Available online at www.sciencedirect.com

ScienceDirect

Procedia - Social and Behavioral Sciences 90 (2013) 497 – 503

Procedia
Social and Behavioral Sciences

6th International Conference on University Learning and Teaching (InCULT 2012)

The Use of Cyclomatic Complexity Metrics in Programming Performance's Assessment

Noraini Mohamed[a], Raja Fitriyah Raja Sulaiman[b], Wan Rohana Wan Endut [c*]

[abc]*Faculty of Computer and Mathematical Sciences,Universiti Teknologi MARA Pahang , Jengka,26400,Malaysia*

Abstract

Programming course is usually difficult and complex as taken from a student's point of view. Complexity of a programming code may be determined by the number of linearly independent path of its source code. This paper will be discussing the consistencies of the complexity of the code supposedly produced by students based on the programming questions found in the programming assessment throughout a semester of studies. The comparisons of the level of complexity and the students' grade for various assessments materials are done. Based on this a relationship is observed and the conclusion is drawn on whether the consistencies in producing a certain pattern of expected complexity of a programming code has an effect on students performance.

Keywords: Cyclomatic Complexity; programming; performance assessment

1. Introduction

Programming is an activity that deals with a lot of interacting entities such as the language, technique, approach and the platform chosen. Inside a working program there is a set of data operated using a set of operations. Thus programming is complex in terms of the process involved and the component that make up the program itself. Due to these complexities, there are many aspects of programming that can be measured to ensure good quality is present in the end product. One of the most fundamental entities that are subjected to measurement is the complexity of the code that made up the program.

Students' performance in a programming class usually reflects their working ability in using computer programs to solve a problem. Michael J. (2012) had suggested that higher order skills and advanced reasoning skills are required for the students who seek to make careers in computing industries. A suitable assessment must

* Corresponding author. Tel.: +609-4602604; fax: +609-4602488.
E-mail address: noraini_mohamed@pahang.uitm.edu.my

be designed carefully so that it will serve not only as an assessment material but also as a tool that might educate students. Apart from obtaining their grades students may get the correct exposure in programming from the question posed in the assessment material. This is also important in order to provide a fair treatment to the students' effort since their effort in producing the software programs indicates the intellectual energy (Podgorelec, 2010) they have built into solving the problem.

The motivation of this work is to establish a relationship between the cyclomatic complexity numbers of the expected programming problems solved by the students during the assessment period of a programming course with the grades they have obtained. For this purpose, McCabe (1976) cyclomatic complexity metric which is usually used to measure a quality of a programming code written by a programmer is used. The objective of this work is to make use of the assessment material as a platform for the student to be assessed and to gain required skills in programming.

2. Cyclomatic Complexity Metrics

The term complexity refers to a large number of interacting components in a system. If applies to a program's code, it means the path that the programs codes executes. Clark (2008) stated that one of the areas where a good quality program may exist is from the code complexity which is the most atomic substance of a computer program. Therefore the best place to start evaluating the quality of the program is to measure the complexity of the code itself.

"Cyclomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program. When used in the context of basis path testing method, the value computed for cyclomatic complexity defines the number of independent paths in the basis set of a program and provides us with an upper bound of the test that must be conducted to ensure that all statements have been executed at least once" (Pressman, 2001).

The complexity measure of a program is to measure the control number of paths through a program (Mccabe, 1976). It is based on the Control Flow Graph (CFG) theory. It is defined as:

$$V(G) = E-N +2$$

Where E is the number of flow graph edges, N is the number of flow graph nodes.

3. Literature Review

There are a number of researches conducted on the use of Cyclomatic Complexity (Mccabe, 1976). For example, Pádua (2010) worked on establishing the relationship between the measurement of software complexity and how effective and efficient the students overcome that complexity.

Vandana Bhattacherjee (2009) used complexity metric to estimate the student effort in programming in terms of comparing the program complexity and the cost needed to work on it. But he did not mention directly the effect of program complexity with the time taken for the students to complete their program. However, S. A. Mengel (1998) had managed to make use of this metric with the help of an automated tool named VERILOG LOGISCOPE to distinguish between the weak and better programs written by their students. Thus, they were able to distinguish the better and weak students.

