 lines of code in any software and there is no conditional statements in the code then cyclomatic complexity metric calculates the complexity of that software as one. One more problem with this metric is that it considers the complexity of two statements having while and if as same [8].

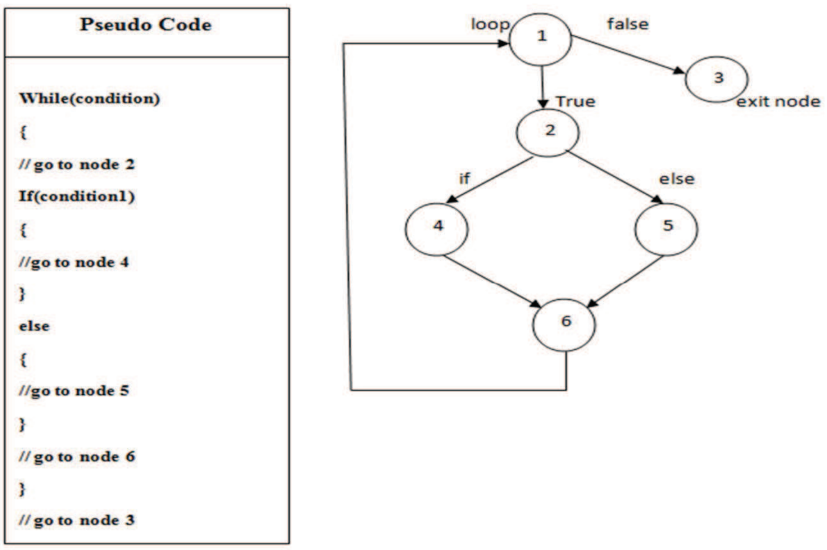
Table I: Software Complexity Metrics

Parameters	**Software Complexity Metrics**		
	LOC	**Halstead's**	**Cyclomatic Complexity**
Approach Used	Uses physical length of the code.	Uses the count of unique operators and operands.	Calculates the number of independent paths from control flow graph.
Software Life Cycle Phase	It can be calculated either at the coding stage or after the end of the complete life cycle phase.	It can be calculated only at the end of the complete life cycle phase.	It can be calculated at the design or code phase of the life cycle.
Bug Density	Concave relationship	Forecasts the bug density.	Highly related
Base used for calculation	Source Code	Source Code	Logic Structure
Language	Language independent	Language dependent	Language independent
Usability	Easy	Medium	Medium
Data and Control Statements	Ignores the complexity generated by the decision statements.	Considers the complexity due to data but ignores the complexity due to decision statements.	Ignores the complexity due to data but considers the complexity due to decision statements.
Theory Base	No	No	Yes
Additional Uses	Productivity and man effort can also be calculated.	Error rate , vocabulary, code length, difficulty level, volume, effort, time can be calculated.	Risks associated, effort, relative complexities can be calculated.
Popularity	Narrow	Wide	Wide

Different methods for Cyclomatic complexity: Three methods used to calculate the cyclomatic complexity:

# edges – # nodes + 2P
1. # binary decision statements + 1
2. # closed regions + 1

A First Method:
In first technique creates a control flow graph of a program's source code and then measures all linearly independent paths from the graph. This control flow graph consists of nodes which are connected by edges and then complexity is measured through this control flow graph.
Cyclomatic complexity measured by:
CC (Cyclomatic complexity) = #edges-#nodes+2P
Where, E is the number of edges, and represent the flow of control between nodes, N is the number of nodes represent expressions and statements and P is the number of connected component[9],[10].



Pseudo Code

```
While(condition)
{
//go to node 2
If(condition1)
{
//go to node 4
}
else
{
//go to node 5
}
// go to node 6
}
// go to node 3
```

Fig. 1: Pseudo code with Control Flow Graph

So, in first technique control flow diagram is used to find the cyclomatic complexity. There are total 6 nodes and 7 edges in this control flow graph.
Control flow graph actually represents the logic structure of a program's source code or module which has only single entry point and exit point.
Control Flow Graph is very easy to understand and always gives useful results. In CFG there is a node labeled Start that has no incoming edge, and another node labeled End that has no outgoing edge. From that graph now calculate the number of edges and nodes and put in formula.

