International Research Journal of Computer Science (IRJCS)
Issue 04, Volume 6 (April 2019)
ISSN: 2393-9842
www.irjcs.com

OBJECT ORIENTED PROGRAMMING LANGUAGES FOR SEARCH ALGORITHMS IN SOFTWARE COMPLEXITY METRICS

Kehinde A. Sotonwa
Computer Science and information Technology,
Bells University of Technology,Ota, Nigeria
kasotonwa@bellstech.edu.ng

Monsurat O. Balogun
Electrical and Computer Engineering,
Kwara State University, Malete, Nigeria
monsurat.balogun@kwasu.edu.ng

Esther O. Isola
Information and Communication Technology,
Osun State University, Osogbo, Nigeria
esther.isola@uniosun.edu.ng

Stephen O. Olabiyisi
Computer Science and Engineering,
Ladoke Akintola University of Technology,Nigeria
soolabiyisi@lautech.edu.ng

Elijah O. Omidiora
Computer Science and Engineering,
Ladoke Akintola University of Technology,Nigeria
eoomidiora@lautech.edu.ng

Christopher A. Oyeleye
Computer Science and Engineering,
Ladoke Akintola University of Technology, Nigeria
caoyeleye@lautech.edu.ng

Manuscript History
Number: **IRJCS/RS/Vol.06/Issue04/APCS10082**
Received: 24, March 2019
Final Correction: 02, April 2019
Final Accepted: 12, April 2019
Published: April 2019
Citation: Sotonwa, Balogun, Isola, Olabiyisi, Omidiora & Oyeleye (2019). OBJECT ORIENTED PROGRAMMING LANGUAGES FOR SEARCH ALGORITHMS IN SOFTWARE COMPLEXITY METRICS. IRJCS:: International Research Journal of Computer Science, Volume VI, 90-103. **doi://10.26562/IRJCS.2019.APCS10082**
Editor: Dr.A.Arul L.S, Chief Editor, IRJCS, AM Publications, India

Abstract— 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 match vigorous object oriented applications (Visual Basic, C#, C++, Java and Python languages) of linear and binary search algorithms using software complexity metrics such as (Line of Codes (LOC), McCabe cylomatic complexity metrics and Halstead complexity metrics) and measured the sample program using LOC, McCabe method, the program difficulty using Halstead method and degree of relationship using Pearson correlation Coefficient and Statistical Analysis of Variance (ANOVA) for these three metrics. The five (5) languages correlate and do not differ significantly, then any of the five (5) programming languages is good to code linear and binary search algorithms.

Keywords — Software metrics (Line of Codes (LOC)**;** McCabe method and Halstead method)**;** Searching Algorithms**;** Object oriented programming languages (VB, C#, C++, Java and Python)**;**

I. INTRODUCTION

Software development involves creating a software system depending on requirements. Requirements are usually complex and this situation makes software projects change continuously. Software projects are changed or modified in order to better understand the user requirements or eliminate errors. Hence, software systems are called to be complex [1][2][3][4][5]. Software life cycle is the process of developing and changing software systems. A software life cycle consists of all the activities and products that are needed to develop a software system.

Due to the fact that software systems are complex, life cycle models tend to enable developers to cope with software complexity. Life cycle models expose the software development activities and their dependencies in order to make them more visible and manageable [6][7][8][9]. A programming language is an artificial language designed to communicate instructions to a machine, particularly a computer. Programming languages can be used to create programs that control the behaviour of a machine and/or to express algorithms precisely, example include procedural programming, object oriented programming and multi paradigm programming language but this work is focused on object oriented languages [10][11][12]. A computer program is a set of instructions developed to perform a task, whereas a software system is a set of programs developed with an engineering discipline under consideration of quality with an aim to accomplish many tasks properly. The main distinguishing factor is quality [2]. Thereby, quality is the indispensable fact of a software system.

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 [13][14]. 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 [15] 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 [16]. Results based on real life projects have shown that there is a correlation between the complexity of a system and the number of faults [17][18].

II. LITERATURE REVIEW

Over the last three decades many measures have been proposed by researchers to analyze software complexity, understandability, and maintenance. Metrics designed to analyze software such as imperative, procedural, and object-oriented programs. Software measurement is concerned with deriving a numeric value for an attribute of a software product, i.e. a measurement is a mapping from the empirical world to the formal world [19][20].

I. Existing Software Complexity Metrics

Software metrics have been found to be useful in reducing software maintenance costs by assigning a numeric value to reflect the ease or difficulty with which a program module may be understood. There are hundreds of software complexity measures that have been described and published. For example, the most basic complexity measure the number of lines of code (LOC), simply counts the lines of executable code, data declarations, comments, and so on. While this measure is extremely simple, it has been shown to be very useful and correlates well with the number of errors in programs. Reference[21] formulated a metric that counts the number of lines of codes but neglected the intelligence content, layout and other factors that affect the complexity of the code [19][20][21][22].

