Comparative Analysis of Software Complexity of Searching Algorithms Using Code Based Metrics

Olabiyisi S.O[1], Omidiora E. O[2] and Sotonwa K. A[3]
Department of Computer Science and Engineering Ladoke Akintola University of Technology P. M.B. 4000, Ogbomoso, Nigeria.

1. soolabiyisi@lautech.edu.ng
2. eoomidiora@lautech.edu.ng
3. kehindesotonwa@yahoo.com

Abstract: Software complexity metrics are used to quantify a variety of software properties. Complexity measures can be used to predict critical information about testability, reliability and maintainability of the software systems from automatic analysis of the source code. In this paper different software complexity metrics were applied to searching algorithms, our intention is to compare software complexity of linear and binary search algorithms, evaluate, rank competitive object oriented applications (Visual Basic, C#, C++ and Java languages) of these two algorithms using code based complexity metrics such as (line of codes, McCabe cylomatic complexity metrics and Halstead complexity metrics) and measured the sample programs using length (in lines) of the program, line of code (LOC) without comments, LOC with comments, McCabe method, the program difficulty using Halstead method. The result revealed that McCabe method has negligible values of complexity for Visual Basic, C#, C++ and Java languages for linear search and similar measuring values for binary search and also from statistical analysis of ANOVA (Analysis of Variance) the result showed that for both linear and binary search techniques, the four (4) languages do not differ significantly, therefore it is concluded that any of the four (4) programming languages is good to code linear and binary search algorithms.

Index Terms – Software metrics, Searching Algorithms, Code Based Metrics, Length in line of the program, LOC with Comments, LOC without Comments, McCabe method and Halstead method.

1. INTRODUCTION

The first problem encountered when attempting to understand program complexity is to define what it means for a program to be complex. Basili defines complexity as a measure of the resources expended by a system while interacting with a piece of software to perform a given task. If the interacting system is a computer, then complexity can be defined by the execution time and storage required to perform the computation. If the interacting system is a programmer complexity is defined by the difficulty of performing tasks such as coding, debugging, testing, or modifying the software. The term software complexity is often applied to the interaction between a program and a programmer working on some programming tasks [1].

Software complexity is defined as "the degree to which a system or component has a design or implementation that is difficult to understand and verify [2] i.e. complexity of a code is directly dependent on the understandability. All the factors that make a program difficult to understand are responsible for its complexity. Software complexity also is an estimate of the amount of effort needed to develop, understand or maintain the code and the more complex the code is the higher the effort and time needed to develop or maintain this code [3]. Results based on real life projects have shown that there is a correlation between the complexity of a system and the number of faults Munson and [4] and [5].

According to Webster's New Dictionary of Synonyms [6]: "Something is complex when it is made up of so many different interrelated or interacting parts of elements that it requires deep study or expert knowledge to deal with it". As stated above, complexity has to do with the amount of resources that is required to perform some activities. The more resources that must be spent to achieve something, the more complex of the entity with respect to this task and the amount of resources used is not a sufficient characteristic for classifying a task as complex which means that complexity and length are different characteristics of the task and the resources needed to accomplish a task is a function of the size of the task and its unit complexity. Of course complexity may increase as a function of the size of the task but this increase should normally be less than increase in size. Since complexity is an attribute with many meanings therefore, complexity also correlates strongly with the length of a program.

Toularkis distinguished between two classes of complexity measures namely, dynamic complexity measure and static complexity measure. Dynamic complexity measure measures the amount of resources consumed during computation and static complexity measure measures the size (e. g program length) or structural complexity (e.g. level of nesting do loop) of an algorithm description [7].

2. LITERATURE REVIEW

2.1 Complexity Concepts

For information system, especially for software, the word complexity was first used for what is called computational or time complexity. As an example the task of searching a sorted list of length 'n' for a single item has complexity O(log n) meaning that any logarithm giving a solution to the task will in the worst case need the order log n pair wise comparisons to solve the task for large 'n'. The task to sort such a list has computational complexity O(nlogn) and is thus a more complex task. These complexities characterize the class of problems to be solved and give the least possible growth in computation times as a function of the growth in problem size. In addition to this each method designed to solve problems belonging to some class has its own complexity which of course cannot be less than the complexity of the corresponding problem class.

