International Journal of Computer and Information Technology (ISSN: 2279 – 0764)
Volume 12– Issue 01, March 2023

Comparative Analysis on the Evaluation of the Complexity of C, C++, Java, PHP and Python Programming Languages based on Halstead Software Science

Kevin Agina Onyango
Department of Information Technology
Murang'a University of Technology
Nairobi, Kenya
Email: konyango [AT] mut.ac.ke

Geoffrey Wambugu Mariga
Department of Information Technology
Murang'a University of Technology
Nairobi, Kenya
Email: gmariga [AT] mut.ac.ke

Abstract— **Quality plays center stage in any software development industry. Software metrics have proven over time as the best measure to be used to assess and assure the software developers of the quality of their products. Halstead software science is an essential technique for measuring software complexity at the source code. In this study, we present a comparative study using this technique to help the developer by evaluating the code complexity by considering the structural composition of a programming language. In this study, an experiment was done using Halstead metrics to evaluate the complexity of PHP, C++, Java, C and Python programming languages. This study demonstrate that Halstead gives a better approach in evaluating the level of complexity of programming languages at source code level. The results showed that C++ and Java are the most complex programming languages while Python was the least complex warranting less of the programmer's time and effort when developing a similar project. These findings can be used by the software developers to make decisions on the programming language to adopt when they want to come up with less complex software of high quality. In the future, the researchers will advance the study to incorporate other software paradigms and also modify the technique to capture also inter and intra-modular structural complexity of the various programming languages.**

Keywords; Code Complexity, Complexity Evaluation, Halstead Software Science, Programming Language

I. INTRODUCTION

Software complexity explains how complex the components of systems are by defining how a particular set of features of the systems interrelate. For instance, the higher the interaction of these features, the higher the complexity meaning that the system will be complicated making it difficult to test, modify, maintain and understand [1, 2].

The planning, design, implementation, and testing phases of the software development process are all interconnected. The software industry relies on measuring the complexity of various software phases to produce efficient and reliable software programs.

The difficulty of assessing software complexity has been embraced strongly by the software developers [3]. The amount of computing work required to develop, maintain, and execute software code can thus be defined as software complexity [4]. McCabes Cyclomatic Complexity metric and Halstead Software science are the popular techniques for the evaluation of software code complexity [5,6,4]. McCabes Cyclomatic Complexity gives the static analysis of code complexity by considering the linearly independent paths in a control flow graph, while Halstead measures considers the complexity of a program in terms of the interconnections of operands and operators [7,8]. This study tries to come up with a comparative study to show code complexity can be evaluated using Halstead software science to measure the complexity of various programming languages to give programmers a better direction on the type of programming language to choose when developing their projects.

Maurice Howard Halstead as explained in [9] created Halstead complexity measures, which are software measurements considering a programming language at an algorithmic level to be composed of only operands and operators. Halstead's aimed at identifying these operands and operators are the main measurable features that contributes to the complexity of a programming language. This technique has been widely adopted since it is simple to compute and come with better support tools [10, 11].

The rest of this paper is organized as follows: related works are discussed in section 2, section 3 describes the materials used and the experimental procedure used in this study, section 4 summarizes the study findings, a discussion of the experimental findings are presented in section 5 of this document while conclusions, recommendations and future scope of the study are presented in section 6.

II. RELATED WORKS

Today the idea of software code evaluation has attracted attention in the software industries. One of the popular techniques being used for software code evaluation process is Halstead software science. Several studies have been done to evaluate the complexity of different programming languages using the Halstead software science.

Abdulkareem & Abboud in 20221 [8] conducted an experiment that used Halstead metrics to measure the complexity of function and branching structures in Java, Python, C++, and Javascript programming languages. The results demonstrated that Java has the highest effort requirement, difficulty, program length, estimated length, truth program length, volume, and program time while python has the least effort requirement.

The complexity of a program that checks palindromes in five different languages namely; C, C++, PHP, JAVA, and Python, was measured using Halstead metrics. The results of the study were as follows in descending order: Difficulty: Java, C++, C, PHP, Python. Effort: Java, C++, C, PHP, Python. Time: Java, C++, C, PHP, Python. Bugs delivered: C++, Java, C, PHP, Python as seen in Govil in his work in 2020 [12].

A study was done using Halstead metrics to measure the complexity level of a C++ and a Python program. The study focused on nested while… and for… loops. The results demonstrated that C++ is more complex than python in effort required, the number of bugs expected, and time requirements [13].

III. METHODOLOGY

A. *Introduction*

This comparative study follows a repeat study methodology. The study is a repeat study of Govil's work [12], who did a study to evaluate the complexity levels in five different languages namely; JAVA, C++, PHP, C, and Python, which was measured using Halstead metrics. Therefore, the materials and methods including the datasets and technique used for this study are mapped to the study being repeated and the comparative results presented. As shown in the subsequent sub-sections, the detailed explanation of the technique used and the nature of the experiment done gives a "blueprint" of the study and sufficient information for future repeat studies to be conducted from this study.

B. *Research Design*

This comparative study adopted an experimental design following the quantitate approach, where the technique used, that is, Halstead Software Science was used during the experiment to evaluate the code complexity of five different programming languages viz; C, C++, PHP, JAVA, and Python. The complexity results of these five programming languages are expressed in the figure – quantitative approach presented in four base metrics and nine derived metrics of the Halstead Software Science as elaborated in the subsequent sub-sections.

C. *The Technique Used*

To achieve the aim of the study, a comparative study was done using Halstead Software Science as it was considered the best metric for the implementation since it defines outstanding features that can easily be measured in a programming language. It is also being a software metric that reflects the implementation of algorithms in different programming languages and it is capable of measuring the complexity of a program code efficiently with better support tools.
Halstead Software Science follows a mathematical formulation that is used to compute software metrics, this technique computes software complexity of programming languages in two-level viz four base metrics and nine derived metrics.