McCabe measured metric is based on program size but more on information/control flow which is based on specification flow graph representation and the complexity of an algorithm that goes beyond the algorithm measure of the cognitive aspect. But the metric was not sufficient to measure code complexity especially of modern programming languages [23]. Yas used McCabe method of software complexity to compare the complexity of several object oriented languages such as C++, Java and Visual Basic. The binary search algorithm written in three languages; C++, Java, and Visual Basic was used to measure the complexity of the algorithm. He did not consider Halstead method with the algorithm and so also linear search [20]. Halstead measured the number of operators, the number of operands and their respective occurrence in the program (code). These operators and operands are to be considered during calculation of Program Length, Vocabulary, Volume, Potential Volume, Estimated Program Length, Program Difficulty, Effort and Program Time. Some questions were aroused such as what was an operator and operand. It was difficult to differentiate them therefore there is no a sharp distinction between operators and operands. Another weakness of this complexity is that they do not measure control flow complexity and also difficult to compute [24].

Sotonwa *et al.* compared software complexity of searching algorithms using code based metrics such as LOC, McCabe method and Halstead method. The metric was more realistic because it compared four object oriented languages with searching algorithms of linear and binary search. Also statistical analysis of variance was conducted on the metrics to show that the four (4) objects oriented programming languages were good to code linear and binary search algorithms [25].

II. Object Oriented Programming Languages (OOPL)

OOP is a type of programming in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure. OOPL proves to be faster to learn and easier to maintain. In addition, OOPL provide a way to break large and sometimes difficult-to-manage programming projects into smaller modules that can be managed easily. Examples are Java, C++, C#, Python, Net, and Visual Basic.Net etc [25][26]. Each OOPL has a number of keywords; sometimes these keywords are different from one language to the other. There are a large number of similarities between Java, C#, C++ and Visual Basic. The Table below show the comparison between these keywords, Tables 1 shows that there are 35 keywords with similar identification in both languages C# and Java. There are 15 keywords having different identification from both languages for example base in C# language is represented with super in Java language, Table 2 shows the similar identification in C++ and Visual Basic and Table 3 showed keyword for Python [27].

III. Search Algorithm

A search algorithm is an algorithm for finding an item with specified properties among a collection of items [28]. Common types of search algorithm are linear search, binary search, interpolative search and fibonacci search but the scope f this study is specializing on only linear search algorithm and binary search algorithm

A. Linear search*:* is a search algorithm that is suitable for searching a list of data for a particular value that consists of checking every one of its elements, one at a time and in sequence until the desired one is found.[28][29]. In linear search list does not have to be sorted which make it very efficient when used for small database because It requires small memory space and not expensive

B.Binary search: is an algorithm for locating the position of an element in a sorted list: if equal to the sought value, then the position is found; otherwise the upper half or lower half is chosen for further searching.[28][30]. Binary search utilizes the divide and conquer approach which made it more efficient and very fast than linear search approach.

Table 1: C# and Java Keywords

C# Keyword	Java Keyword	C# Keyword	Java Keyword	C# Keyword	Java Keyword	C# Keyword	Java Keyword
Abstract	abstract	Explicit	n/a	Object	n/a	This	This
As	n/a	Extern	Native	Operator	n/a	throw	Throw
Base	super	Finally	Finally	Out	n/a	True	True
Bool	boolean	Fixed	n/a	Override	n/a	Try	Try
Break	break	Float	Float	Params	n/a	typeof	n/a
Byte	n/a	For	For	Private	private	Unit	n/a
Case	case	Foreach	n/a	Protected	n/a	ulong	n/a
Catch	catch	Get	n/a	Public	Public	unchecked	n/a
Char	char	Goto	Goto	Readonly	n/a	unsafe	n/a
Checked	n/a	If	If	Ref	n/a	unshort	n/a
Class	class	Implicit	n/a	Return	Return	using	Import
Const	const	In	n/a	Sbyte	Byte	value	n/a
Continue	continue	Int	Int	Sealed	Final	virtual	n/a
Decimal	n/a	Interface	Interface	Set	n/a	void	Void
Default	default	Internal	Protected	Short	Short	volatile	Volatile
Delegate	n/a	Is	Instanceof	Sizeof	n/a	While	While
Do	do	Lock	synchronized	Stackalloc	n/a	:	Extends
Double	double	Long	Long	Static	Static	:	Implements
Else	else	Namespace	Package	String	n/a	n/a	Strictfp
Enum	n/a	New	New	Struct	n/a	n/a	Throws
Event	n/a	Null	Null	Switch	Switch	n/a	Transient