Complexity has also been used to characterize software. According to [8] complexity "relates to data set relationship, data structures, data flow and the algorithm being implemented" and "measures the degree of decision making logic within the system. Beizer states that 'using only our intuitive notion of software complexity, we expect that more complex software will cost more to build and test and will have more latent bugs' [9]. Software complexity is defined as the degree of difficulty in analysis, testing, design and implementation of software. Not attempting to attach a single number to software complexity [8]. In [9], Jones discussion on measuring programming complexity identifies 'two logically distinct tasks:

(i) Measuring complexity of the problem, i.e. the functions and data to be programmed;

(ii) Measuring the complexity of the solution of the problem, i.e. the software itself.

Complexity is a metaphysical property and thus not directly measurable that is, it required the linkage behaviour of the product characteristics that are measurable [10]. Banker, Datar and Zweig [11] states that 'software complexity refers to the extent to which a system is difficult to comprehend, modify and test, not to the complexity of the task which the system is meant to perform; two systems equivalent in functionality can differ greatly in their software complexity'. They noticed that most complexity metrics proposed to confound to the complexity of a program with its length. They also propose to measure length-independent complexity metrics by measuring 'density of decision making' and 'density of branching' within a program. In a high correlation of cyclomatic complexity with lines of codes given as reason for proposing a transformed metric 'complexity density' is defined as the ratio of cyclomatic complexity to thousand lines of code [12].

Zuse [6] agreed with Ramamoorthy and Shepperd that the term software complexity is still not well defined. Here the term complexity measure is a misnomer. It deals with the psychological complexity of programs. The overall complexity of software is a function of many factors. In literature we can find many types of measures, for example process measures, product measures, resource measures, static measures, descriptive measure, black-box measures, quality measures, code measures, design measures, inter-intra-modular measures, data flow measures, information flow measures and specification measures. The complexity of a module or a program system is influenced by the factors of cohesion, coupling, decomposition and intra-modular complexity. It can be said that the measurement of complexity is synonymous with determining the degree of difficulty in analyzing, maintaining, testing, designing and modifying software.

The term complexity is commonly used to capture the totality of all internal attributes of software. When people talk of the need to control complexity what they really meant is the need to measure and control a number of internal (structural) product attributes. Fenton stated that 'there appears to be three distinct (orthogonal and fundamental) attributes of the software such as length, functionality and complexity of the underlying problem which the software is solving' [13].

2.2 **Lines of Code (LOC)**

The line of codes (LOC) is generally considered as the count of the lines in the source code of the software. Usually, (LOC) only considers the executable sentence. LOC is independent of what program language used. The LOC evaluates the complexity of the software via the physical length. LOC is based upon two rules:

i. the relationship between the count of code lines and the bug density,

ii. the independence between the bug density and the program language.

Also sometimes, the LOC is estimated by the other factors [14]. The original purpose of its development was to estimate man-hours for a project [15]. Some types of LOC are as follows:

i. Lines of Code (LOC): It is obvious from its name that it counts the number of lines which are uncommented in source code. Some developers write code statement and comment on a same physical line. In such cases this metric can be further defined easily.

ii. Kilo Lines of Code (KLOC): it is LOC divided by 1000.

iii. Effective Lines of Code (ELOC): It only counts the lines that are not commented, blank, standalone braces or parenthesis. In a way this metric presents the actual work performed.

iv. Logical Lines of Code (LLOC): This metric shows the count of logical statements in a program, it only counts the statements which end at semi-colon. This definition of metric is only applicable for languages like C or Java, but for languages like Haskell this metric won't work

v. Multiple Line of codes (MLOC): It contains several separate instructions, multiple line of code like million lines of code.