For introductory programming class, one cannot expect students to write codes that make full use of the control structure which contributes to the control flow graph. There must be some level of complexity that the students must be able to cope with during their introductory course. For this, a threshold complexity was introduced (M. Lopez). But it only focus on Java programming language which is adopting an Object Oriented technique (OOP). There were 694 samples of software Java project are taken from Open Source community and industry to measure its program complexity. This study found that more than 90% of the methods have a complexity less than 5 and this result showed that most methods are simple.

One may argues that the complexity of a program will increase the number of errors in the program. But beyond certain threshold value of complexity then the errors will increase sharply (A. H. Watson, 1996). It shows that complexity needs not be put aside but rather should be incorporated inside the solution posed by a programmer.

Cyclomatic Complexity metric is also applied in measuring the complexity of critical software system which was done on Autonomous Vehicle Systems (Clark, 2008). The threshold values produced at the end of the program testing helped recognize the risky modules in the system. This study focus on code size, content and code complexity as the measuring software attributes. Code size is the easiest attribute to measure and the elements of code size are lines of code, lines of comments, lines of mixed code and comments, and lines left blank. The restriction of code size is the line counts cannot measure the difficulty of code, the effort needed to code a program and the algorithmic complexity of code units.

Clark (2008) also stated that a tool named static complexity analyzer has been used to analyze the code complexity and code maintainability of the system. The results of measurement are the threshold value for code complexity is 10, while for code maintainability is 4. Results showed that the program complexity of the system is difficult and it is helped developers to focus on the risky modules of the systems.

4. Research Method

This research was conducted on two groups of diploma level students which were from Computer Science and Civil Engineering Faculty. Data collected are based on the continuous assessment which comprises four quizzes, two assignments, and two tests. The data will be compared with the final examination paper. The continuous assessment contributed 50 percent of the final grade whereas another 50 percent was coming from the final examination.

The C++ language is used in the first semester Computer Science Diploma level and the third semester Civil Engineering Diploma level. The course titled Fundamental Problem Solving using Computer covers the basic C++ language constructs which are statements, expressions, selections, loops, functions and arrays.

For every assessment question which involves construction of either partial or full C++ program the Cyclomatic Complexity is measured. All questions which only satisfied level 1 and 2 of Bloom's Taxonomy are not measured in terms of Cylomatic Complexity. Average Complexity of the assessment of type quiz is taken since not all questions found in the quiz had asked the students to construct a C++ program.

Students' score on continuous assessment and final examination that involves construction of a working program are compared to the complexity gained for each assessment.

5. Result

Table 1 below shows the differences in Cyclomatic Complexity numbers calculated from the expected answers for both continuous assessment and final examination:

Table 1: Type of Assessment and Level of Complexity

Type of Assessment	Average Quizzes	Assignment 2	Assignment 2	Assignment 3	Assignment 4	Test 1	Test 2	Final Examination
Level of Complexity	1	1	2	3	5	4	4	7

The graph below shows the marks obtained in tests and the final exam question that involved programming from two different classes of students. It shows that students perform better in the final exam (series1) for those who scored high marks. But for those who scored low marks, the final exam marks is lower than in a test (series2) for the first group of students. For the second group of students, only 4 students managed to have their final exam's score higher than their coursework's score. This indicates that the final exam question contributed to their final performance.