Table 2: C++ and Visual Basic Keywords

C++ Keyword	VB keyword	C++ Keyword	VB Keyword	C++ Keyword	VB Keyword	C++ Keyword	VB Keyword
Alignas	alias	Decitype	Decimal	Namespace	Namespace	struct	structure
Alignof	ansi	Default	Default	New	New	switch	switch
And	and	Delete	delete setting	Noexcept	n/a	template	n/a
and eg	and also	Do	Do	Not	Not	this	then
Asm	anc	Double	Double	not eg	Nothing	threadlocal	n/a

Auto	auto	dynamic cast	Dim	Nullptr	n/a	throw	throw
Bitand	n/a	Else	Else	Operator	n/a	true	true
Bitor	n/a	Enum	Enum	Or	Or	try	try
Bool	boolean	Explicit	Erase	or eg	Orelse	typedef	n/a
Break	n/a	Export	n/a	Private	Private	typeid	n/a
Case	case	Extern	n/a	Protected	Protected	typenaame	typenam e
Catch	catch	False	False	Public	Public	union	unicode
Char	char	Float	n/a	Register	n/a	unsigned	unlock
char16t	cshort	For	For	reinterpret cast	n/a	using	until
char32t	csng	Friend	Friend	Return	Return	virtual	virtual
Class	class	Goto	Goto	Short	Short	void	void
Compl	n/a	If	If	Signed	n/a	volatile	volatile
Const	const	Inline	n/a	Sizeof	Synlock	wchar	void
Constexpr	cstr	Int	Int	Static	Static	while	while
const cast	n/a	Long	Long	static assert	n/a	xor	xor
Continue	n/a	Mutable	Mustinherit	static cast	n/a	xor eg	n/a

Table 3: Python Keywords

And	except	Lambda	With	As	finally	nonlocal	while
Assert	false	None	Yield	Break	for	for	not
Print	class	From	Or	Exec	continue	global	pass
Def	if	Raise	Del	Import	return	elif	in
True	else	Is	Try				

III. MATERIAL AND METHODS

The metrics were applied on searching algorithms codes written Visual Basic, C#, C++, Java and Python languages. Ten (10) different types of searching algorithms codes were considered. These programs were different in their architecture. Sample of the program are given in C# linear and binary search with their flow graph Figure 1-4 respectively.

I. Evaluation of Software Complexity of Searching Algorithms

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

A. Line of codes: counts every line of the program including comments, standalone brace, blank lines and parenthesis:

$$\text{LOC} \qquad \text{(i)}$$

B. McCabe method: using cyclomatic complexity method

$$MC = V(G) = e - n + 2p \qquad \text{(ii)}$$

where e is the edges,
n is the nodes and
p is the connected component.

C. Program Difficulty using Halstead method:

$$D \text{ of } P = (\mu_1 \div 2) * (N_2 \div \mu_2) \qquad \text{(iii)}$$

where $\mu_1$is the number of unique operator,
μ_2 is the number of unique operands and
N_2 is the occurrences of operands.

II. Analysis of the Program for Linear Search in C#

i. LOC = 75
ii. McCabe method we have e = 49, n = 41 and p = 2
McCabe MC = V (G) = e – n + 2p = 49 – 41 + 2(2) = 12
iii. Halstead method where $\mu_1 = 27$, $\mu_2 = 24$, $N_1 = 82$ and $N_2 = 78$
The Program Difficulty = $(\mu_1 \div 2) * (N_2 \div \mu_2) = (27 \div 2) * (78 \div 24) = 43.88$

```
1.  // Linear Search in C#
2
3    using System;
4        using System.Collections.Generic;
```



```
15      {
16          x[counter] = a.Next(counter * 10);

17          Console.WriteLine(x[counter]);
18      }
19      //display
20      Console. Clear();//clean of everything written on the dos screen
21      for (int counter = 0; counter <x.Length; counter++)
22      {
23          if ((counter % 10) == 0)
24          {

25              Console.CursorTop = 0;
26          }
27          Console.CursorLeft = 5 * (int)Math.Floor((counter / 10.0));
28          Console.WriteLine(x[counter]);
29                      }
30                      //search
31                      Console.WriteLine("Type in the number you are searching for: ");
5   using System.Text;
6
7   internal static class modSearch
8   {
9     private static int[] x = new int[100];
10    public static void Main()
11    {
12      Console.Clear();
13      Random a = new Random();
14      for (int counter = 0; counter <x.Length; counter++)
32      Linear_Search(System.Convert.ToInt32(Console.ReadLine()));
33    }
34    private static void Linear_Search(int key)
35    {
36                      for (int i = 0; i<x.Length; i++)
37                      {
38          if (key == x[i])
39          {
40              Console.WriteLine("Found at position " + (i + 1));
41              Console.WriteLine("--- Press any key to exit ---");
42              Console.ReadKey();
43              return;
44          }
45      }
46      Console.WriteLine("key is not found");//let the user knows that the number, he is looking for is not there
47      Console.WriteLine("--- Press any key to exit ---");
48      Console.ReadKey();
49    }//End Program
50                      }//End Class
```

