Advanced Program Complexity Metrics and Measurement

Abdul Jabbar. P
Full Time Ph.D., Research Scholar
School of Computer Science and Engineering
Bharathiar University, Coimbatore

S.Sarala
Asst. Professor
School of Computer Science and Engineering
Bharathiar University, Coimbatore

ABSTRACT

The complexity metrics has an important role in assessing the quality of source code. Obtainable complexity metrics is failed to accumulate precise system failure information. In this work proposes adequate complexity metrics for structural measure which collect the exact system information that cause the system failure in source code. Then, by censure McCabe's cyclomatic number and the framework of adequate metrics that can be extensively observe the structural complexity of the various statements in the code. The proposed metrics verifies the accuracy of the source code evaluation, gains the proportion of software complexity metric by the experimental approach whereas, software complexity measurement is consistent with the actual result.

General Terms

Complexity measurement

Keywords

Advanced Complexity Measurement, Decision Statement Weight, Total Information (Ti), McCabe Cyclomatic number.

1. INTRODUCTION

Software engineering is a standard approach to develop reliable software based problem solving tasks. The accuracy and consistency of the up-and-coming computer based on internal and external systems attribute. The working software might be unsuccessful due to complexity in the system and the related components [10].

System reliability engineering is act as an important role to identify system failure prediction in various levels of development process which includes hardware reliability and software reliability. Hardware defects are repeatable and predictable because hardware failure caused by a physical problem in a machine, structure, system, especially one that prevents it from functioning correctly. The software failure frequently caused by the faults, complexity and method in the solution are implemented. To be exact, software failures happen due to the human design error, infrequently predicting at run time because requirements of software can be dissimilar in different versions [5]. Software complexity metrics are used as input in software reliability measures. In order to measure software reliability several techniques have been proposed. Nearly all techniques consider failure data to measure software development and operational environment in a system. The authors have explained the various software reliability models and its features in [2], [3], [4], [5], [7], [8] and [11]. The challenges are remain exist with these measures; hence software reliability engineering is in use an active subject for Stakeholders to develop consistent software. Software complexity is the intricacy of system components in the problem solving task. Over the years, a collection of software complexity metrics are proposed and it using input information for many code refactoring tools [6]. In this work express the various complexities involved in a system development process. Further describes and verifies the deficiency of existing control flow complexity metrics to measure the complexity of the source code. The proposed metrics are considered the source code behavior and predict the complexity in a module. From the experimental result, the existing metrics are analyzed and compared. The results are proved with a better prediction.

2. SOFTWARE COMPLEXITY METRICS

Software complexity metrics can be defined as failure caused by abnormalities in the code. Software complexity measurement tends to have estimates the lines of code in the problem solving task, when code length highly interrelates with structural complexity of the system [12]. The scientific and engineering literatures propose a number of complexity metrics. The predictive capability of metrics is imperative and employ in software testing and its application [14]. The complexity metrics are essential for the computational measurement subsequently the reliability model has been determined [16].

2.1 Classification of Complexity Metrics

Complexity affects the productivity and quality of a project, so it is an active component among the researchers. The author [6] described a large number of complexities that affects software project in the various levels of development processes. However complexity measurements concentrate only less number of complexity measurements such as cyclomatic complexity and essential complexity. And extend 50 variants of structural complexity for programming which may found to be relevant for software applications. The scientific and engineering literature described important of various complexities. Even though, varieties of scientific complexity are not yet used in software reliability engineering. In table 1 shows the various complexities affect on software projects. Each of the complexities affects project outcomes, software sizing, project scheduling, cost estimation and quality assurance in many aspects.

2.2 Issue of Complexity Metrics

On hand software metrics moderately reflect the physical attributes such as lines of code, input, output data and various controls in the source code. Although many complexities affect project sizing and outcome in considerable, no metrics defined to identify exact code complexity. Therefore well defined complexity metrics to describe, analyze and forecast the error prone is most essential.

Table 1. Various complexity affect software project

Complexity affect sizing software	Complexity affect project outcomes	Other important complexity
Cyclomatic	Algorithmic	Computational
Essential	Entropic	Diagnostic
Problem complexity	Mnemonic	Fan
Function point	Organizational	Flow
Data	Process	Graph
Halstead	Semantic	Information
	Logical	Perceptional
		Combinatorial
		Syntactic
		Topological

2.3 Complexity Measures

Complexity measure estimates durability of system components and its exchanges. The table 1 explains various complexities which categorized sizing software project and project outcome. The code refactoring tool use cyclomatic complexity and essential complexity as input information [6].

