International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor (2018): 7.426

A Comparative Analysis of Complexity of C++ and Python Programming Languages Using Multi-Paradigm Complexity Metric (MCM)

Balogun M. O.[1], Sotonwa K. A.[2]

Abstract: ***Software complexity metrics have used to quantifydifferent types of software properties such as cost, effort, time, maintainability, understanding and reliability. The existing metrics considered limited factors that affect software complexity, but do not consider the characteristics that affect complexity of multi-paradigm languages. In this work, a Multi-paradigm Complexity Metric (MCM) for measuring software complexity was developed for multi-paradigm codes. Multi-paradigm languages that were considered in thiswork are C++ and Python, these two languages combine the features of procedural and object oriented paradigms, therefore this research began with investigation of factors that affect the complexity of procedural code and object oriented code, so that the developed metric could be used not only for procedural code, but also either object oriented codes or in more general for multi-paradigm codes. The developed metric was then applied on sample programs written in most popular programming languages such as Python and C++, and the result of the developed metric was further evaluated with other existing complexity metrics like effective line of code (eLOC), cyclomatic complexity metric and Halstead complexity measures. The study showed that the developed complexity metric have significant comparison with the existing complexity metrics and can be used to rank numerous programs and difficulties of various modules.***

Keywords: Basic Control Structure,Cyclomatic Complexity Measure, Complexity of Distinct Class, Complexity of Inherited Class,Halstead Complexity Measure, Procedural Complexity, Multi-paradigm Code

1. Introduction

The requisite for improved quality control of the software development process has given rise to the discipline of software engineering, withpurposes of applying the systematic approach present in the engineering paradigm to the progress of software development. In ISO (2010) it was said that, if a software system is functional, reliable, maintainable, portable, useable, and efficient, then it is said to be of high quality. Alert, due to complexity of software system it is difficult to attain all these qualities during software development process. Software metrics have always been important tools in the development of software's. Software quality has been of rising demand for decades and some definitions have been provided throughout software history. According to (Sommerville, 2004) the most necessary software quality attributes is maintainability. To efficiently be able to maintain a software system, the codes should be understandable to the developer. For code to be easily understands, it has to be of low complexity. But in other to reduce software complexity, software metrics are used. Metrics are indicators of complexities; they expose several weakness of a complex software system. Software metric is used to measure some properties of a piece of software or its specifications. Since quantitative measurements are essential in all sciences, there is a continuous effort by computer science practitioners and theoreticians to bring similar approaches to software development. The goal is obtaining objective, reproducible and quantifiable measurements, which may have numerous valuable applications in schedule and budget planning, cost estimation, quality assurance testing, software debugging, software performance optimization, and optimal personnel task assignments. Reason for using software metrics as 'you cannot manage what you cannot measure'' therefore in order to monitor and improve software quality, measurement is essential said by DeMarco in 1986. Also McCabe and Watson in 2010 defined software complexity as ''one branch of software metrics that is focused on direct measurement of software attributes, as opposed to indirect software measures such as project milestone status and reported system failures''. It is believed that for coding and modifying a software system, a higher comprehensibility of the code is required. If the comprehensibility is higher, then the complexity of the software is lower, and therefore testing is easier.

Related to the definition above, software complexity metric can be classified according to paradigms as follows:
1) Procedural paradigm
2) Object-oriented paradigm
3) Multi-paradigm
4) Other paradigms

But the complexity factors and metric that are employed in this research is based on multi-paradigm metric which is a combination of procedural and object oriented paradigm.

2. Overview of Some Existing Metrics

According to Eclipse Metrics Plug-in 1.3.8 2010 'Most of the metrics used for measuring code complexity of procedural languages include Lines of Code, Cyclomatic Complexity Measure and Halstead Complexity Measures, while those of object oriented languages includes the chidamber and kemerer metrics suite, weighted class complexity just to mention but view.

Effective lines of code
This metric considers only the number of lines of code inside a program. According to (Resource Standard Metrics, 2010) Effective Line of Code Metric has the following types:
1) Lines of Code (LOC): counts every line including comments and blank lines.