Figure 1: The Program for Linear Search Algorithm written in C#

III. Analysis of the Program for Binary Search

i. LOC = 76

ii. McCabe method where e = 49, n = 41 and p = 2
McCabe MC = V (G) = e – n + 2p = 49 – 41 + 2(2) = 12

iiii. Halstead method where $\mu_1 = 29$, $\mu_2 = 25$, $N_1 = 93$ and $N_2 = 83$
The Program Difficulty = $(\mu_1 \div 2) * (N_2 \div \mu_2) = (29 \div 2) * (83 \div 25) = 48.14$

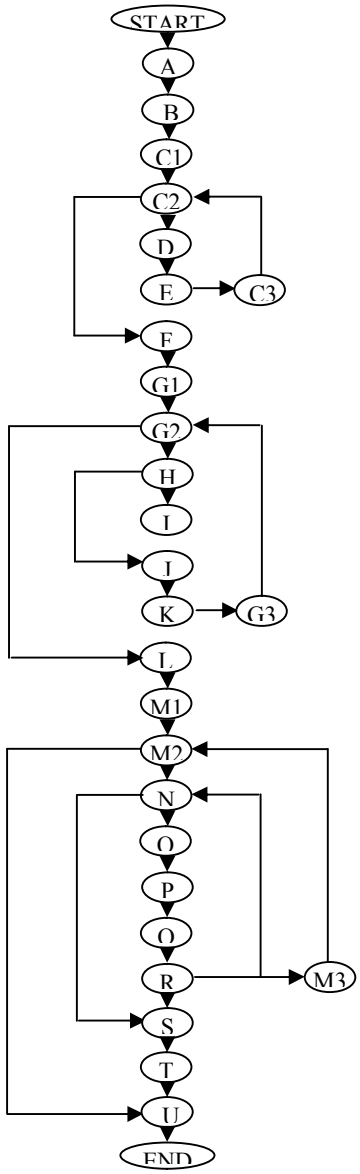


Figure 2: Linear Search in C# Flow Graph

```
1   // Binary Search in C#
2   using System;
4   using System.Collections.Generic;
5   using System.Text;
6   internal static class modSearch
8   {
9      private static int[] x = new int[100];
10     public static void Main()
11     {
12        int temp;
13        Console.Clear();
14        Random a = new Random();
15        for (int counter = 0; counter <x.Length; counter++)
16        {
17              x[counter] = a.Next(counter * 10);
18              Console.WriteLine(x[counter]);
19        }
20        //sort
```

```
21      for (int i = 0; i<x.Length; i++)
22      {
23          for (int j = i; j <x.Length; j++)
24          {
25              if (x[j] < x[i])
26              {
27                  temp = x[i]; //store x(i) somewhere
28                  x[i] = x[j]; //then store x(j) into x(i)
29                  x[j] = temp; //then store temp into x(j)
30              }
31          }
32      }
33      //display
34      Console.Clear();
35      for (int counter = 0; counter <x.Length; counter++)
36      {
37          if ((counter % 10) == 0)
38          {
39              Console.CursorTop = 0;
40          }
41          Console.CursorLeft = 5 * (int)Math.Floor((counter / 10.0));
42          Console.WriteLine(x[counter]);
43      }
44      //search
45      Console.WriteLine("Type in the number you are searching for: ");
46      Binary_Search(System.Convert.ToInt32(Console.ReadLine()), 0, x.Length);
47      }
49    private static void Binary_Search(int key, int left, int right)
50    {
51      int m = 0;
52      while (left <= right)// then left number is the first on the list, the right is the last on the list
53      {
54          m = (left + right) / 2;//calculate the middle position
55          if (key == x[m]) // is the number at the middle position the number we're looking for?
56          {//yse
57              Console.WriteLine("Found at position " + (m + 1));
58              Console.WriteLine("--- Press any key to exit ---");
59              Console.ReadKey();
60              return;
61          } //no ok, is it bigger than the number we are looking for
62          else if (key > x[m])
63          {
64              left = m + 1;
65          }//no, ok, is it smaller than the number we are looking for
66          else if (key < x[m])
67          {
68              right = m - 1;
69          }
70      }
71      Console.WriteLine("key is not found");

72      Console.WriteLine("--- Press any key to exit ---");
73      Console.ReadKey();
74    }
75  }
```

Figure 3: The Program for Binary Search Algorithm written in C#

The evaluation of software complexity metrics for linear search algorithms and binary search algorithm using object oriented languages (Visual Basic, C#, C++, Java and Python) were given in table 3.1 and 3.2. These show the complexity values for all the variations of different.