The authors [7], [13] have referred the program complexity which is measured using McCabe's cyclomatic number. The author [9] proposed cyclomatic complexity, program flow based on a graph *G* in graph theoretical concept, which counts the number of linearly independent paths through the code. The cyclomatic complexity is,

$$v\;G\; = e - n + 2 \qquad (1)$$

Consider the flow graph from figure 1 e=6 n=6 *v (G)* =2

The author [7] indicates McCabe's cyclomatic complexity mathematical simplification, if *d* is the number of decision nodes in *G* then cyclomatic complexity is

$$v\;G\; = d + 1 \qquad (2)$$

Consider flow graph from figure 1 *d=1, that is v(G) = 2*

2.4 Limitation of Cyclomatic Number

Cyclomatic complexity easily predicts and maintains large program or large number of decisions. Hence the various software tools consider the cyclomatic number to measure the source code. Simplify the cyclomatic number and the measure would be more than one of the total decisions in the code. In consequence it represents a partial observation of complexity of

```
ELOC 1   { int lo=1, int hi=1;
ELOC 2      System.out.printline(lo);
ELOC 3      while(hi<50)  {
ELOC 4      System.out.printline(hi);
ELOC 5     hi=hi+lo;
ELOC 6    lo=hi-lo  }}
```

Source Code

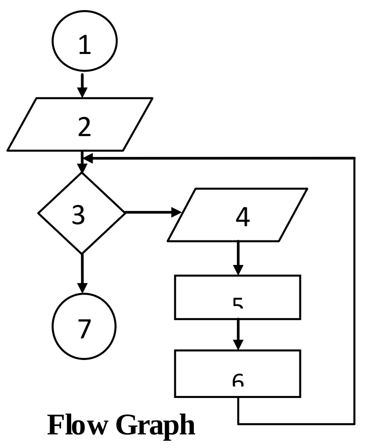

Figure1. Flow graph and source code of the Fibonacci sequence for the values <50(ELOC: - Effective line of code)

source code, however cyclomatic number is imperfect to measure the program complexity [7].

In (1) cyclomatic number does not take into account in the order of the nested loop statements. Moreover it seems same complexity of loop statement and selection statement. For instance, the cyclomatic number is "2" of selection statement and loop statement in figure 2. Data processing is an important system activity but cyclomatic complexity excludes input and output data. Although cyclomatic number ignores line of code, but structural complexity measure shows a high correlation with the number of lines of code [12]. Accordingly McCabe's cyclomatic number is insufficient to measure overall complexity of a program.

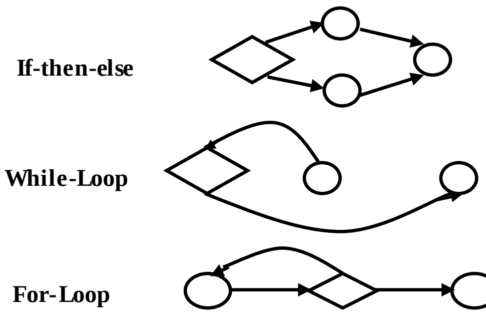

Figure 2. Graph of selction and loop statement Flow graph

The challenges on software complexity metrics remain lack of a complete model of program complexity. Complexity affects project sizing and outcomes however no metrics are successful to identify the complexity of source code. Existing metrics like McCabe's cyclomatic number is deficient to measure the parameters and actual outcomes over the implementation of the modules. In this research proposed an APCM metrics which is extensively considers the deficiency of existing measures.

3. ADVANCED COMPLEXITY MEASUREMENT

Complexity affects a system project sizing, project outcome and quality assurance. The abnormality in a code cause complexity and is determined that evaluate the control flow and data flow of a source code. The various flows and data in the code affect the project performance. In [9] McCabe used flow graph theory to count the sub flows from the code which is inadequate to measure the various levels of the complete system. Hence a new metrics has proposed. In this metrics proportionality define the interrelation of total information and control flows in a source code [15].

The correlation between control flow and data flow defined in the context of proportionality theory [15]. In the first phase, all control flow statement is selected, typically with the help of code reflecting of program sub flow. These controls then calculate the decision statement weight. The second phase consists of selecting the total information from the code using fan-in fan-out technique [1]. Using these attributes APCM is formulated. Figure 3 shows the APCM measurement strategy.