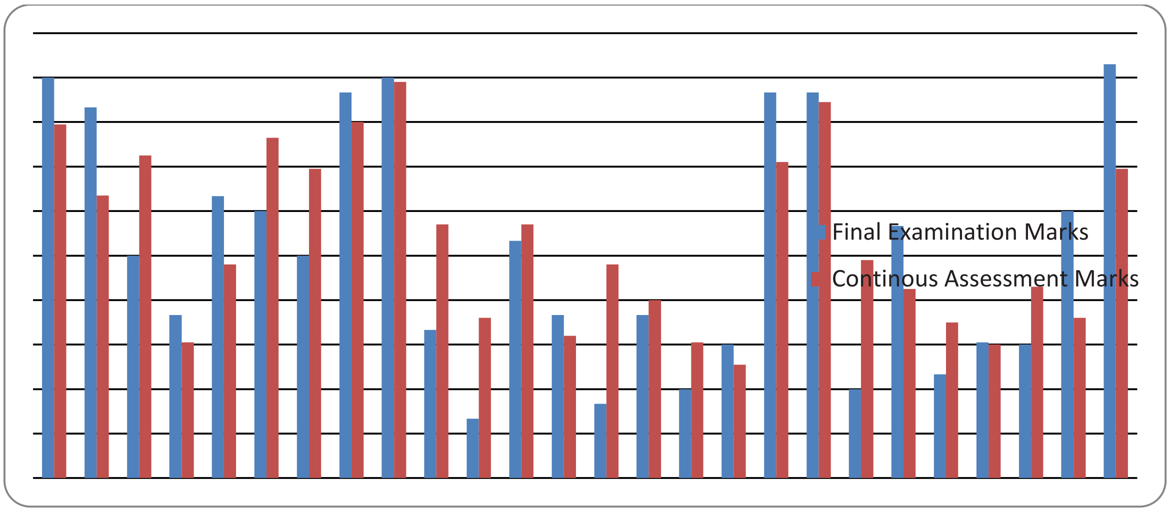

Fig. 1. (a) Computer Science Students

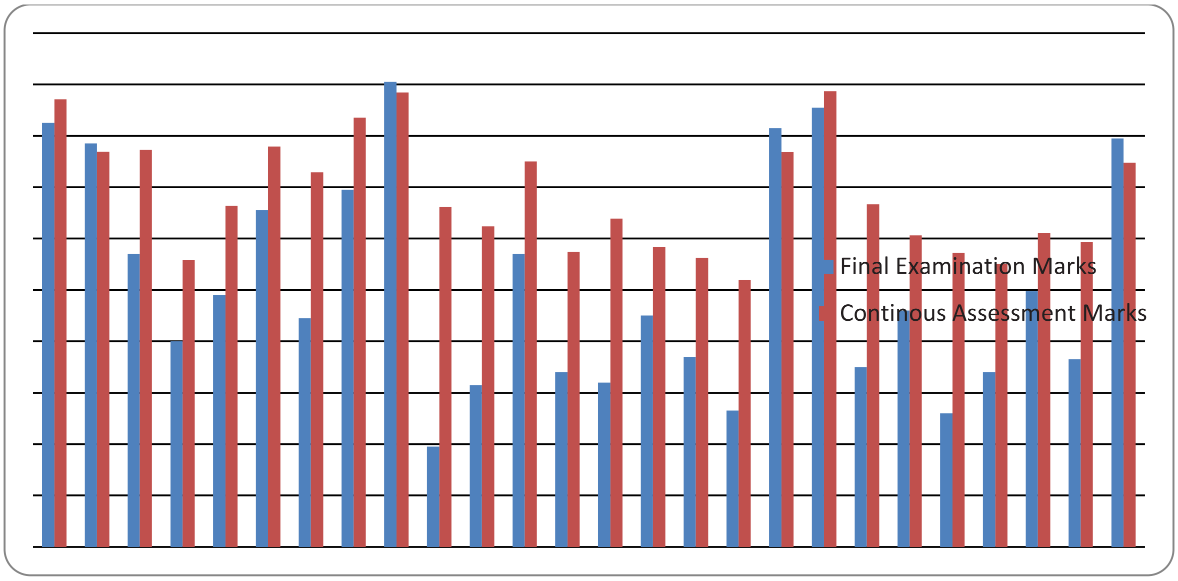

Fig. 1. (b) Civil Engineering Students

A relationship between the score and the complexity level of the expected answer is drawn and tabulated as a contingency table.

$$\begin{aligned} X^2 &= \frac{(x_1 - E_1)^2}{E_1} + \frac{(x_2 - E_2)^2}{E_2} + \cdots + \frac{(x_k - E_k)^2}{E_k} \\ &= \sum_{i=1}^{k} \frac{(x_i - E_i)^2}{E_i} \end{aligned}$$

Using chi-square test on a contingency table, it is found out that the chi-square value is 4.033 with 1 degree of freedom. It indicates that the association between the high score with the associated complexity level has a relationship with the number of failures among students.