2.3 **Halstead Complexity Metric**

Halstead introduced the concept of software science and use scientific methods to analyze the characteristics and structure of the software. The idea resulted in the introduction of the Halstead complexity metric (HCM). The HCM is calculated on the count of the operators and operands16]. The operators are symbols used in the expressions to specify the manipulations to be performed. The operands are the basic logic unit to be operated. The HCM measures the logic volume of the software. Firstly, the HCM compute the following parameters:

$\mu 1$= the number of unique operators

$\mu 2$= the number of unique operands

N1 = the total occurrences of operators

N2= the total occurrences of operands

P = the program

From these statements, some indicators can be calculated

The length N of P: $N = N1 + N2$ (2.1)

The vocabulary μ of P: $\mu = \mu 1 + \mu 2$ (2.2)

The volume V of P: $V = N * \log_2(\mu)$ (2.3)

The level L of P: $L = (2 \div \mu 1) * (\mu 2 \div N2)$ (2.4)

The program difficulty D of P: D =

$(\mu 1 \div 2) * (N2 \div \mu 2)$ (2.5)

The effort E to generate P is calculated as:

$E = D * V$ (2.6)

Error Estimate: $B = V/X^*$ (2.7)

Programming Time: $T = E/18$ (2.8)

Number of Delivered Bugs: $B = E^{(2/3)} / 3000$

The V* is the software's ideal volume.

This formula is commonly used: $V^* = (\mu 1 N2 \div 2\mu 2)$ $(N1+N2) \log_2(\mu 1 + \mu 2)$

To estimate the V* the X* means the programmer's ability. Halstead sets X* for a fixed value of 3000.

2.4 **McCabe Cyclomatic Complexity Metric**

Based upon the topological structure of the software, Thomas J. McCabe introduced a software complexity metric named McCabe Cyclomatic Complexity Metric. As described by McCabe, the primary purpose of the measure is to identify software modules that will be difficult to test or maintain [17]. The nodes correspond to the code lines of the software, and a directed edge connects two nodes if the second node might be executed immediately after the first one. If the conditional evaluation expression is composite, the expression is broken down

$$MC = V(G) = e - n + 2p \quad (2.11)$$

where:

V(G) is the cyclomatic complexity

e is the number of edges of the graph

n is the number of nodes of the graph and

p is the number of connected components.

3. MATERIALS AND METHODS

The metrics were applied on searching algorithms codes written Visual Basic, C#, C++ and Java languages. Eight (8) different types of searching algorithms codes were considered. These programs were different in their architecture.

3.1 Evaluation of Software Complexity of Searching Algorithms

To find the complexity of variations of different implementation languages. The following approaches were applied:

i. Length in line of the program: counts every line of the program including comments, standalone brace, blank lines and parenthesis.

ii. LOC without comments: counts line of codes that do not contain comments

iii. LOC with comments: counts line of codes that contain comments.

iv. McCabe method: using cyclomatic complexity method MC = V(G) = e – n + 2p

v. Program difficulty: using Halstead method D of P is D = (μ1 ÷2) * (N2 ÷ μ2)

The evaluation of code based metrics for linear search algorithms and binary search algorithm were given in Tables 1 and 2. It was discovered that visual basic has the lowest value of complexity for all the variations of different implementation except for LOC with comment and McCabe method that has highest values for both linear and binary search algorithms.This is due to the language that lacks a keyword to directly implement one of the steps and the implementation of the step leads to an increase in the number of steps require to implement the algorithm.

Table 1: Comparison of the Metrics for Linear Search Algorithms

Complexities Values					
	Length in Lines	**LOC with Comments**	**LOC without Comments**	**McCabe Methods**	**Program Difficulty**
VB	37	6	29	11	16.1
C#	50	4	44	11	25.4
C++	51	4	42	11	32
Java	54	5	43	9	21.6

Table 2: Comparison of the Metrics for Binary Search Algorithms

Complexities Values					
	Length in Lines	**LOC with Comments**	**LOC without Comments**	**McCabe Methods**	**Program Difficulty**
VB	54	6	46	12	25.8
C#	75	9	63	12	43.9
C++	76	2	68	12	48.1
Java	77	9	62	11	35.5

3.2 Statistical Analysis

Statistical analysis was carried using Analysis of Variance (ANOVA) which is a rigorous statistical tool used in making inferential decisions in experimental design studies to ensure the equivalence of comparative groups even when number per group differed across the group. Therefore statistical analysis carried out using ANOVA at 0.01 levels significant for values obtained for linear search and binary search techniques.

Tables 3 and 4 show the Analysis of Variance (ANOVA) table for linear search and binary search and it is discovered that $F_{0.01,\ 3,\ 12} = 5.95 > F_{calculated}$, since the F_{table} exceeds the $F_{calculated}$for both linear and binary search we accept the null hypothesis Ho, therefore is significant relationship between the metrics and the programming languages for linear and binary search techniques.