Table 4: Comparison of the Metrics for Linear Search Algorithms

Object Oriented Languages	Line of Codes	McCabe Methods	Program Difficulty
VB	37	11	16.05
C#	50	11	26.83
C++	51	11	31.97
Java	54	9	21.59
Python	24	5	28.4

Table 5: Comparison of the Metrics for Binary Search Algorithms

Object Oriented Languages	Line of Codes	McCabe Methods	Program Difficulty
VB	54	12	25.77
C#	75	12	43.88
C++	76	12	48.14
Java	77	11	35.8
Python	31	6	38.1

IV. Statistical Evaluation of Pearson Correlation Coefficient

The correlations coefficient is a statistical measure that measures the relationship between two variables. If one variable is changing its value then the value of second variable can be predicted. The positive correlation exists when a high value of one variable is associated with high value of another variable and the negative correlation exists when a high value is associated with low value. The value close to +1 means positive correlation exists, -1 means negative correlation exists and 0 means there is no correlation at all.

Table 6: Pearson Correlation Coefficient for Linear Search

		Line of Codes	McCabe Methods	Program Difficulty
Line of Codes	Pearson Correlation	1	.715	.065
	Sig. (2-tailed)		.175	.918
	N	5	5	5
McCabe Methods	Pearson Correlation	.715	1	-.213
	Sig. (2-tailed)	.175		.731
	N	5	5	5
Program Difficulty	Pearson Correlation	.065	-.213	1
	Sig. (2-tailed)	.918	.731	
	N	5	5	5

Table 7: Pearson Correlation Coefficient for Binary Search

		Line of Codes	McCabe Methods	Halstead Method
Line of Codes	Pearson Correlation	1	.836	.407
	Sig. (2-tailed)		.077	.496
	N	5	5	5
McCabe Methods	Pearson Correlation	.836	1	.045
	Sig. (2-tailed)	.077		.943
	N	5	5	5
Halstead Method	Pearson Correlation	.407	.045	1
	Sig. (2-tailed)	.496	.943	
	N	5	5	5

Table 6 and 7 showed the correlation values for LOC, McCabe and Halstead methods for linear and binary search, the correlation values between LOC and McCabe method for linear and binary search are 0.715 and 0.836 indicated strong correlation exist between LOC and McCabe method and these yielded the computed probability values (p) of 0.175 and 0.77 > 0.05. Therefore P accepts H_0. The correlation values between LOC and Halstead method are 0.65 and 0.407 yielded showed moderate and weak correlation existence and the P-value of 0.918 and 0.496 > 0.05 therefore P value accepts H_0 also.

Finally, the correlation values between McCabe and Halstead methods are -0.213 and 0.45 showed negative and weak correlation between the two measures, the 2-tailed values of 0.713 and 0.943? 0.005.