Halstead Software Science
η_1= number of (distinct) unique operators
η_2= number of (distinct) unique operands
N_1=total number of operators
N_2=total number of operands

All the nine (9) Derived Halstead metrics are computed from the above base measures as shown in the equations below:

Program Vocabulary (η) = sum of (distinct) unique operands and (distinct) unique operators
$\eta = \eta_1 + \eta_2$ (1)

Length of a program (N)= Sum of the total number of operands and the total number of operators
$N = N_1 + N_2$ (2)

Estimated Length $\dot{N} = \eta_1 \log_2 (\eta_1) + \eta_2 \log_2 (\eta_2)$ (3)

Truth Length $= \frac{\dot{N}}{N}$ (4)

Program Volume (V)= $(N1+N2) \log_2 (\eta_1 + \eta_2)$ or $V = N \log_2 (\eta)$ (5)

Program Difficulty (D)= Number of unique operators and total usage of operands.

$D == (\frac{\eta 1}{2}) * (\frac{N2}{\eta 2})$ (6)

Program Effort (E)= This is the effort required in implementing or understanding the program. It is directly proportional to difficulty and volume.
$E = D*V$ (7)

Number of bugs (B)= calculating the flaws available in a program

$$B = \frac{V}{3000} \qquad (8)$$

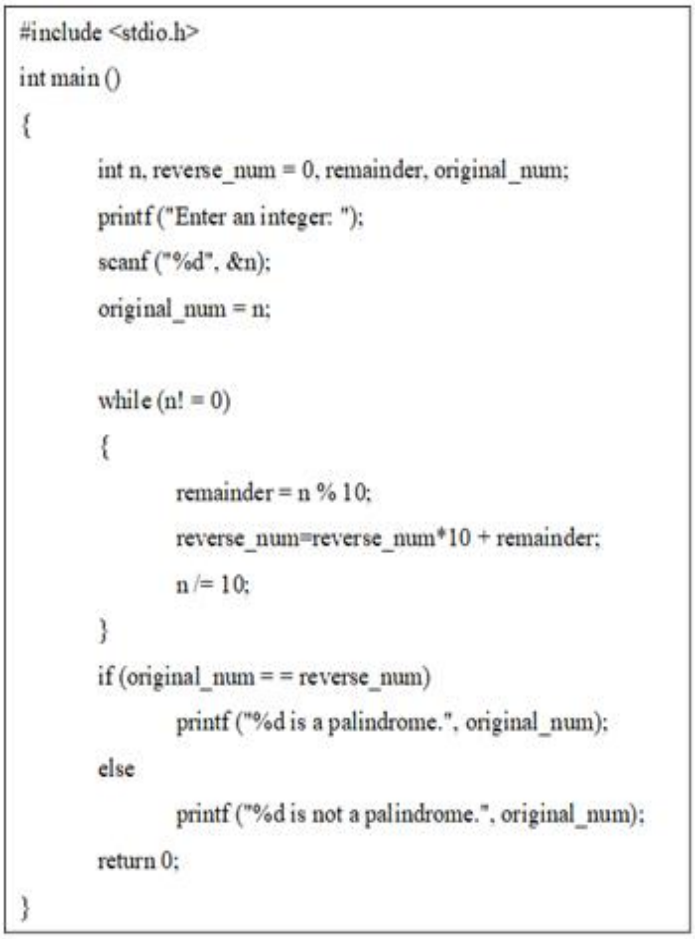
Program Time (T)= it is the time required to code the program. Time is directly proportional to effort.

$$T = \frac{E}{18} \qquad (9)$$

D. Dataset Used and Complexity Computation Experiment

This study is a repeat study on the use of Halstead Software Science to evaluate and compare the results of the complexity level of codes written in C, C++, Java, PhP and Python programming languages [12]. The datasets (program codes) used in this study are a program to check whether a number is a Palindrome or not in these five different programming languages (C, C++, Java, PHP, and python).

Scenario 1: Program to check Palindrome in C Programming Language



```
#include <stdio.h>
int main ()
{
        int n, reverse_num = 0, remainder, original_num;
        printf ("Enter an integer: ");
        scanf ("%d", &n);
        original_num = n;

        while (n! = 0)
        {
                remainder = n % 10;
                reverse_num=reverse_num*10 + remainder;
                n /= 10;
        }
        if (original_num = = reverse_num)
                printf ("%d is a palindrome.", original_num);
        else
                printf ("%d is not a palindrome.", original_num);
        return 0;
}
```

Figure 1. Program to check Palindrome in C Programming Language

The first scenario shown in Figure 1 is a code to check Palindrome in C Programming Language. The computation of Halstead Software Science base metrics results is shown in Table 1.

TABLE 1. HALSTEAD'S BASE METRICS FOR C PROGRAMMING LANGUAGE

Operators	**Frequency**	**Operands**	**Frequency**
{}	2	int	2
=	5	main ()	1
= =	1	n	5
;	10	reverse_num	4
while ()	1	0	3
%	1	remainder	3
%d	3	original_num	5
/	1	printf ()	3
*	1	scanf ()	1
%n	1	10	3
:	1	***n2=10***	***N2=30***
!	1		
+	1		
.	3		
else	1		
return	1		
,	6		
#	1		
include< >	1		
if ()	1		
" "	4		
n1=21	***N1=47***		

Now the base metrics are used to compute the nine derived metrics,

$$\text{Program Vocabulary } (\eta) = \eta_1 + \eta_2 = 21+10 = 31$$

$$\text{Length of a program } (N) = N_1 + N_2 = 47 + 30 = 77$$

$$\text{Estimated Length } \grave{N} = \eta_1 \log_2 (\eta_1) + \eta_2 \log_2 (\eta_2) = 21 \log_2(21) + 10 \log_2(10) = 92.24 + 33.22 = 125.46$$

$$\text{Truth Length} = \frac{\grave{N}}{N} = 125.46/77 = 1.63$$

Program Volume (V)= (N1+N2) $\log_2$ (η_1+ η_2) or V= N $\log_2$ (η)

$$= 77 \log_2 (31) = 381.47$$

$$\text{Program Difficulty (D)} = \left(\frac{\eta 1}{2}\right) * \left(\frac{N2}{\eta 2}\right) = (21/ 2) * (30/10) = 10.2 * 3 = 30.6$$