Table 3: Analysis of Variance for Linear Search

Sources of Error	**Sum of Squares**	**Variance Estimate**	**DF**	**F**
Between Groups	SS_b (283.23)	S_b^2 (94.41)	3	0.3235
Within Groups	SS_w (3502.55)	S_w^2 (291.88)	12	
Total	3785.78	362.69	15	

Table 4: Analysis of Variance Binary Search

Sources of Error	**Sum of Squares**	**Variance Estimate**	**DF**	**F**
Between Groups	SS_b (646.16)	S_b^2 (215.39)	3	0.3093
Within Groups	SS_w (8355.44)	S_w^2 (696.29)	12	
Total	9001.6	857.83	15	

4. RESULTS AND DISCUSSIONS

4.1 Complexity of Different Implementation of Linear Search

As shown in Figure 1 there are considerable differences among the implementation complexities of the different languages. The Figure shows the comparison between the object oriented languages of Visual Basic, C#, C++ and Java languages by using linear search as a case study for comparison. The Figure reveals that the length (in lines) of the program of Visual Basic, C# and C++ languages are less than that of Java language in this case Visual Basic, C# and C++ have less complexity than Java, the LOC without

comments of Visual Basic, C++ and Java are less than that of C# therefore Visual Basic, C++ and Java have less complexity than C# and the LOC with comments of C#, C++ and Java are less than that of Visual Basic then C#, C++ and Java have less complexity than Visual Basic.

The McCabe method of Java language is less than that of Visual Basic, C# and C++ therefore Java language has less complexity than Visual Basic, C# and C++ because if the implementations are based on the same number of steps and decision points and that is why they have the same value for cyclomatic complexity, the program difficulty using Halstead method for Visual Basic, C# and Java are less than that of C++, therefore Visual Basic, C# and Java have less complexity than C++.

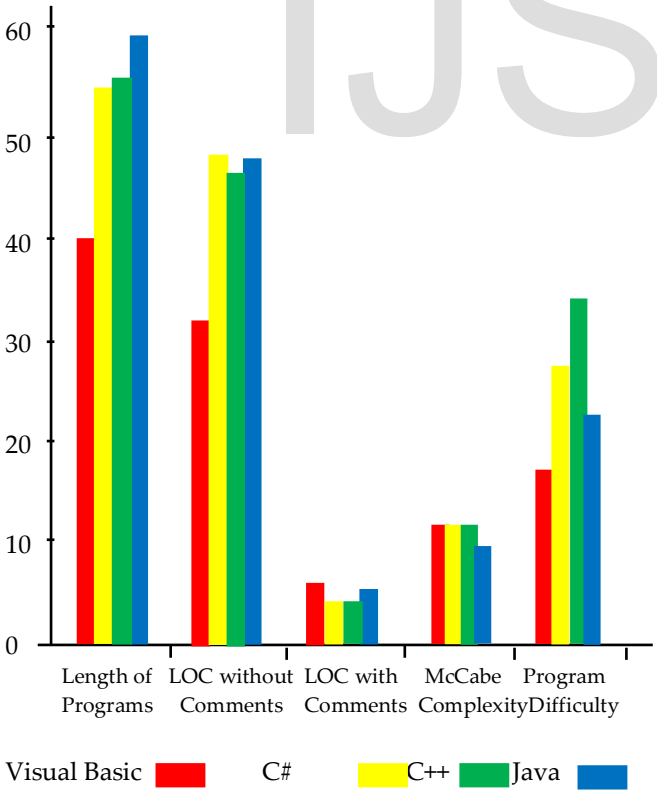

Fig 1: Complexity Comparison between the Object Oriented Languages VB, C#, C++ and Java for Linear Search Algorithm

4.2 Complexity of Different Implementation of Binary Search

As shown in Figure 4.2 there are considerable differences among the implementation complexities of the different languages. The Figure shows the comparison between the object oriented languages of Visual Basic, C#, C++ and Java by using binary search as a case study for comparison. The Figure reveals that the length (in lines) of the program of Visual Basic, C# and C++ languages are less than Java language in this case Visual Basic, C# and C++ have less complexity than Java, the LOC without comments of Visual Basic, C# and Java are less than that of C++ therefore Visual Basic, C# and Java have less complexity than C++ and LOC with comments of Visual Basic and C++ are less than that of C# and Java, then Visual Basic and C++ have less complexity than C# and Java.