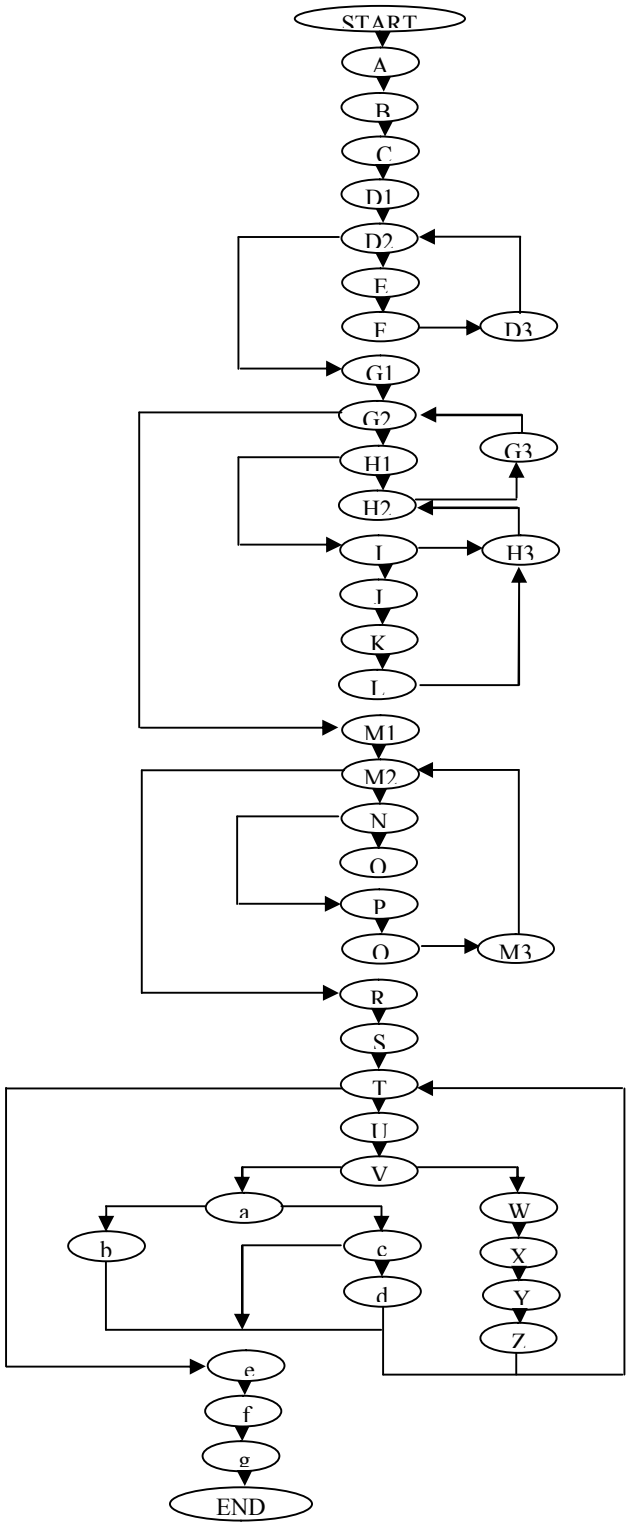


Figure 4: Binary Search in C# Flow Graph

Hence, P-value accepts H_0. The Pearson correlation coefficient showed correlation existence and non significant relationship among the measures and the programming languages for both linear and binary search techniques.

V. Statistical Evaluation of Statistical Analysis of Variance

Statistical Analysis of Variance (ANOVA) is a statistical method used to test differences between two or more means. Table 7 and 8 show the Analysis of Variance (ANOVA) table for linear search and binary search and it is discovered that $F_{0.01, 2, 12}$= 6.93 < $F_{calculated}$, since the F_{table} less than $F_{calculated}$ for both linear and binary search then p value reject the null hypothesis H_0, therefore is significant relationship between the metrics and the programming languages for linear and binary search techniques.

Table 7: Statistical Analysis of Variance for Linear Search

Sources of Error	Sum of Squares	Variance Estimate	DF	F
Between Groups	SS_b (2862)	S_b^2 (1431)	3	22.62
Within Groups	SS_w (759.21)	S_w^2 (63.27)	12	$F_{0.01, 3, 8}$ = 6.93
Total	3621.21	1494.27	15	

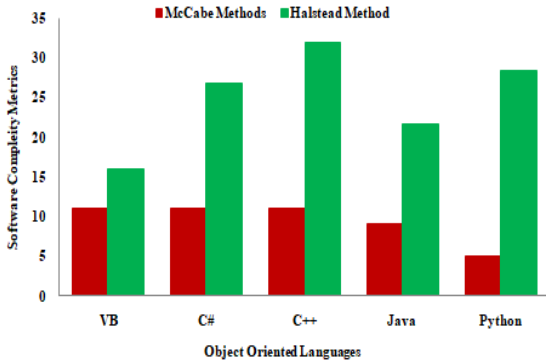
Table 8: Statistical Analysis of Variance Binary Search

Sources of Error	Sum of Squares	Variance Estimate	DF	F
Between Groups	SS_b (6770.09)	S_b^2 (3385.05)	3	20.81
Within Groups	SS_w (1951.8)	S_w^2 (162.65)	12	$F_{0.01, 2, 12}$ = 6.93
Total	8721.89	3547.7	15	

VI. RESULTS AND DISCUSSIONS

I. Comparison between McCabe and Halstead Methods for Linear and Binary Search

The Figure 1 and 2 are showing the differences between McCabe and Halstead measurement for linear search and binary search for the (4) four languages. 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 VB language having the lowest value for linear and binary search.



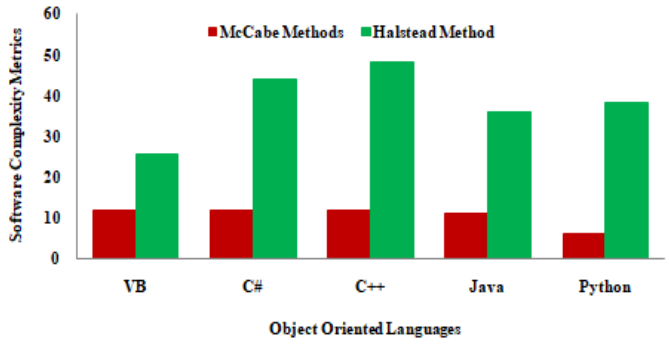
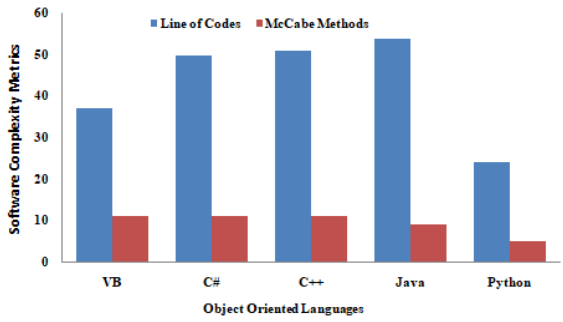


Figure 1: Comparison of McCabe and Halstead Methods for Linear Search`

Figure 2: Comparison of McCabe and Halstead Methods for Binary Search

II. Comparison between Line of Codes (LOC) and McCabe Methods for Linear and Binary Search

Figure 3 and 4 are showing the differences between line of codes and McCabe method for linear and binary search for the (4) four object oriented languages.