2) Kilo Lines of Code (KLOC): it is LOC divided by 1000.
3) Effective Lines of Code (eLOC): estimates effective line of code excluding parenthesis, blanks and comments.
4) Logical Lines of Code (lLOC): estimates only the lines which form statements of a code. For example, in C, the statements which end with semi-colon are counted to be lLOC.

Limitation of this method includes: measurement is highly dependent on programming languages. A code written in Java may be much more effective than C. Also two programs that give the same functionalities written in two different languages may have very different LOC values.

Cyclomatic Complexity Measures (CCM)
Cyclomatic complexity is a static software metric pioneered in the 1976 by Thomas McCabe. The Cyclomatic complexity number (CCN) is a measure of how many paths there are through a method. CCM serves as a rough measure of code complexity and as a count of the minimum number of test cases that are required to achieve full code-coverage of the method.

McCabe's gives the formular for Calculating Cyclomatic Complexity as:
$m = e - n + 2p$
Where,
m is the Cyclomatic complexity
e is the number of edges
n is the number of vertices
p is the connected components

For example, if e= 8, n=10, p=2
$m = 8 - 10 + 2(2)$
$m = 4$

Halstead Complexity Measures
This metric was presented by Halstead in 1977. Halstead makes the observation that metrics of the software should reflect the implementation or expression of algorithms in different languages, but be independent of their execution on a specific platform. These metrics are therefore computed statically from the code. Halstead's goal was to identify measurable properties of software, and the relations between them. This is similar to the identification of measurable properties of matter (like the volume, mass, and pressure of a gas) and the relationships between them (such as the gas equation). Thus his metrics are actually not just complexity metrics.

The method includes:
(i) n1: the number of distinct operators,
(ii) n2: the number of distinct operands,
(iii) N1: the total number of operators, and
(iv) N2: the total number of operands.

The following values can be deduced from Halstead Complexity Measure:
Program Length => $N = N1 + N2$
Vocabulary Size => $n = n1 + n2$
Program Volume => $V = N^* \log 2(n)$
Difficulty Level => $D = (n1/2) * (N2/n2)$
Program Level => $L = 1/D$
Effort to Implement => $E = V * D$
Time to Implement => $T = E/18$
Number of Delivered Bugs => $B = E^{2/3} / 300$

Further research made it cleared that all these metrics considered limited factors that affect software complexity, and neglect many other factors responsible for the complexity of a code, they do not consider the characteristics of multi-paradigm languages at all, for instance line of code only considered the effective lines we have in a particular code, Cyclomatic complexity measures only make a basis path testing by measuring the flow of a program, while Halstead complexity measure only identifies the measurable properties in a code and the relationship that exist between those properties.

Chidamber and Kemeree metrics suite and weighted class complexity deals with only the weight classes ,weight methods and weight of subclasses we have in a code, neglecting many other factors responsible for the complexity of such a code been considered.

3. Proposed Methodology

The study investigates the factors that affect the complexity of Multi-paradigm codes and then developed a metric for Multi-paradigm programming languages. For validation of the metric, the metric is applied on some searching algorithms codes written in C++ and Python.

The developed metric
A new metric was developed for the procedural parts of multi-paradigm codes. The research was extended by considering also the object oriented parts of the codes. This means that the developedmetric, Multi-paradigm Complexity Measure (MCM) combines procedural and OO factors.

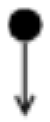
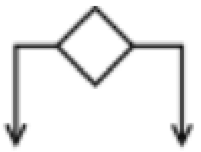
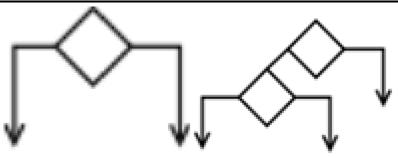
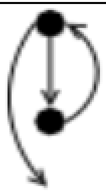
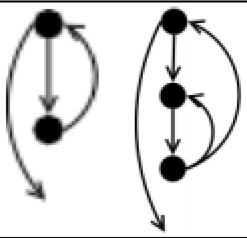
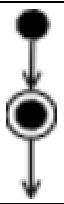
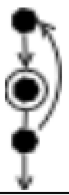
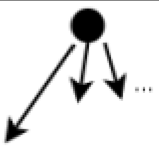
The following are recognised as the factors that are responsible for the complexity of multi paradigm language are.