The McCabe method of Java is less than that of Visual Basic, C# and C++ because if the implementations are based on the same number of steps or decision points their cyclomatic complexity will be the same that is why Visual Basic, C and C++ have the same McCabe method and that of Java has less complexity, the program difficulty of Visual Basic, C# and Java are less than that of C++, in this case Visual Basic, C# and Java have less complexity than C++.

differences between the values for McCabe complexity are negligible while that of Halstead method show remarkable differences with C++ language is having the highest value and Visual Basic language having the lowest value.

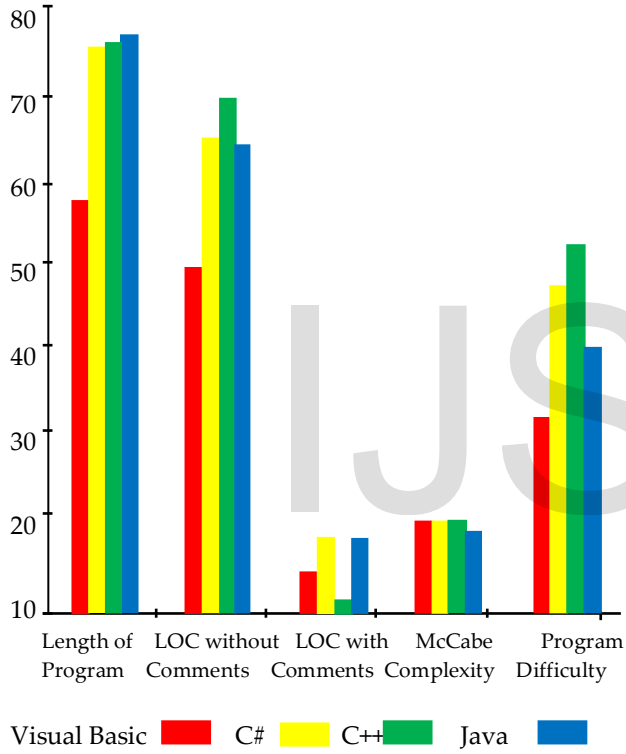

Fig 2: Complexity Comparison between the Object Oriented LanguageVB, C#, C++ and Java for the Binary Search Algorithm

4.3 Comparison between McCabe Method and Halstead Methods for Linear Search

Figures 3 and 4 are showing the differences between McCabe and Halstead measurement for linear search and binary search for the (4) four languages. In Figure 3 it is discovered that the

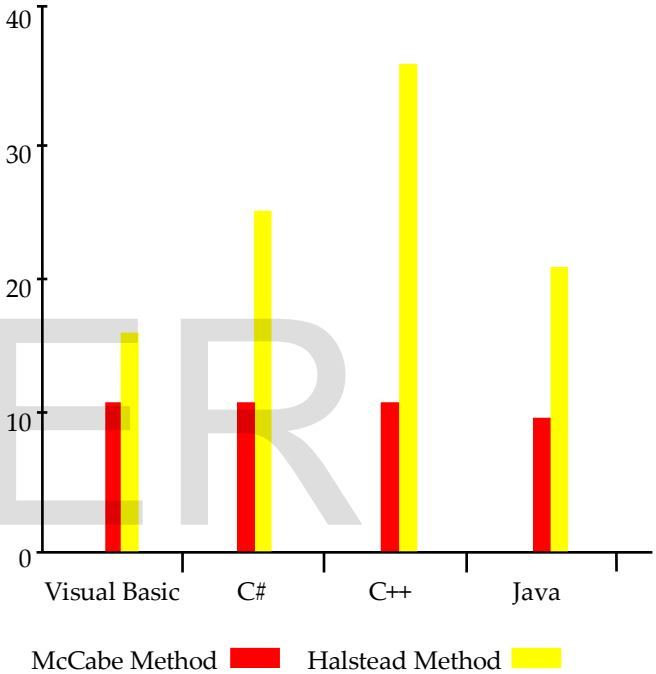

Fig 3: Comparison of McCabe Method and Halstead Method for Linear Search

4.4 Comparison between McCabe Method and Halstead Methods for Binary Search

Also in Figure 4, it is discovered that the differences between the values for McCabe complexity are negligible while that of Halstead method show remarkable differences with C++ language is having the highest value and Visual Basic language having the lowest value.

the program length of Visual Basic, C# and C++ are less than that of Java, in this case it can be predicted that Visual Basic, C# and C++ have less complexity than Java and the complexity are negligible.