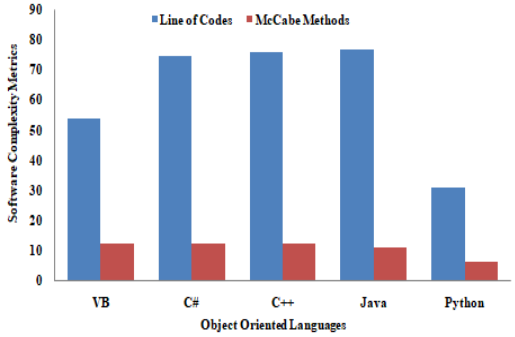


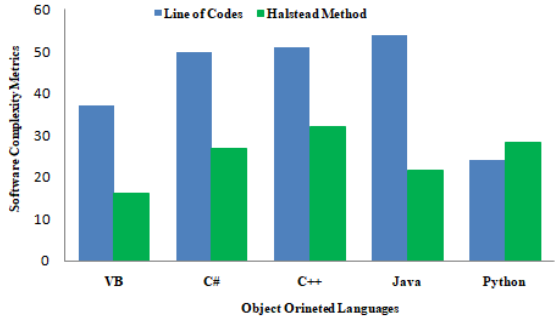
Figure 3: Comparison of LOC and McCabe Method for Linear Search

Figure 4: Comparison of LOC and McCabe Method for Binary Search

It is discovered that the line of codes for 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 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, therefore this can be used as a second pre indicator for programs complexity.

III. Comparison between LOC and Halstead Methods for Linear and Binary Search

Figure 5 and 6 show difference between line of codes and Halstead method for linear and binary search. The results show that Java has the highest complexity value for line of codes and C++ for Halstead method for both linear and binary search. This is because if one language lacks a keyword to directly implement one of the steps, implementations of more step leads to an increase in the number of steps required implementing the algorithm.



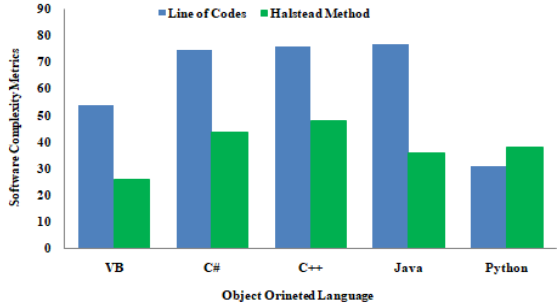


Figure 5: Comparison of LOC and Halstead Method for Linear Search
Figure 6: Comparison of LOC and Halstead Method for Binary Search

VII. CONCLUSION

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. It was found that McCabe method has negligible values of complexity for Visual Basic, C#, C++, Java and Python with fewer values for linear search and a bit higher similar measuring value for binary search. Halstead methods show remarkable difference among the object oriented programming languages (VB, C#, C++, Java and Python) for both linear and binary search techniques while Java language has highest complexity values and VB lowest values for LOC. This is because in LOC some codes have more lines than others because object oriented programming languages provide a way to break large and sometimes difficult to manage programming projects into smaller modules that can be managed easily therefore languages with more number of keywords expected to be less complex and needs little access time compared with the language that has less keywords. Further result from statistical evaluation showed that there is correlation existence between the measures and the programming languages while ANOVA showed that for both linear and binary search techniques the five (5) languages do not differ significantly. Therefore, it is concluded that any of the five (5) programming languages is good to code linear search and binary search algorithms.

ACKNOWLEDGMENT

The authors wish to thank the authority of Bells University of Technology, Ota, Ogun State and Kwara State University Malete, Kwara State, Osun State University, Osogbo, Osun State and Ladoke Akintola University of Technology, Ogbomoso, Oyo State for enabling environment provided as well as uninterrupted internet services. The moral support of member of faculty is also appreciated.