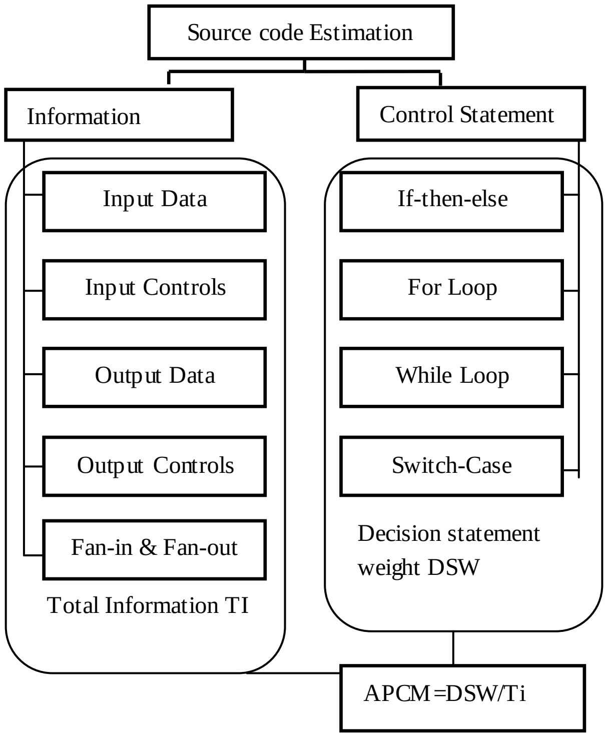

Figure. 3. APCM Estimation method

Total information in a source code *Ti*, Decision Statements weight in the code DSW and Advanced Program Complexity Measurement (APCM) is,

$$APCM = DSW / Ti \quad (3)$$

In order to measure the source code DSW is increase APCM increase proportionally. Let DSW and APCM, APCM is (directly) proportional to DSW (APCM and DSW vary directly, or x and y are in direct variation) if there is a non-zero constant Ti such that DSW=APCM*Ti [15].

Total information raise in the source code APCM is decrease. APCM and Ti are inversely proportional or reciprocal proportion if one of the metrics is directly proportional with the reciprocal of the other, or equivalently if their product is a constant. APCM is inversely proportional to the variable Ti if there exists a non-zero constant DSW such that Ti=DSW/APCM [15].

Where *Ti* is the total information handled by the module which include Number of input data parameter (id), Number of input control parameters (ic), Number of output data parameters (od), Number of output control parameter (oc), global variable used as data (gd), number of global variable used as control (gc), number of modules called (fi, fan-in), number of number of modules calling by this module (fo, fan-out). In [1] proposed the fan-in and fan-out measurement for a module. The total information *Ti* in a source code is,

$$Ti = id + ic + od + oc + gd + gc + fi + fo \quad (4)$$

Number of decisions in the code *d*, and count of decision control variable of each decision *di is D*. Decision Statements Weight in the code is,

$$DSW = (\mathrm{d1} * \mathrm{D1}) + (\mathrm{d2} * \mathrm{D2}) + ..(\mathrm{dn} * \mathrm{Dn}) \quad (5)$$

Number of decisions in the code *d* can be measure using the McCabe cyclomatic number v (G). In (2) explained cyclomatic complexity is one more number of decisions in the code.

$$\mathrm{d} = \mathrm{v(G)} - 1 \quad (6)$$

Assume the selection statement or loop statement in a module is zero (*d=0*) then DSW, ((d_1*D) + (d_2*D) +... ($Dn*D$) = 1.

3.1 DSW calculation in various controls

Source code includes various control statements to perform the task. DSW calculation measures the weight of the each control. The control flow predicts from the various controls analysis in the source code. In this case, DSW is calculated up on the conditional statement defined to the objective flow in the code. The key point of DSW how to measure the sub flows and recognizes the code structure. According to this all sub flow can be differentiate based on its features. The DSW calculation shows the controls that are essentially in the source code which follows as,

TABLE 2Comparison of Complexity Metrics

NO	ELOC	d	DSW	Ti	V(G)=d+1	APCM
1	6	1	7	10	2	0.7
2	37	5	15	16	6	0.9375
3	30	11	25	17	12	1.4705
4	18	2	10	5	3	2
5	21	1	5	10	2	.05

ELOC effective lines of code, d= Decision statements in the code, DSW Decision statement weight, Ti= Total information in the code, v(G)=d+1= Cyclomatic number, APCM= Advanced program complexity measurement.