Factors of complexity of procedural part:
a) Variables and constants
b) Basic Control Structures

Factors of Complexity of object oriented Part:
a) Attributes and constants,
b) Basic Control Structures; and
c) Classes.

The metric is developed in a way that it can measure OO and procedural parts separately. However, some programs may not cover OO features in a code. Then, 0 should be assigned to the parts that are not related with OO paradigm. Table 1presents basic control structure of MCM.

Table 1: Basic control structure for Multi-paradigm Complexity Measure (Wang and Shao 2003)

Category	CWU	Flow Graph
Sequence	1	
Condition	2	
Nested sub-condition	1	
Loop	3	
Nested Sub-loop	2	
Module call	2	
Recursion	3	
Exception	1	

Based on the mentioned factors, the research developed a metric for multi-paradigm codes as below.
Multi-paradigm Complexity Measurement (MCM):

$$MCM = CIclass + CDclass + PCCM \quad (i)$$

Where, CIclass = Complexity of Inherited Classes
CDclass = Complexity of Distinct Class
PCCM = Procedural Complexity

Although, PCCM measures the procedural complexity, it is assumed that using PCCM is difficult. MCM includes various complexity factors of PCCM. PCCM, being a part of MCM, would make the metric too complex and too difficult to apply. Therefore, it is recommended that Cprocedural is used in MCM instead of PCCM. However, it is possible to use MCM with PCCM for more detailed measurement.

$$\text{From(i) } MCM = CIclass + CDclass + Cprocedural$$

Cprocedural = Procedural Complexity

All these factors are defined as follows:
Cclass can be defined as complexity of a class. Cclass takes a major role in the calculation of both CDclass and CIclass. For example, for calculating CIclass, CDclassis needed. Cclass is defined as,
Cclass= W (attributes) + W (variables) + W(structures) + W(objects) – W(cohesion)

Where, Cclass = Complexity of Class(ii)
The reason of subtraction of cohesion is that it reduces the complexity and thus it is desirable from the point of view of software developers said by Roger in 2005.

Where, weight of attributes or variables is defined as

$$W(variablesorattribute) = 4 * AND + MND \qquad (iii)$$

Where, AND = Number of arbitrarily named distinct variable/ attributes
MND = Number of meaningfully named distinct variables/attributes
Weight of structure W (structures) is defined as:
W (structure) = W (BCS)
Where, BCS are basic control structure.
Weight of objects, Weight (objects) is defined as:
W (objects) = 2

An object created is counted as 2, because while creating an object constructor is automatically called. Thus, coupling occurs. Therefore, it is the same as calling a function or creating an object. Here it is meant to be the objects created inside a class.

Moreover, a method that calls another method is another cause of coupling, but that fact is added to MCM value inside Weight (structures).
Weight of cohesion is defined as:
W (cohesion) = MA/AM
Where, MA = Number of methods where attributes are used
AM = Number of attributes used inside methods
While counting the number of attributes there is no any importance of AND or MND.
CIclass can be defined as;

There are two cases for calculating the complexity of the Inheritance classes depending on the architecture:
i) If the classes are in the same level then their weights are added.
ii) If they are children of a class then their weights are multiplied due to inheritance property.

If there are m levels of depth in the object oriented code and level j has n classes then the Cognitive Code Complexity (CCC) of the system is given as

$$CIclass = \prod_{j=1}^{m}[\sum_{k=1}^{n} CC_{JK}] \qquad (iv)$$

CDclass can be defined as;
CDclass Cclass(x) + Cclass(y) + ------- (v)
Note: All classes, which are neither inherited nor derived from another, are parts of CDclass even if they have caused coupling together with other classes.
Cprocedural can be defined as;
Cprocedural= W (variables) + W (structures) + W (objects) – W (cohension)(vi)
Weight of variable W (variable) is defined as:
From equation (iii) W (variables) = $4 * AND + MND$
The variables are defined globally.
Weight of structure W (structures) is defined as:
W(structures) = W(BCS) + object.method