For students who have done well with higher Cyclomatic Complexity number of the expected code written as a solution in a final exam, the majority of them passed the exam. This indicates that their score in the final exam is higher as compared to the continuous assessment. For student whose score is higher in a continuous assessment as compared to the final examination, the majority of them failed that particular assessment.

6. Discussion and Conclusion

The final exam usually possessed higher complexity readings as compared to the rest of the assessment. Thus it requires a more complex approach to the final exam questions. Given on next page is the programming code.

The Cyclomatic Complexity path that the program traversed is as shown below:

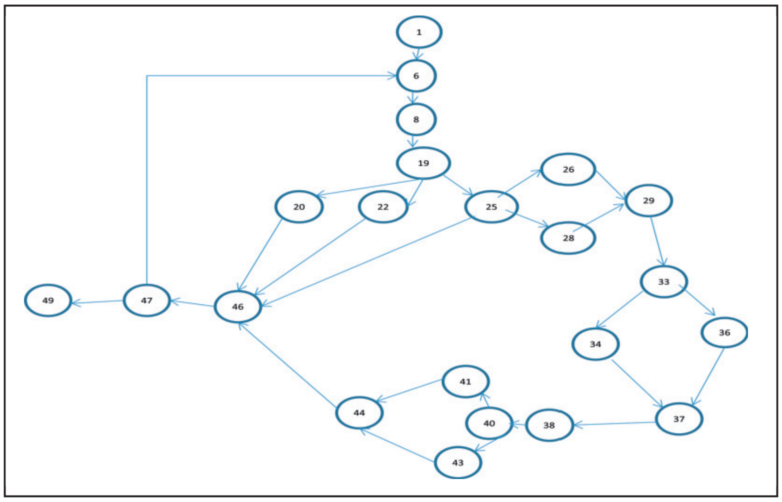

Figure 2. Cyclomatic Complexity Graph

```
#include <iostream>
#include <conio>
void main()
{
1  char filmCode,hallType;
2  int numTickets;
3  int totalGold=0,totalPremier=0;
4  float goldTickets,premierTickets;
5  char status='Y';
6  while(status=='Y'||status=='y')
7  {
8    cout<<"Welcome to Star    Entertainment"<<endl;
9    cout<<"Enter Code for Choice Of Film"<<endl;
10    cout<<"[T] Ong Bak III(Thai)"<<endl;
11    cout<<"[U] The Twilight Saga:Eclipse (USA)"<<endl;
12    cin>>filmCode;
13    cout<<"Enter type of hall code"<<endl;
14    cout<<"[G] Gold"<<endl;
15    cout<<"[U] Premier"<<endl;
16    cin>>hallType;
17    cout<<"Number of tickets?"<<endl;
18    cin>>numTickets;
19    if (filmCode!='T'&&filmCode!='t'&& filmCode!='U'&& filmCode!='u')
20       cout<<"Incorrect FILM CODE"<<endl;
21    else if (hallType!='G'&&hallType!='g'&&hallType!='P'&&hallType!='p')
22       cout<<"Incorrect HALL TYPE"<<endl;
23   else
24   {
25           if (hallType=='G'||hallType=='g')
26               totalGold+=numTickets;
27           else
28               totalPremier+=numTickets;
29           cout<<"STAR ENTERTAINMENT"<<endl;
30           cout<<"=================="<<endl;
31           cout<<"Film Code\t\t:"<<filmCode<<endl;
32           cout<<"Film Title\t\t:";
33           if(filmCode=='T'||filmCode=='t')
34              cout<<"Ong Bak III (Thai) "<<endl;
35           else
36              cout<<"The Twilight Saga:Eclipse(USA)"<<endl;
37           cout<<"Type of Hall\t\t"<<hallType<<endl;
38       cout<<"Number of tickets\t"<<numTickets<<endl;
39       cout<<"Total Price (RM)\t:";
40       if (hallType=='G'||hallType=='g')
41          cout<<(40 *numTickets);
42       else
43          cout<<20*numTickets;
44       cout<<"\n\n Thank You. Please come again\n";
45  }
46  cout<<"do you want to continue(Y/N)?"<<endl;
47  cin>>status;
48 }
49 goldTickets=totalGold*40;
50 premierTickets=totalPremier*20;
51 cout<<"STAR ENTERTAINMENT REPORT"<<endl;
52 cout<<"=========================="<<endl;
53 cout<<"Total Ticket Collection "<<endl;
54 cout<<"Gold Hall (RM)\t"<<goldTickets<<endl;
55 cout<<"Premier Hall (RM) \t:"<<premierTickets<<endl;
56 cout<<"Grand Total (RM)\t :"<<(goldTickets + premierTickets)<<endl;
57 getch();
```