B Second Method:
In the second method McCabe cyclomatic complexity is calculated by determining the number of decision statements which are caused by conditional statements in a program and plus one.
So, it is one of the simplest method for calculating the cyclomatic complexity because we directly count the number of decisional statements like if, loops(while, for, do-while) etc. from the source code.
Cyclomatic complexity = # decision statements + 1[9].
In above mentioned code numbers of decision or conditional statements are two (if and while). So, cyclomatic complexity from this method is also 3.
There are some basic rules that can be used to measure cyclomatic complexity.

1. Calculate the number of if/ then, else if but do not count the else statements in the program.
2. Find the switch statement and count the total of the cases in the program but do not count the default in the program.
3. Calculate all the loops like for, while and do-while statements and also all the try/catch statements in the program.
4. Count conditional operator && and || operator and also ternary operators like ?: from the expression.

Now add one to the numbers from the previous step numbers.

C Third Method:
In third method,
Cyclomatic complexity = # enclosed areas + 1
Calculate the number of closed regions from control flow diagram. Here, number of closed regions are 2.
So, Cyclomatic complexity is= 2+1=3.
From all three methods we get the same Cyclomatic complexity number and Cyclomatic number will be equal to number of independent paths in the graph.
Discussion: In this section the focus is on the cyclomatic complexity. Why this is preferred over two metrics LOC and Halstead's metric, Different methods to calculate the Cyclomatic Complexity number and what will be the effect of this cyclomatic

number on other parameters.
All three metrics are used to measure the software complexity but only cyclomatic complexity metric measures the complexity of software from conditional statements. Cyclomatic complexity metric uses three methods to calculate the cyclomatic complexity. In first method control flow graph is generated from the source code and then calculate the number of nodes and edges from graph.