REFERENCES

1. Roger S. P. (2005): Software Engineering – A Practitioner's Approach, 6th Edition McGraw-Hill.
2. Bruegge B. and Dutoit A.H. (2004): Object-Oriented Software Engineering – Using UML, Patterns, and Java, 2nd International Edition, Prentice Hall.
3. Pfleeger S.L. and Atlee J.M (2006): Software Engineering – Theory and Practice, 3rd International Edition, Prentice Hall.
4. Aprna Tripathy, Dharmender Singhkushwaha and Arun Kumar (2012): Software Change Complexity: A New Dimension for Analysing Requested Change: IJCA proceeding on International Confeence on Recent Trends in Information Technology and Computer Science 2012.
5. Edward B. Allen, Sampath Gottipai and Raji Govindarjan (2007): Measuring Size, Complexity and Coupling Hyper Graph Abstractions of Software: An Information Theory Approach, Published in Journal Software Quality Journal Volume 15, Issue 2, June 207, pages 179-212
6. Bruegge Bernd and Allen H. Dutoit (2010): Object Oriented Software Engineering using UML Patterns and Java, 3rd Edition ed Published by Boston Prentice Hall 2010 xxx111 778p: ISBN: 0136061257, 9780136061250, Dewey No. 004.
7. Bruegge Bernd, Stephan Krusche and Martin Wagners (2012): Teaching Tornado: from Communication Model to Release. In Proceedings of the 8th Edition of the Educations Symposium ACM Imsbruck, Austria 5-12 pages.

8. Bruegge Bernd, Stephan Krusche and Lukas Alperowit (2014): Tutorial: How to Run a Multi Customer Software Engineering Capstone Course. In 11th International Conference on Model Driven Engineering Languages and Systems, Valencia, Spain (Sept. 2014) Retrieved Feb. 26th 2015.
9. Charles P. Pfleeger, Shari Lawrence Pfleeger and Willis H. Ware (2006): Security in Computing 4th Edition Published by Prentice Hall Oct. 27th 2006 ISBN-10: 0132390779.
10. Fenton N. E. (1992): Software Metrics – A Rigorous Approach, Chapman & Hall, London Computer Journal 29(4), 330 - 340.
11. Aaby and Anthony (2004): Introduction to Programming Languages. Walla Walla College 204 S. College Ave. College Place, WA 99324 E-mail: aabyan@wwc.edu
12. Maclennun Bruce (1987): Principles of Programming Languages. Oxford University Press:1. ISBN 0-19-511306-3.
13. Basil, V.R (1980): Quantitative Software Complexity Models: A Summary in Tutorial on Models and Methods for Software.
14. Cafer Ferid, İbrahim Akman and Sanjay Misra (2010): Estimating Complexity of a Source Code. The Graduate School of Natural and Applied Sciences, Software Engineering Department, Atılım University, Ankara, Turkey.
15. IEEE Standard 1998): Volume: 1998, Issue: IEEE-std-1061-1998, Publisher: IEEE Computer Science: 278-278
16. Li and Henry (1993): Object Oriented Metrics that Predict Maintainability, Journal of Systems and Software, 23(2): 111-122.
17. 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.
18. 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.
19. Cardoso (2006): Complexity Analysis of BPEL Web Processes, Improvement and Practice Journal, John Wiley and Sons.
20. Yas A. (2009): Using McCabe to Compare the Complexity of Object Oriented Language Arabian Gulf University (AGU), Manama Bahraain, Manama, Bahrain 26671 IJCSNS International Journal of Computer Science and Network Security, VOL. 9 No. 3, March 2009.
21. Milutin, A. (2009): Software Code Metrics, (Online: accessed on 2010-06-21 from Introduction to Algorithms).
22. Amit K. J. and Kumar R. (2014): A New Cognitive Approach to Measure the Complexity of Software. International Journal of Software Engineering and its Applications Volume 8, No. 7 pp. 185-198.
23. 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.
24. McCabe T. (1976): "A Complexity Measure". IEEE Transactions on Software Engineering, 1: 312-327.
25. Sotonwa K. A., Olabiyisi S. O and Omidiora E. O (2013): Comparative Analysis of Software Complexity of Searching Algorithms Using Code Based Metrics International Journal of Scientific & Engineering Research, Volume 4, Issue 6, June-2013 2983 ISSN 2229-5518
26. Norman Fenton (2014): Software Metrics: Success, Failure and Directives 2014 PSRC-funded project IMPRESS, and the ESPRIT-funded projects DEVA and SERENE.
27. Aaby and Anthony (2004): Introduction to Programming Languages. Walla Walla College 204 S. College Ave. College Place, WA 99324 E-mail: aabyan@wwc.edu
28. Donald K., (1997): The Art of Computer Programming. 3: Sorting and Searching (3rd ed.) Addison Wesley pp. 396-408 ISBN 0-201-89685-0.
29. Kruse R. L., (1999): Data Structure and Program Design in C++ Prentice-Hall, ISBN 0-13-768995-0: 280.
30. Netty van Gasteren, and Wim Feijen, (1995): The Binary Search Revisited, AvG127/WF214, 1995. (Investigates the Foundations of the Binary Search, Debunking the Myth that it applies only to Sorted Arrays).