Where, BCS are basic control structure, and those structures are used globally. ‘object.method’ is calling a reachable method of a class using an object.‘object.method’ is counted as 2, because it is calling a function written by the programmer. If the program consists of only procedural code, then the weight of the ‘object.method’ will be 0.
Weight of objects, W (objects) is defined as:
W (objects) = 2
Creating an object is counted as 2, as it is described above. Here it is meant to be the objects created globally or inside any function which is not a part of any class. If the program consists of only procedural code, then the weight of the ‘objects’ will be 0.

$$W\ (\text{Cohesion}) = \frac{NF}{NV} \qquad vii$$

Where, NF is number of functions, and NV means number of variables. Coupling is added inside W (structures) as mentioned in the beginning of the metric description.

4. Demonstration of the Metrics and Discussion of the Results

For demonstration of MCM, linear and binary search algorithms codes were considered, the codes were written in two different programming languages, which include C++ and Python. Since the metric consist of both procedural and object oriented part, so in the Table the parts that are in dark colour represents the object oriented part of the code, from which the complexity of the distinct class is calculated, while the light part represent the procedural part. Table2 is showing the demonstration of binary search algorithm written in python, while calculation of MCM is shown immediately below the table3, just to show how the proposed metric was implemented.

Table 2: Linear Searching Demonstration in Python

	Att	str	var	obj	MA	AM	Cohesion	Complexity
import time					2	1		
a = ["0"]	1							1
class ClassOfSearchAlgos:								
deflinear_search(self, item_list, wanted):								
fl = False			5					5
msg = "Using Linear Search, The Element Found At position "		1	1					2
for i in range(len(item_list)):		7	5					12
if int(wanted) == int(item_list[i]):		6	6					12
msg = msg + str(i) + "," + " "		4	6					10
fl = True			5					5
if fl == False : msg = "Using Linear Search, The Element Not Found"		3	6					9
return msg			1					1
defload_data_from_file(filename):		1						1
i = 0			5					5
with open(filename) as fp:			1	2				3
for line in iter(fp.readline, ''):		7	1					8
a.append("0")	1	2	1					4
i = i + 1			9					9
a[i] = line	1		5					6
print(a[i])	1	2	4					7
load_data_from_file("C:\EclipseWorkspaces\csse120\Tolu\codes.txt")		3						3
wanted = input("search: ")		3	1					4
x = ClassOfSearchAlgos()				2				2
print (x.linear_search(a, wanted))	1	4	1					6

MCM = CIclass +CDclass + Cprocedural
From the table CIclass = 0
CDclass = 5+2+12+12+10+5+9+1+1+15+3+8+4+9+6+7 =99
Cprocedural = 1+3+4+2+6 = 16
Therefore, **MCM** =0 +99 +16 =115

Table 3: Comparison of the developed metric (MCM) with the existing metrics

Program	MCM	eLOC	CC	V	D	E	T
Binary Search C++	266	93	5	8388	61.5	515856	28659
Linear Search C++	149	51	5	4021	40	160824	8935
Binary Search Python	117	31	6	2664	38.1	101487	5638.2
Linear Search Python	115	24	5	1910	28.4	54148.5	3008.3

The results showed that linear search algorithm written in C++ and Python gave 149and 115 respectively for MCM. While binary search algorithm written in C++ and python also revealed 266and 117 respectively for MCM. Evaluating the results with other established complexity metrics revealed that binary search algorithm coded in C++ with MCM of 266, cyclomatic complexity (CC) of 5, effective line of code (eLOC) of 93 and Halstead volume (V) of 8388 has the highest complexity values. Furthermore linear search algorithm coded in Python with MCM of 115, eLOC of 24, CC of 5 and Halstead volume of 1910 has the lowest complexity value. The following figures were used to further validate in efficient of the proposed metric (MCM). The follow figures were also used to validate the efficiency of MCM over the other metrics considered in the work.