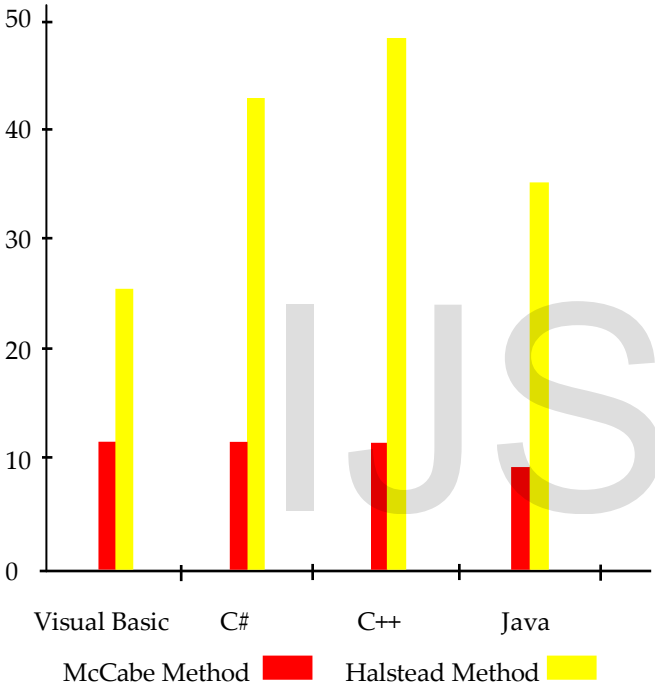

Fig 4: Comparison of McCabe Method and Halstead Method for Binary Search

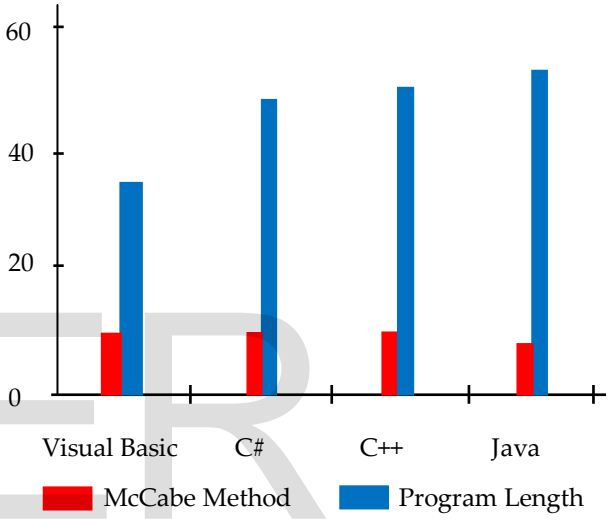

Fig 5: Comparison of Program Length and McCabe Method for Linear Search

4.5 Comparison between Program Length and McCabe Methods for Linear Search

Figures 5 and 6 are showing the differences between program length and McCabe measurement for linear and binary search for the (4) four object oriented languages. In Figure 4.5 it is discovered that

4.6 Comparison between Program Length and McCabe Methods for Binary Search

Also Figure 6 shows that the program length of Visual Basic, C# and C++ are less than Java, therefore languages with less number of keywords expected to be more complex and needs more access time compared with the language that has more keywords which is demonstrated among the four languages in Figure 4.6 therefore this can be used as a second pre indicator for programs complexity.