3.1.1 *If then Else*
The decision handles two statements, the weight of the code *D=2.*

3.1.2 *Nested If*
The nested if statement contains number of decisions *d1, d2...dn,* hence the number of decisions are counted.
$DSW= d_1+d_2+....d_n,$ where each d=1, D=1 in all case

3.1.3 *For Loop*
Loop execute according to predefined criterion, so *D=n*, DSW=*d1*1+d2*n.*

3.1.4 *Nested For Loop*
In the case of nested loops, let d_i, d_{i+1}, $d_{i+2,...}$ d_{i+n} are the loops and count of decision control variable of d_i *is* D_i *and* d_{i+1} is $D_{i+1}= D_{i+1}*D_{((i+1)-1)}$, $D_{i+2=} D_{i+2}* D_{((i+2)-1)}$ and $D_{i+n=} D_{i+n}* D_{((i+n)-1)}$.

3.1.5 *While Loop*
DSW estimation in the case of *Do-Loop while* and *Do-Loop until* are two various aspects such as top- test loop and bottom test loop. In top- test loop *d=1* and *D=n* in all case of *Do-Loop while* and *Do-Loop until* apart from some exceptional coding, in this case *DSW = d*n.*

The bottom-test Do Loop shows different in DSW estimation. *Do-Loop while* and *Do-Loop until* shows *d=1* and *D=n+1* because the bottom-test Do Loop is execute at least once without checking condition. Suppose n= *0,* then *D=1* in bottom-test Do Loop.

In figure 4, according to the various parameters of DSW and Ti the complexity was varied. The reduced amount of decision statement weight in complexity1 then the program complexity is less. And complexity3 shows decision statement weight is high consequently it proves high complexity. When compare complexity3 and complexity4, the complexity3 dealt with more data than complexity4, thus the complexity4 proves high complexity. Figure4 represents APCM value near zero which is less complexity and away from one indicates rising error prone or complexity in the source code.

The source code contains considerable decision statements and it manipulates less amount of data cause high complexity. Explicitly APCM explore structural measures of software complexity such as control flow and data flow. The measure indicates tangible program complexity measure from the empirical research done in various program modules in table 2. The APCM value in the sample code evaluation of figure.1 shows in row 1 of table.2 which is indicates low complexity, because the code shows high data control over less decision that can be noticed the correlation between control flow and dataflow. The comparative analysis in table 2 displays the APCM most consistent over McCabe's cyclomatic number and other complexity metrics to measure complexity of source code.

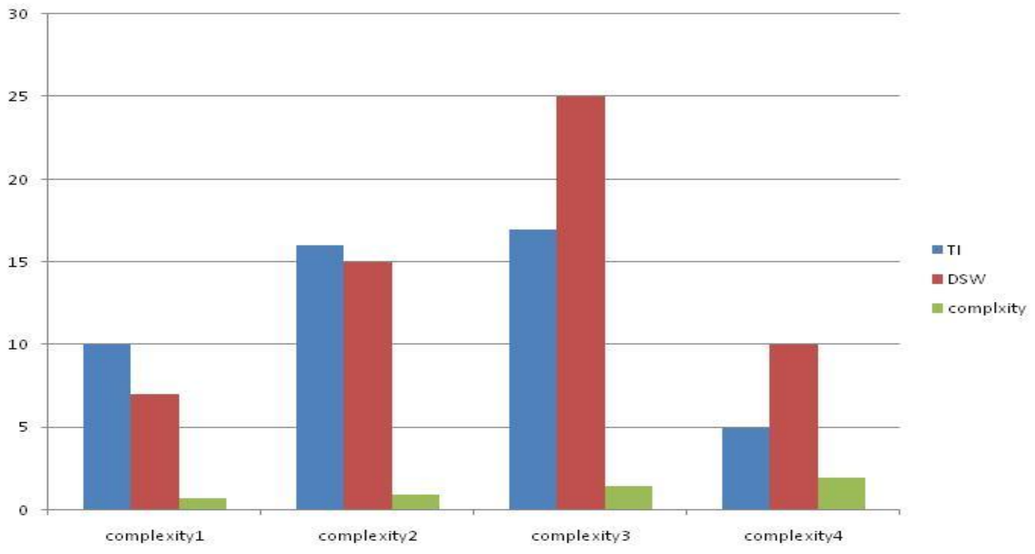

Figure 4 APCM measurement variations in different Ti, DSW parameters

The accuracy of the APCM metrics is calculated on the basis of the number of properties satisfy the code instances and attributes which were predicted acceptably. The author's work [17] has been described to estimate the accuracy, six- properties validation was applied.