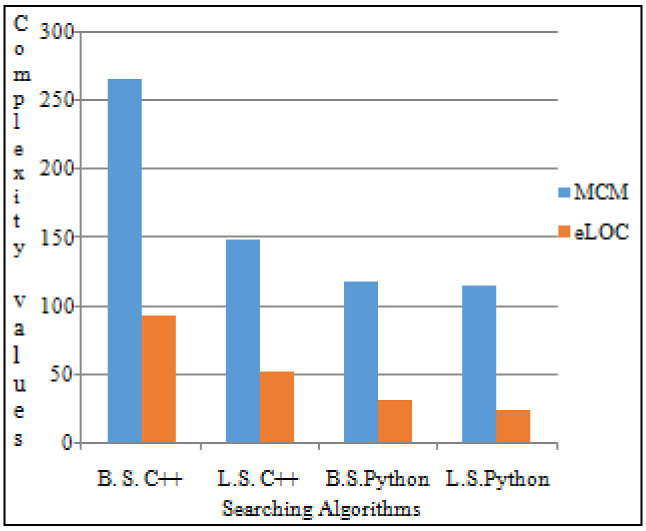


Figure 1: Graph of Comparison between MCM and eLOC

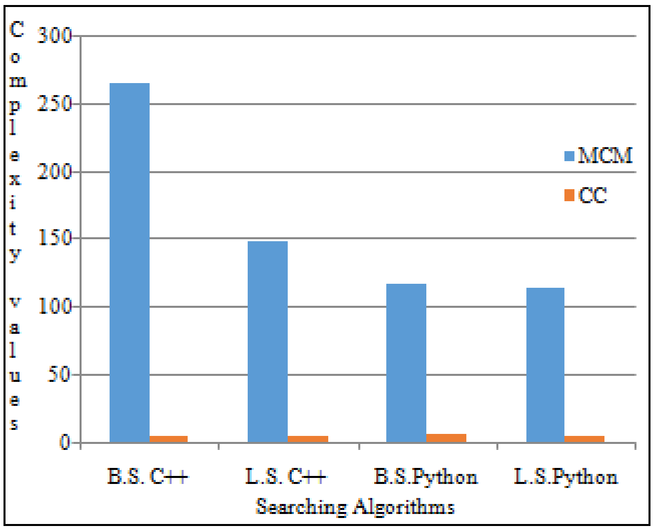


Figure 2: Graph of Comparison between MCM and CC

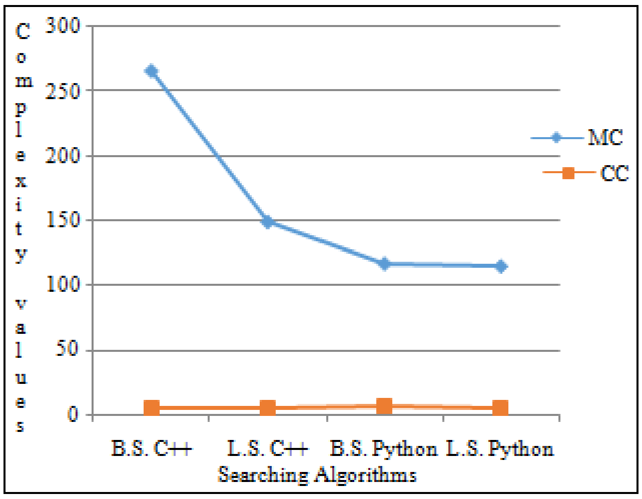


Figure 3: Relative Graph between MCM and CC

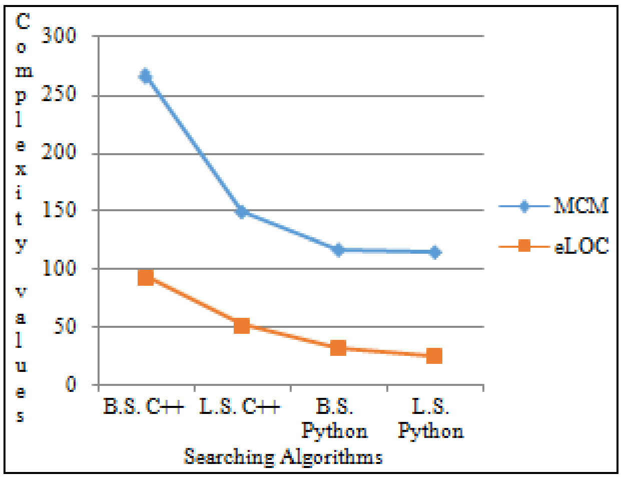


Figure 4: Relative Graph between MCM and eLOC

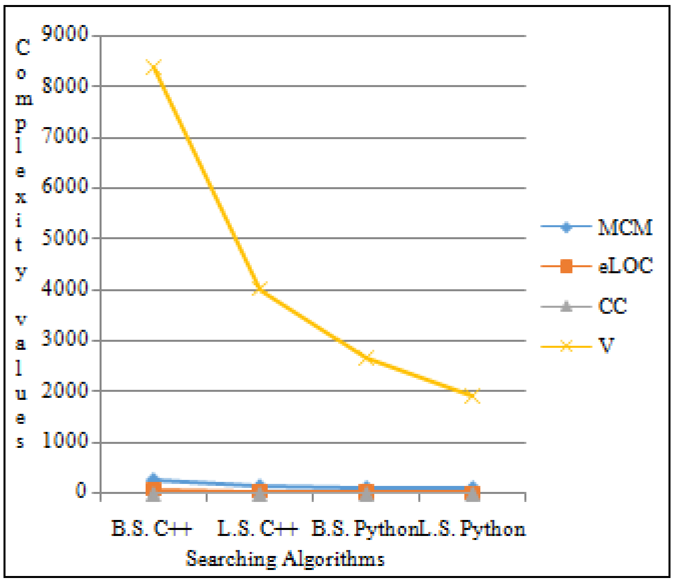


Figure 5: Relative Graph between MCM, eLOC, CC, V and D

5. Conclusion

In this study, a complexity metric was proposed to include the factors that affect the complexity of Multi-paradigm Programming Languages. The proposed metric was formulated to include the factors of procedural and object oriented languages. Various existing software complexity metrics such as effective line of code (eLOC), cyclomatic complexity measure (CC) and Halstead complexity metric were reviewed in other to applied them to sample programs written in Multi-paradigm languages such as C++ and Python

Two case studies to measure the complexity of linear search and binary search algorithms were discussed. The complexity of each of these codes were measured using Multi-paradigm complexity metric, the results was later compared with those of the existing metrics.

It was discovered that the values gotten when multi-paradigm complexity metric was applied on each of the codes are higher than that of eLOC and CC, this is because MCM includes all other factors that affect code complexity neglected by eLOC and CC. It was also found out that the values realised from the application of Halstead method are somehow too exaggerated when compared with those from other existing metrics and the proposed metric.