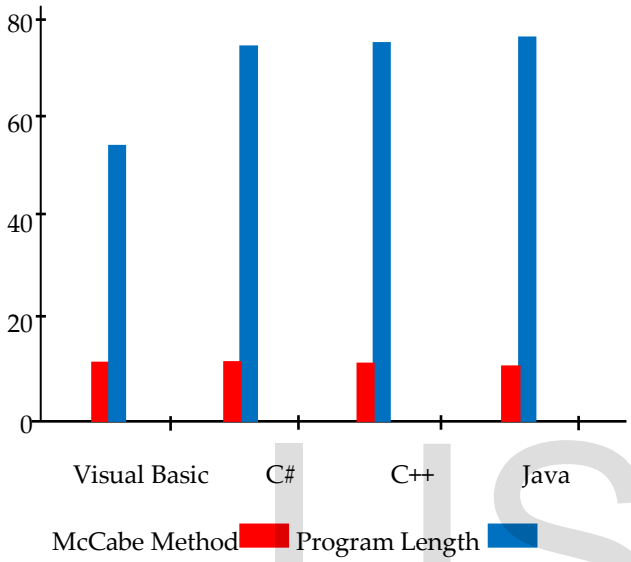

Fig 6: Comparison of Program Length and McCabe Method for Binary Search

5. CONCLUSION

It was found that McCabe method has negligible values of complexity for Visual Basic, C#, C++ and Java for linear search, the value of Java language was nine (9) and similar measuring value for Visual Basic, C#, C++ and Java for binary search, the value of Java language was eleven (11). The measured complexity with McCabe method is higher for Visual Basic, C# and C++ in binary search. Further result from statistical analysis of Analysis of Variance (ANOVA) showed that for both linear and binary search techniques the four (4) languages do not differ significantly. Therefore, it is concluded that any of the four (4) programming languages is good to code linear search and binary search algorithms.

REFERNCES

[1] Basil, V.R (1980): Quantitative Software Complexity Models: A Summary in Tutorial on Models and Methods for Software.

[2] IEEE Standard 1998): Volume: 1998, Issue: IEEE-std-1061-1998, Publisher: IEEE Computer Science: 278-278

[3] Li and Henry (1993): Object Oriented Metrics that Predict Maintainability, Journal of Systems and Software, 23(2): 111-122

[4] Munson J. and Khoshgoftaar T. (1996): Software Metrics for Reliability Assessment", in Handbook of Software Reliability Engineering, Michael Lyu (edt.), McGraw-Hill, Chapter 12: 493-529.

[5] Ammar H. H., Nikzadeh T. and Dugan J. (1997): A Methodology for Risk Assessment of Functional Specification of Software Systems Using Colored Petri Nets", Proc. Of the Fourth International Software Metrics Symposium, Metrics'97, Albuquerque, New Mexico: 108-117.

[6] Zuse H. (1991): Software Complexity: Measures and Methods: 605, 498figures. Berlin, New York: DeGruyter.

[7] Tourlakis G. J. (1984): Computability, Reston, Virginia. 12: 39-42.

[8] McCall J.A., Richards P.K. and Walters G.F. (1977): Factors in Software Quality, I-III, Rome Air Development Centre, Italy.

[9] Beizer B. (1984): Software System Testing and Quality Assurance, Van Nostrand Reinhold, New York.

[10] Ramamoorth, C.V., Tsai W.T., Yamura T. and Bhide A. (1985): Metrics Guided Methodology, COMPSAC 85: 111-120.

[11] Jones C. (1986): Programming Productivity, McGraw Hill, New York.

[12] Shepperd M. (1988): An Evaluation of Software Product Metrics, information and Software Technology, 30(3): 177-188.

[13] Banker R.D., Datar, S.M. and Zweig D. (1989): Software Complexity and Maintainability CiteSeer Scientific Literature Digital Library and Search Engine.

[14] Gill G.K. and Kemerer C.F. (1991): Cyclomatic Complexity Density and Software Maintenance, IEEE Trans. Software Engineering, 17: 1284-1288.

[15] Fenton N. E. (1992): Software Metrics – A Rigorous Approach, Chapman & Hall, London Computer Journal 29(4), 330 - 340.

[16] NASA (2008): Repository Overview, http://mdp.ivv.nasa.gov/repository.html.

[17] Milutin A. (2009): "Software code metrics", (Online: accessed on 2010-06-21 from Introduction to Algorithms).

[18] Halstead M. H. (1977): Elements of Software Science, Operating and Programming Systems Series, Elservier Computer Science Library North Holland N. Y. Elsevier North-Holland, Inc. ISBN 0-444-00205-7.

[19] McCabe T. (1976): "A Complexity Measure". IEEE Transactions on Software Engineering, 1: 312-327.