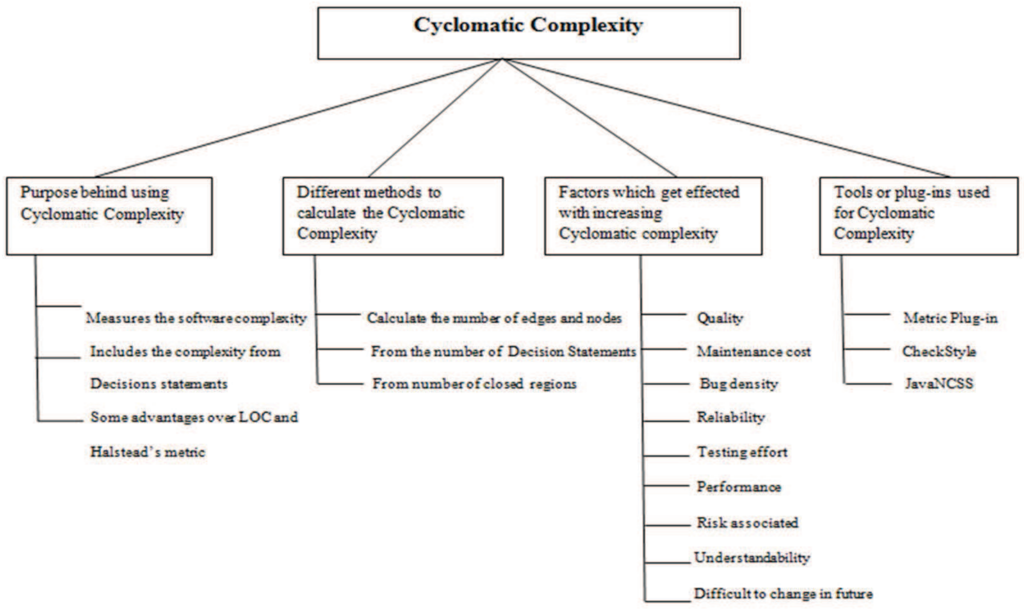

Cyclomatic Complexity
Purpose behind using Cyclomatic Complexity
Different methods to calculate the Cyclomatic Complexity
Factors which get effected with increasing Cyclomatic complexity
Tools or plug-ins used for Cyclomatic Complexity
Measures the software complexity
Includes the complexity from Decisions statements
Some advantages over LOC and Halstead's metric
Calculate the number of edges and nodes
From the number of Decision Statements
From number of closed regions
Quality
Maintenance cost
Bug density
Reliability
Testing effort
Performance
Risk associated
Understandability
Difficult to change in future
Metric Plug-in
CheckStyle
JavaNCSS

Fig. 2: Cyclomatic Complexity metric

Second method uses or calculates the number of decision statements directly from source code and in third method number of regions are calculated from the control flow graph. After that if this calculated cyclomatic number is greater than 10 then there will be chances of more errors in the software then more testing effort will be required to find these errors. So, reliability, performance, maintenance cost, quality and other factors will get affected.

Conclusion: McCabe's cyclomatic complexity is one of the best metric used to measure the complexity of software among LOC and Halstead's metric. It is language independent metric and can be calculated at the design or code phase of the life cycle of software and along with effects many factors like performance, quality etc.

References:

1. Thomas J. Mccabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Dec. 1976, Vol. SE-2, No. 4, pp.308 - 320.
2. Vincent Y. Shen, Tze-Jie Yu, Stephen M. Thebaut, "Identifying Error-Prone Software-An Empirical Study", IEEE Transactions on Software Engineering, April 1985, Vol. SE-11, No. 4, pp.317 - 324.
3. Ayman Madi, Oussama Kassem Zein and Seifedine Kadry, "On the Improvement of Cyclomatic Complexity Metric", International Journal of Software Engineering and Its Applications, March 2013, Vol. 7, No. 2.
4. Ambuj Kumar Agarwal, Dr. Vinodini katiyar, "Implementation of Cyclomatic Complexity Matrix", Journal of Nature Inspired Computing (JNIC), 2013,Vol. 1, No. 2.
5. N. I. Enescu, D. Mancas, E. I. Manole, and S. Udristoiu," Increasing Level of Correctness in Correlation with McCabe Complexity", International Journal of Computers, 2009, Vol. 3.
6. Geoffrey K. Gill and Chris F. Kemerer, "Cyclomatic Complexity Density and Software Maintenance Productivity", IEEE Transactions on Software Engineering, Dec. 1991,Vol. 17, No. 12, pp.1284 - 1288.
7. Graylin Jay, Joanne E. Hale, Randy K. Smith, David Hale, Nicholas A. Kraft, Charles Ward,

"Cyclomatic Complexity and Lines of Code, Empirical Evidence of a Stable Linear Relationship", J. Software Engineering & Application, June 2009, pp.137-143.

8. Sheng Yu, Shijie Zhou, "A Survey on Metric of Software Complexity", IEEE International Conference on Information Management and Engineering (ICIME), 16-18 April 2010, Chengdu, pp. 352 – 356.
9. M. R. Woodward, M. A. Hennell and D. A. Hedley, "A measure of control flow complexity in program text", IEEE Transactions on Software Engineering, Jan. 1979,Vol. 5, No. 1.
10. Mir Muhammd Suleman Sarwar, Ibrar Ahmad, Sara Shahzad, "Cyclomatic Complexity for WCF: A Service Oriented Architecture", Frontiers of Information Technology (FIT) 10th International Conference, 17-19 Dec. 2012, Islamabad, pp.175 – 180

* * *

Thapar University, Student, ankitagarg60@gmail.com
Thapar University, Assistant Professor, vkbhalla@thapar.edu