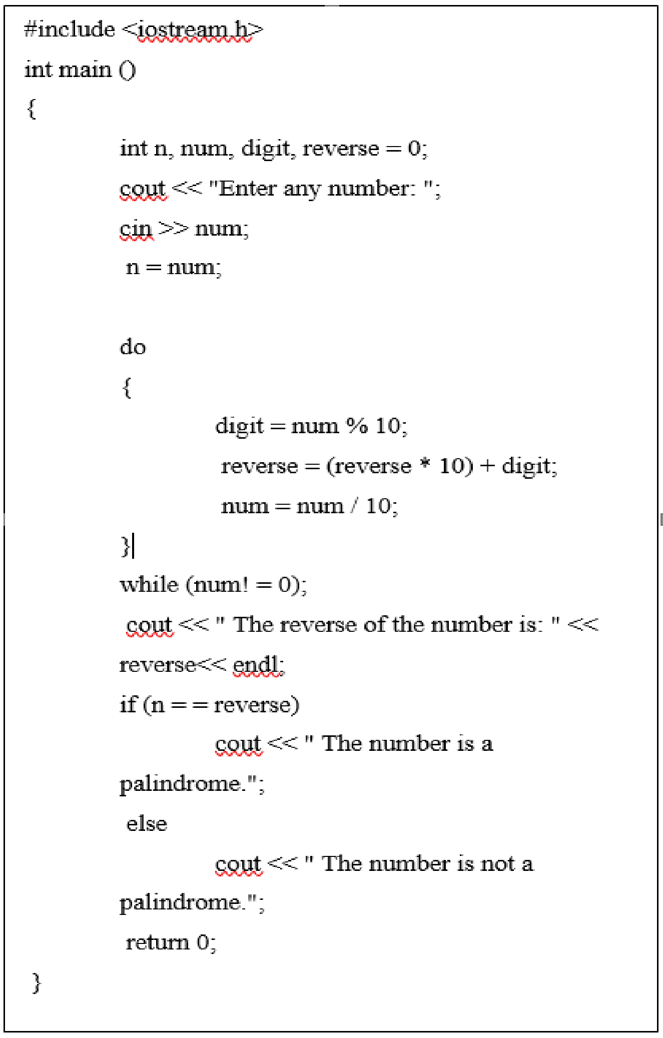
$$\text{Program Effort (E)} = D*V = 30.6*381.47 = 11{,}672.982$$

$$\text{Number of bugs (B)} = \frac{V}{3000} = 381.47/ 3000 = 0.127$$

$$\text{Program Time (T)} = \frac{E}{18} = 11{,}672.982/18 = 648.488 \text{ sec.}$$

The second scenario shown in Figure 2 represents the same program scenario of checking Palindrome but now the implementation is done using C++ Programming Language. The computation of Halstead Software Science base metrics was done and the results are shown in Table 2.

Scenario 2: Program to check Palindrome in C++ Programming Language



```
#include <iostream.h>
int main ()
{
        int n, num, digit, reverse = 0;
        cout << "Enter any number: ";
        cin >> num;
         n = num;

        do
        {
                digit = num % 10;
                 reverse = (reverse * 10) + digit;
                 num = num / 10;
        }|
        while (num! = 0);
         cout << " The reverse of the number is: " <<
        reverse<< endl;
        if (n = = reverse)
                cout << " The number is a
        palindrome.";
         else
                cout << " The number is not a
        palindrome.";
         return 0;
 }
```

Figure 2: Program to check Palindrome in C++ Programming Language

TABLE 2. HALSTEAD'S BASE METRICS FOR C++ PROGRAMMING LANGUAGE

Operators	**Frequency**	**Operands**	**Frequency**
{ }	2	Int	2
=	6	main ( )	1
= =	1	N	3
;	12	Num	7
while ( )	1	digit	3
%	1	reverse	5
/	1	0	3
*	1	count	4
+	1	Cin	1
!	1	10	3
:	2	Endl	1
.	3	***n2=11***	***N2=33***
else	1		
return	1		
,	3		
#	1		
include < >	1		
if ( )	1		
" "	4		
do	1		
<<	6		
>>	1		
( )	1		
n1= 23	**N1=53**		

Now the base metrics are used to compute the nine derived metrics,

Program Vocabulary (η) = $\eta_1 + \eta_2$
= 23+11
= 34

Length of a program (N) = $N_1 + N_2$
= 53 +33
= 86

Estimated Length $\grave{N} = \eta_1 \log_2(\eta_1) + \eta_2 \log_2(\eta_2)$
= 23 $\log_2(23)$ +11 $\log_2(11)$
=104.052 + 38.054
= 142.106

Truth Length = $\frac{\grave{N}}{N}$
= 142.106/86
= 1.6524

Program Volume (V)= (N1+N2) $\log_2(\eta_1 + \eta_2)$ or V= N $\log_2(\eta)$
= 86 $\log_2(34)$
= 437.5218

Program Difficulty (D)= $(\frac{\eta 1}{2}) * (\frac{N2}{\eta 2})$
= (23/ 2) * (33/11)
= 11.5 * 3
= 34.5