4. CONCLUSION

To measure the source code, existing complexity metrics are performed with the evaluation of failure rate and failure density in spite of its deficiency is censured. In this work APCM is proposed to evaluate source code. APCM evaluate all parameters in any software module and to find exact complexity in it. The complexity interrelates with the structure and the various methods implemented in the source code. The boundless possibility is attempted with the code. Further the complexity metrics are analyzed with instant skill. By this approach, a constant measuring process is discovered and examined with sample of source code. As the nonlinearity of existing metric attests in the evaluation, the outcomes of the proposed metrics are very efficient also improving the measurement process.

In spite of the heartening result current study, that identified lack of distinct metrics quality analyzing technology. The results of this work should therefore be inferred only as findings, which need to be simulated and confirmed. In addition, because of the nature of the evaluation tasks, in which metrics made a number of predefined changes to the parameters, this work was limited to control flow and dataflow of source code. The metrics authenticity evaluation by properties gives indication to the promising direction of future work. Evaluation strategy selection could be advance in to measurement and experiments to enhance the validation of the result to the scope that the terminations at present availability can be generalized.

5. REFERENCES

[1] S. Sarala, P. Abdul Jabbar, 2010. Information flow metrics and complexity measurement, 3rd IEEE International Conference on Computer Science and Information Technology, vol. 2 pp. 575-578.

[2] Wu-caihua, Zhu-xiaodong and Liu-juntao, 2008. The SRGM Framework of Integrated Fault Detection Process and Correction Process, International Conference on Computer Science and Software Engineering, IEEE Computer Society, pp. 679-682.

[3] Chin-Yu Huang, Sy-Yen Kuo, and Michael R. Lyu., 2007. An Assessment of Testing-Effort Dependent Software Reliability Growth Models, IEEE Transactions on Reliability, Vol. 56, No. 2, pp. 198-211.

[4] Chin-Yu Huang, Michael R. Lyu, 2005.Optimal Release Time for Software Systems Considering Cost, Testing-Effort, and Test Efficiency, IEEE Transaction on Reliability, vol. 54, NO. 4, pp. 583-591.

[5] Michael R. Lyu, 2007. Software Reliability Engineering: A Roadmap, Future of software Engineering (Fose'07), IEEE Computer Society WashingtonDC.USA, pp.153-170.

[6] T. Capper Jones, 2005. Estimating software cost, Tata McGraw Hill Edition.

[7] Norman E. Fenton and S. L Pfleeger, 2004. Software Metrics, Second Edition, Third Reprint, Thomson publication Singapore.

[8] Sy-Yen Kuo, Chin-Yu Huang, and Michael R. Lyu, 2001. Framework for Modeling Software Reliability, Using Various Testing-Efforts and Fault-Detection Rates, IEEE Transaction on Reliability, Vol. 50, No. 3, pp. 310- 320.

[9] McCabe, 1976. A Complexity Measure, IEEE Transaction on software Engineering, vol SE_2 . No.4, pp 308-320.

[10] S.C. Chiemeke, A. O. Oladipupo, 2001. Theoretical Approaches in Software Complexity Metrics, African Journal of Science and Technology (AJST) Science and Engineering Series, Vol. 2, No. 2, pp. 101-107.

[11] Srinivasan Ramani, Swapna S. Gokhale, and Kishor S. Trivedi, 1998. SREPT: Software Reliability Estimation and Prediction Tool, Springer-Verlag Berlin Heidelberg, pp. 27-36.

[12] Reymond PL Buse and Westley R. Weimer, 2009. Learning a Metric for Code Reliability, IEEE Transactions on Software Engineering 2010 (Accepted for publication).

[13] Tu Honglei, Sun Wei, Ahang Yanan, The Research on Software Metrics and software Complexity Metrics, International Forum on Computer Science- Technology and Applications", pp. 131-136.

[14] Ramon Sagarna, 2008. Jose A. Lozano. Software metrics mining to predict the performance of estimation of distribution algorithms in test data generation, Springer-Verlag Berlin Heidelberg, pp.235-254.

[15] Weisstein, Eric W., Proportional, MathWorld A Wolfram Web Resource.

[16] John C.M and Taghi M.K. 1991. The use of software metrics in software reliability modeling, *IEEE*, pp. 2-11.

[17] N.Salman, 2006. Complexity metrics as predictors of maintainability and integrability of software components, Journal of Arts and Sciences Cankaya University, pp 39-50.