More so, it was found out that the complexity values gotten for both linear and binary search algorithms using Python programming language is low when compared with that of C++, this is due to the fact that, python is a modern interpreted language with powerful built-in features and a unique indentation feature to shorten coding.

References

[1] DeMarco, T (1986): Controlling Software Projects, Yourdon Press, New York p.217-220.
[2] Eclipse Metrics Plugin 1.3.8 (last accessed 23.02.2010)Available at: http://metrics2.sourceforge.net/
[3] Halstead M. H. (1997) Element of Software Science, Elsevier North- Holland, New York, p.670-672.
[4] International standard organisation (2010), standard for software quality metrics methodology, p1-20.
[5] McCabe(1976) T. McCabe, A complexity measure, IEEE Transactions of Software Engineering, Vol. SE-1, p.308-320.
[6] McCabe (2010) T.J., Watson, A.H. (1994): Software Complexity, McCabe and Associates, p. 14-56. Inc. (last accessed 17.03.2010)
[7] Available at: http://www.stsc.hill.af.mil/crosstalk/1994/12/xt94d12b.asp
[8] Resource Standard Metrics. (last accessed 18.02.2010). Available at http://msquaredtechnologies.com/m2rsm/docs/rsm_metrics_narration.htm
[9] Sommerville, I. (2004) Software Engineering, 7thEdition, Addison Wesley, slides 6-9.
[10] Wang, Y., Shao, J.: A (2003): New Measure of Software Complexity Based on Cognitive Weights.Can. J. Elec. Computer Engineering, p. 1-5.