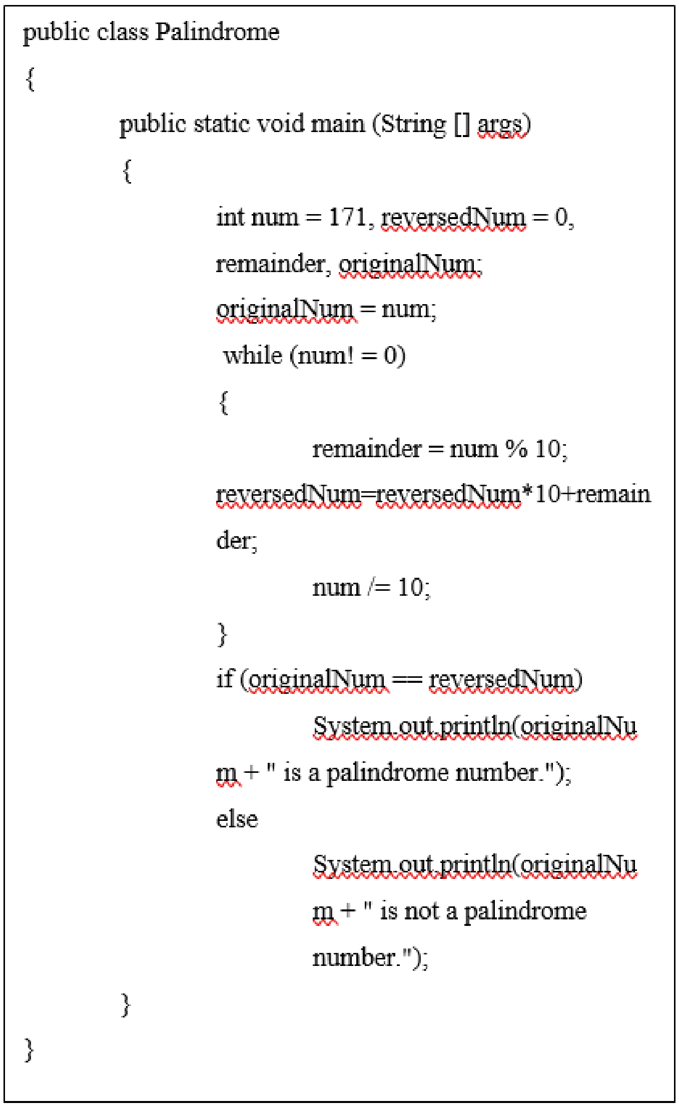
Program Effort (E) = D*V
= 34.5*437.5218
= 15,094.5021

Number of bugs (B) = $\frac{V}{3000}$
= 437.5218/ 3000
= 0.1458

Program Time (T) = $\frac{E}{18}$
= 15,094.5021/18
= 838.58 sec.

The third scenario shown in Figure 3 represents the same program scenario of checking Palindrome but now the implementation is done using JAVA Programming Language. The computation of Halstead Software Science base metrics was done and the results are shown in Table 3.

Scenario 3: Program to check Palindrome in JAVA Programming Language



```
public class Palindrome
{
        public static void main (String [] args)
        {
                int num = 171, reversedNum = 0,
                remainder, originalNum;
                originalNum = num;
                 while (num! = 0)
                {
                        remainder = num % 10;
                reversedNum=reversedNum*10+remain
                der;
                        num /= 10;
                }
                if (originalNum == reversedNum)
                        System.out.println(originalNu
                m + " is a palindrome number.");
                else
                        System.out.println(originalNu
                        m + " is not a palindrome
                        number.");
        }
}
```

Figure3. Program to check Palindrome in JAVA Programming Language

TABLE 3. HALSTEAD'S BASE METRICS FOR JAVA PROGRAMMING LANGUAGE

Operators	**Frequency**	**Operands**	**Frequency**
{}	3	main ( )	1
[ ]	1	String	1
=	7	Int	1
while ( )	1	Num	5
else	1	reversedNum	4
if ( )	1	Remainder	3
= =	1	originalNum	5
+	3	0	2
%	1	10	2
/	1	171	1
!	1	***n2=10***	***N2=25***
" "	2		
.	6		
;	7		
*	1		
,	3		
void	1		
public	2		
static	1		
class	1		
System	2		
out	2		
Args	1		
println ( )	2		
n1=24	***N1=52***		

Now the base metrics are used to compute the nine derived metrics,

Program Vocabulary (Ƞ) = $Ƞ_1 + Ƞ_2$
= 24+10
= 34

Length of a program (N) = $N_1 + N_2$
= 52 + 25
= 77

Estimated Length $\dot{N} = \eta_1 \log_2 (\eta_1) + \eta_2 \log_2 (\eta_2)$
= $24 \log_2(24) + 10 \log_2(10)$
=110.039 + 33.219
= 143.258

Truth Length = $\frac{\dot{N}}{N}$
= 143.258/77
= 1.86

Program Volume (V)= $(N1+N2) \log_2 (\eta_1 + \eta_2)$ or $V = N \log_2 (\eta)$
= $77 \log_2 (34)$
= 391.7346

Program Difficulty (D)= $(\frac{\eta 1}{2}) * (\frac{N2}{\eta 2})$
= (24/ 2) * (25/10)
= 12 * 2.5
= 30