From the graph in Figure 2, the Cyclomatic Complexity number can be computed as:

$$\begin{aligned} V(f) &= e-n+2p \\ &= 27-22+2(1) \\ &= 7 \end{aligned}$$

The same was done for the quizzes, assignments and tests. An average score of V(f) of quizzes is used since not all quizzes required the student to develop and write a program. The problem is that as shown in the graph, the complexity of the program expected out of the student being assessed via quizzes is only one, as compared to the time required for them to sit for a quiz which is part of a two (2) hours lecture. A portion of a few days (not exactly stated accordingly in hours) is also allocated for assignment whose complexity of the program expected is higher than of a quiz. For the test where complexity expected is much higher than the previously mentioned type of assessment students took three hours in answering a final exam question.

Final exam question possessed greater complexity than the continuous assessment. In conclusion, there is no consistency between continuous assessment and final examination in terms of the expected Cyclomatic Complexity and it has an effect on students' overall performance. Therefore, it is suggested that there is a need to set the threshold complexity value the students should take in the continuous assessment. Thus it will provide a fair treatment and better insight into students' programming ability.

References

A. H. Watson, T. M. (1996). *Structured Testing: A Testing Methodology*. Gaithersburg: Computer Systems. Laboratory.

M. Lopez, N. (n.d.). *Relevence Of the Cyclomatic Complexity Treshold for The Java Programming Lang*uage. citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.

M. N. Clark, B. S. (2008). *Measuring Software Complexity to Target Risky Modules in Autonomous Vehicle. Systems*. AUVSI North America Conference (p. 3). San Diego, California: AUVSI North America Conference.

Mccabe, T. J. (1976*). A Complexity Measure. IEEE Transactions on Software Engineering*(4), (pp. 308-320).

McCabe T.J, Michael J. O. (2012). *Practical Problem-Based Learning in Computing Education*. ACM Transactions on Computing Education (TOCE), 10:1-10:16.

Muth, D. F. (1985). Predicting *Performance In An Introductory Computer Science Course*. ACM(28).

Pádua, W. (2010). *Measuring complexity, effectiveness and efficiency in*. ICSE 10, (pp. 545-554).

Podgorelec, V. (2010). *Assessing the Current State of Software Evolution and Intellectual Energy Spent. Proceedings of the Workshop on Advances in Functional Size Measurement and Effort Estimation*. New York: ACM. doi:10.1145/1921705.1921709

Pressman, R. S. (2001). *Software Engineering: A Practitioner's Approach*. New York: McGraw Hill Higher Education.

S. A. Mengel, J. (1998). *Using Verilog Logiscope To Analyze Student Programs. FIE '98 Proceedings of the 28th Annual Frontiers in Education - Volume 03* (pp. 1213-1218). Washington: IEEE Computer Society.

Vandana Bhattacherjee, P. K. (2009). Complexity Metric For Analogy Based Effort Estimation. *Journal of Theoretical and Applied Information Technology*, 1-8.