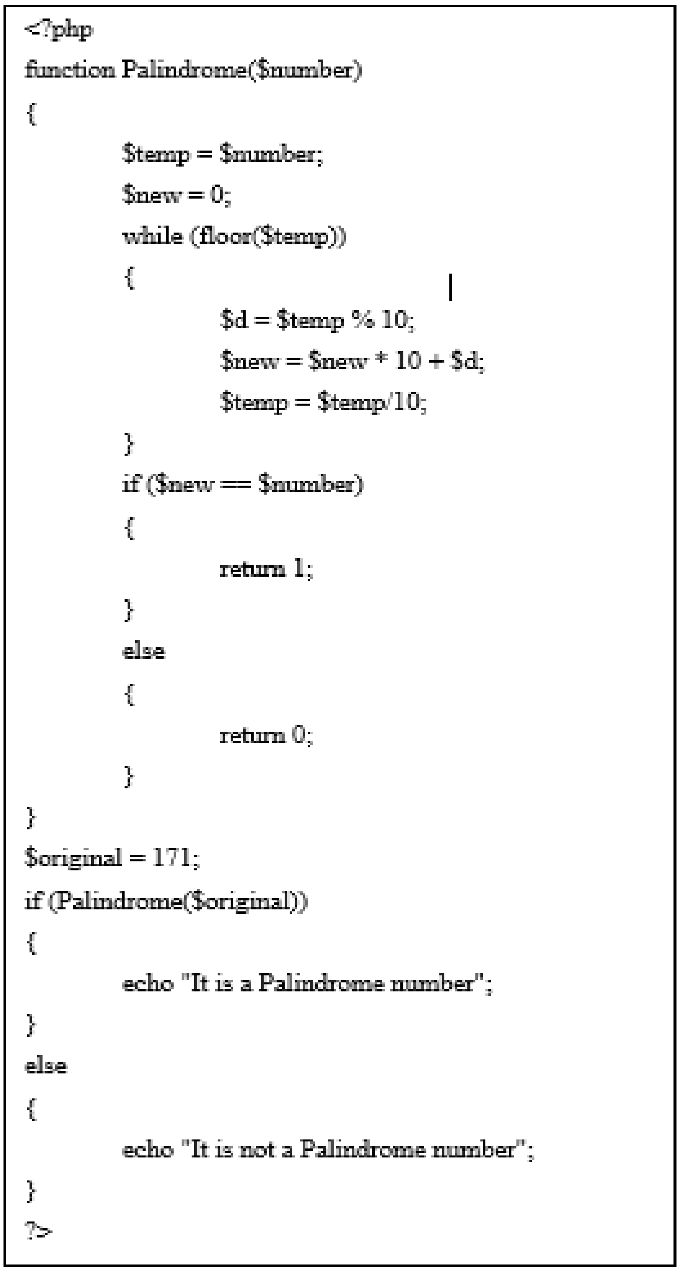
Program Effort (E) = D*V
= 30*391.7346
= 11,752.038

Number of bugs (B) = $\frac{V}{3000}$
= 391.7346/ 3000
= 0.1306

Program Time (T) = $\frac{E}{18}$
= 11,752.038/18
= 652.891 sec.

The fourth scenario shown in Figure 4 represents the same program scenario of checking Palindrome implemented using PHP Programming Language. The computation of Halstead Software Science base metrics was done and the results are shown in Table 4.

Scenario 4: Program to check Palindrome in PHP Programming Language



```
<?php
function Palindrome($number)
{
        $temp = $number;
        $new = 0;
        while (floor($temp))
        {
                $d = $temp % 10;
                $new = $new * 10 + $d;
                $temp = $temp/10;
        }
        if ($new == $number)
        {
                return 1;
        }
        else
        {
                return 0;
        }
}
$original = 171;
if (Palindrome($original))
{
        echo "It is a Palindrome number";
}
else
{
        echo "It is not a Palindrome number";
}
?>
```

Figure 4. Program to check Palindrome in PHP Programming Language

TABLE 4. HALSTEAD'S BASE METRICS FOR PHP PROGRAMMING LANGUAGE

Operators	**Frequency**	**Operands**	**Frequency**
<	1	$number	3
>	1	function	1
?	2	php	1
{ }	6	$temp	5
( )	3	$new	4
=	6	0	2
= =	1	floor	1
%	1	$d	2
*	1	1	1
+	1	10	3
return	2	171	1
while ( )	1	$original	2
if ( )	2	***n2=12***	***N2=26***
else	2		
echo	2		
" "	2		
;	10		
n1=17	***N1=44***		

Now the base metrics are used to compute the nine derived metrics,

Program Vocabulary (η) = $\eta_1 + \eta_2$
= 17+12
= 29

Length of a program (N) = $N_1 + N_2$
= 44 + 26
= 70

Estimated Length $\grave{N} = \eta_1 \log_2(\eta_1) + \eta_2 \log_2(\eta_2)$
= 17 $\log_2(17)$ +12 $\log_2(12)$
=69.275 + 43.02
= 112.295

Truth Length = $\frac{\grave{N}}{N}$
= 112.295/70
= 1.6042

Program Volume (V)= (N1+N2) $\log_2(\eta_1 + \eta_2)$ or V= N $\log_2(\eta)$
= 70 $\log_2(29)$
= 340.059

Program Difficulty (D)= $(\frac{\eta 1}{2}) * (\frac{N2}{\eta 2})$
= (17/ 2) * (26/12)
= 8.5 * 2.1667
= 18.417

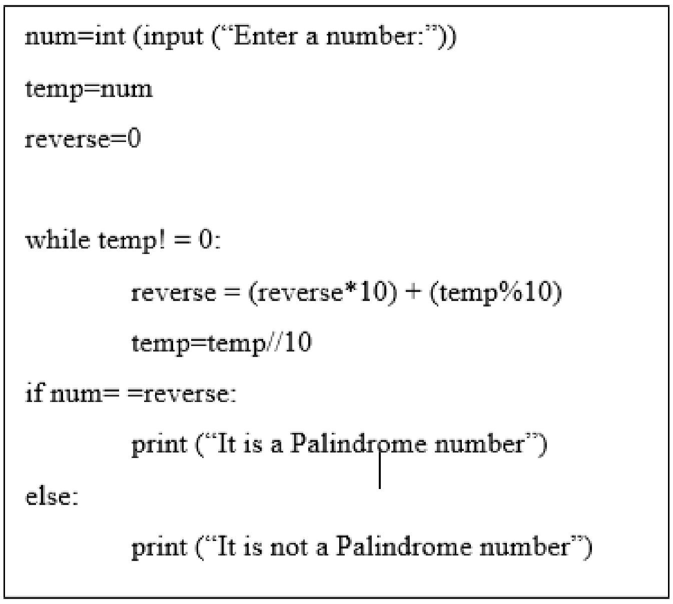
Program Effort (E) = D*V
= 18.417*340.059
= 6,262.877

Number of bugs (B) = $\frac{V}{3000}$
= 340.059/ 3000
= 0.1134

Program Time (T) = $\frac{E}{18}$
= 6262.877/18
= 347.938 sec.

The fifth scenario shown in Figure 5 represents the same program scenario of checking Palindrome implemented using PHP Programming Language. The computation of Halstead Software Science base metrics was done and the results are shown in Table 5.

Scenario 5: Program to check Palindrome in PYTHON Programming Language



```
num=int (input ("Enter a number:"))
temp=num
reverse=0

while temp! = 0:
            reverse = (reverse*10) + (temp%10)
            temp=temp//10
if num= =reverse:
            print ("It is a Palindrome number")
else:
            print ("It is not a Palindrome number")
```

Figure 5: Program to check Palindrome in PYTHON Programming Language

TABLE 5: HALSTEAD'S BASE METRICS FOR JAVA PROGRAMMING LANGUAGE

Operators	**Frequency**	**Operands**	**Frequency**
=	6	num	3
( )	3	int	1
" "	3	temp	5
!	1	0	2
*	1	reverse	4
+	1	10	2
%	1	***n2=6***	***N2=17***
//	1		
= =	1		
:	3		
while	1		
else	1		
print ( )	2		
input ( )	1		
.	1		
if	1		
n1= 16	***N1=28***		

Now the base metrics are used to compute the nine derived metrics,

Program Vocabulary (η) = $\eta_1 + \eta_2$
= 16+6
= 22

Length of a program (N) = $N_1 + N_2$
= 28 + 17
= 45

Estimated Length $\grave{N}$ = $\eta_1 \log_2 (\eta_1) + \eta_2 \log_2 (\eta_2)$
= 16 $\log_2(16)$ +6 $\log_2(6)$
=64 + 15.51
= 79.51

Truth Length = $\frac{\grave{N}}{N}$
= 79.51/45
= 1.767

Program Volume (V)= (N1+N2) $\log_2 (\eta_1 + \eta_2)$ or V= N $\log_2 (\eta)$
= 45 $\log_2$ (22)
= 200.6744

Program Difficulty (D)= $(\frac{\eta 1}{2}) * (\frac{N2}{\eta 2})$
= (16/ 2) * (17/6)
= 8 * 2.833
= 22.664

Program Effort (E) = D*V
= 22.664*200.6744
= 4,548.085

Number of bugs (B) = $\frac{V}{3000}$
= 200.6744/ 3000
= 0.0669

Program Time (T) = $\frac{E}{18}$
= 4548.085/18
= 252.714 sec.

IV. RESULTS

A. Comparative Analysis of the Results

Following directly from the Materials and Methods used in this study as discussed and elaborated in the immediate previous section. The results show that using Halstead Software Science, the complexity of different programming languages varies even if they are subjected to the same task. Table 6 gives the comparative analysis of the code complexity results evaluated by Halstead Software Science for the five programming languages viz C, C++, Java, PhP and Python when used to compute Palindrome number.

TABLE 6: COMPARATIVE ANALYSIS OF THE RESULTS

Metrics Level	**Halstead Metrics**	**Program in C**	**Program in C++**	**Program in Java**	**Program in PHP**	**Program in Python**
Base Metrics	η_1	21	23	24	17	16
	η_2	10	11	10	12	6
	N_1	47	53	52	44	28
	N_2	30	33	25	26	17
Derived Metrics	η	31	34	34	29	22
	N	77	86	77	70	45
	$\dot{N}$	125.46	142.106	143.258	112.295	79.51
	V	381.47	437.5218	391.7346	340.059	200.6744
	D	30.6	34.5	30	18.417	22.664
	E	11,672.982	15,094.5021	11,752.038	6,262.877	4,548.085
	T	648.488sec	838.58sec	652.891sec	347.938sec	252.714sec
	B	0.127	0.1458	0.1306	0.1134	0.0669

B. Summary and Illustration of the Findings

This subsection gives the summary and illustration of the comparative analysis findings presented in Table 6. Here the various selected derived metrics of the Halstead metrics have been graphically presented to give a comparative analysis of the code complexity of the five programming languages based on these parameters-derived metrics.

a. Comparative analysis based on Program Length

Figure 6 shows the comparison of the complexity results of the program length of the five programming languages. C++ gave the highest complexity value of 86, followed by Java programming language which had the same length as C programming language giving a value of 77, then PHP was the third most lengthy programming language at 70, finally, Python was the shortest of the five giving a program length of 45.

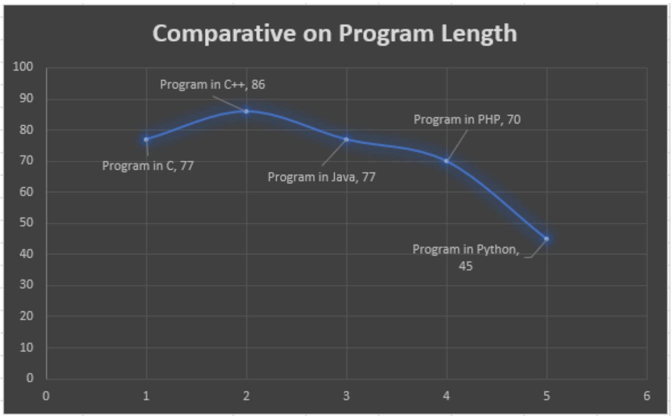


Figure 6: Comparison Chart on Program Length (L)

b. Comparative Analysis based on Program Volume

Figure 7 shows the comparison of the complexity results of program volume of the five programming languages. C++ gave the highest complexity value of program volume at 437.5218, followed by Java programming language at 391.7346, the third in terms of volume was C programming language giving a value of 381.47, then PHP programming language at 340.059, finally Python was the last of the five giving a program volume of 200.6744.

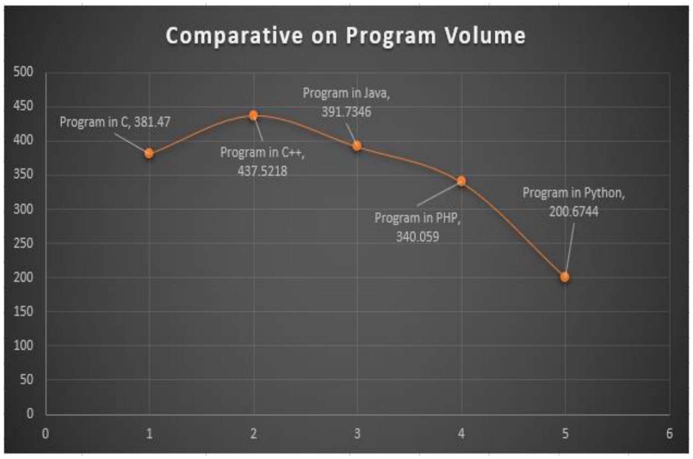


Figure 7: Comparison Chart on Volume (V)

c. Comparative Analysis based on Program Difficulty

Figure 8 shows the comparison of the complexity results of program difficulty of the five programming languages. C++ gave the highest complexity value of program difficulty at 34.5, and Java and C programming languages followed closely at 30 and 30.6 respectively. Python came forth in terms of program difficulty giving a difficulty value of 22.664 and finally, the less difficult programming language to learn among the five was PHP with a difficulty value of 18.417 according to the code complexity experiment done in this study.

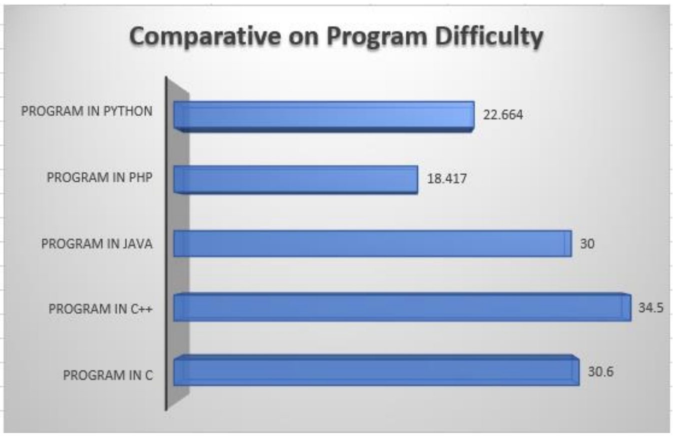


Figure 8: Comparison Chart on Program Difficulty (D)

d. Comparative Analysis based on Program Effort

Figure 9 shows the comparison of the complexity results of program Effort required of the five programming languages. C++ gave the highest complexity value of program effort of 15,094.50, and C and Java programming languages followed closely at 11,672.98 and 11,752.04 respectively. PHP came forth in terms of program effort required giving the effort value of 6,262.88 finally the programming language with the least effort required among the five was Python with an effort value of 4,548.09 according to the findings experiment done in this study.

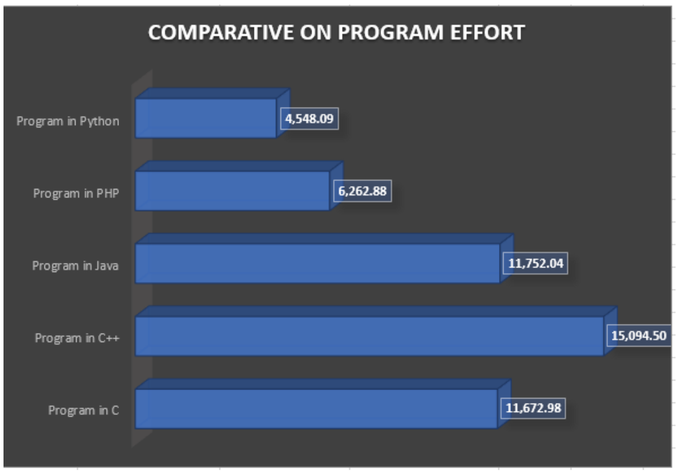


Figure 9: Comparison Chart on Program Effort (E)

e. Comparative Analysis based on Program Time

Figure 10 shows the comparison of the complexity results of program Time of the five programming languages. C++ gave the highest complexity value of program time of 838.58sec, followed closely by Java and C programming language at652.891sec and 648.488sec respectively, then PHP programming language at 347.938sec, finally, Python was the last of the five giving a program time of 252.714sec according to the findings of this study.

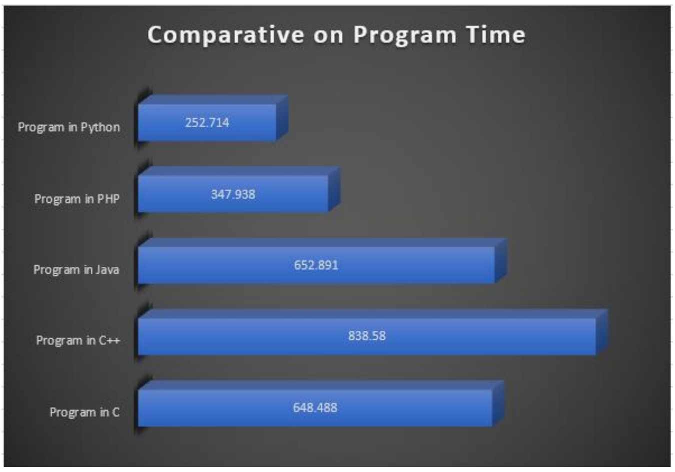


Figure 10: Comparison Chart on Program Time (T)

f. Comparative Analysis based on Program Bugs Delivered

Figure 11 shows the comparison of the complexity results of program Bugs delivered among the five programming languages. The study findings showed that C++ gave the highest complexity of bugs value of 0.1458, followed by Java programming language with a bug value of 0.1306, the third in terms of bugs delivered was C programming language giving a value of 0.127, then PHP programming language at 0.1134, finally Python giving the least complexity value of number of bugs delivered at the value of 0.0669.

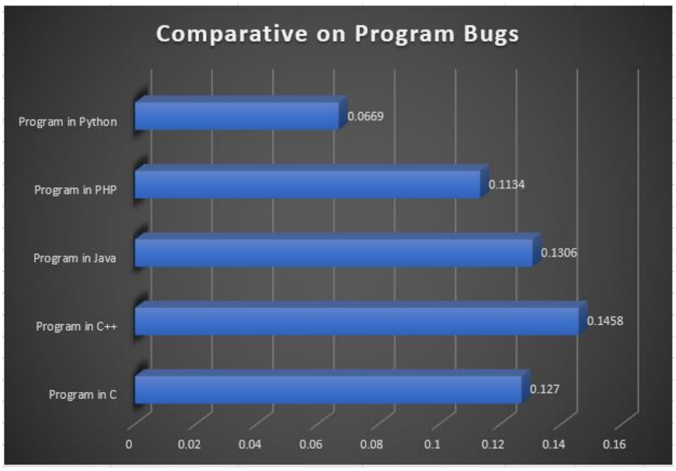


Figure 11: Comparison Chart on Program Bugs Delivered (B)

V. DISCUSSION

The study findings graphically show six (6) most substantial derived metrics of the Halstead Software Science viz program length, volume, difficulty, effort, time and bugs delivered. The findings show that writing Palindrome programming coding in C++ is the lengthiest while code written in Python for the same is the shortest of the five programming languages used in this study. This implies that more lines of code will be employed when developing software using C++ compared to developing the same software using python. In terms of volume, the same observation was made, where when the five programming languages were subjected to code the same program, C++ gave the largest volume while Python gave the smallest volume, so this means that more work will be required by the developer using C++ in the development process than the developer implementing the same project using python. C++ continued to prove to be the most difficult programming language to learn compared to the other four programming languages, however, PHP was observed to be the least difficult. More programmer effort and time was observed to be needed when developing a project in the C++ programming language followed by the Java programming language, on the flip side, less programmer effort and time will be employed when developing a project using the Python programming language. The study findings also revealed that fewer bugs will be generated when the developer implements a program in python while more bugs will be witnessed when the same project is implemented using C++ programming language.

Being a comparative analysis, the findings of this study have a positive correlation with the previously published work. All the identified derived metrics have a one-on-one mapping with the findings of the previous study with very insignificant exceptions, showing that on the majority of the derived Halstead metrics, C++ and Java were leading in the code complexity while Python and PHP were the least complex, leaving C programming language to be moderate in terms of code complexity evaluation experiment when these five programming languages were subjected to run the same program of determining whether a number is a palindrome or not.

VI. CONCLUSION AND FUTURE WORKS

The findings of this study revealed that C++ is the most complex programming language compared to Java, C, PHP and Python. When developing a project in C++, more lines of codes, program volume, program difficulty as well as programmer effort and time will be required, more bugs will also be delivered when coding with C++ followed by Java, C, then PHP finally python. The findings of this study can help software developers to make important decisions regarding software costing, software quality assurance, and software maintenance, among others.

In the future, more studies can be done by applying the Halstead software science technique to other programming domains like Aspect-oriented programming, besides modification of the Halstead can be done in the future to capture the complexity of a programming language by considering both inter-modular and intra-modular composition of operands and operations interactions in the program structure.

REFERENCES

[1] Vard Antinyan, Miroslaw Staron, and Anna Sandberg, (2017) "Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time", Empirical Software Engineering, Vol. 22, No. 6, pp. 3057-3087,2017.

[2] M. Madhan, I. Dhivakar, T. Anbuarasan, and Chandrasegar Thirumalai. (2017) "Analyzing complexity nature inspired optimization algorithms using 4.halstead metrics." In 2017 International Conference on Trends in Electronics and Informatics (ICEI), pp. 1077-1081. IEEE.

[3] Safa Omri, Pascal Montag, and Carsten Sinz. (2018)" Static Analysis and Code Complexity Metrics as Early Indicators of Software Defects", Journal of Software Engineering and Applications 11, No. 04, pp. 153-166.

[4] Wilch, J., Fischer, J., Neumann, E. M., Diehm, S., Schwarz, M., Lah, E., ... & Vogel-Heuser, B. (2019, October). Introduction and evaluation of complexity metrics for network-based, graphical IEC 61131-3 programming languages. In IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society (Vol. 1, pp. 417-423). IEEE.

[5] F. Fioravanti, P. Nesi, (2000 August 31). "A method and tool for assessing object-oriented projects and metrics management," Journal of Systems and Software, Volume 53, Issue 2, Pages 111-136.

[6] Madi, O. K. Zein, S. Kadry. (2013). On the Improvement of Cyclomatic Complexity Metric, vol. 7 no. 2,

[7] Abdul Rehman Shaikh, "Applying Halstead Metrics in Your Programs", https://www.academia.edu/23024048/Applying_Halstead_Metrics_in_Your_Programs/ last accessed on 18 March 2020.

[8] Abdulkareem, S. A., & Abboud, A. J. (2021, February). Evaluating Python, C++, JavaScript and Java Programming Languages Based on Software Complexity Calculator (Halstead Metrics). In IOP Conference Series: Materials Science and Engineering (Vol. 1076, No. 1, p. 012046). IOP Publishing.

[9] Balogun, M. O. (2022). Comparative Analysis of Complexity of C++ and Python Programming Languages. Asian J. Soc. Sci. Manag. Technol, 4, 1-12.

[10] Chandrasegar Thirumalai, Shridharshan R R, Ranjith Reynold L, "An Assessment of Halstead and COCOMO Model for Effort Estimation ", International Conference on Innovations in Power and Advanced Computing Technologies (i-PACT), April 2017.

[11] Coimbra, Rodrigo Tavares, Antônio Resende, and Ricardo Terra. "A Correlation Analysis between Halstead Complexity Measures and other Software Measures." In 2018 XLIV Latin American Computer Conference (CLEI), pp. 31-39. IEEE, 2018.

[12] Govil, N. (2020, June). Applying Halstead Software Science on Different Programming Languages for Analyzing Software Complexity. In 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI) (48184) (pp. 939-943). IEEE.

[13] T Hariprasad, K Seenu, G Vidhyagaran and Chandrasegar Thirumala. (2017, May) "Software Complexity Analysis Using Halstead Metrics", International Conference on Trends in Electronics and Informatics (ICEI) IEEE & 978-1-5090-4257-9.