Journal of
Innovation, Technology and Sustainability

A comparative study of the lexicographical complexity of Java, Python and C languages based on program characteristics

Kevin Agina Onyango[1]

Jackson Kamiri[1]

Geoffrey Muchiri Muketha[1]

[1]*Murang'a University of Technology, Kenya*

Abstract

In software engineering, software complexity measures how complicated it is to design, test, maintain, and comprehend a system or a program. Metrics have been appreciated over time as a measure of various attributes of software products. Some of the most well-known languages for scientific, object-oriented, and imperative programming are Python, Java, and C, respectively. However, it is not easy to distinguish the structural complexity of these programming languages and the existing studies have overlooked this issue. This study, therefore, uses a technique based on Halstead Software Science to conduct a comparative investigation to evaluate the lexicographical complexity of sequence, selection, and looping program structures in object-oriented, scientific, and imperative programming languages. Halstead Complexity Metrics were implemented utilizing sequence, selection, and loop control structures in Java, C, and Python to accomplish the study's goal. When subjected to the Halstead software science comprising of nine measurement criteria, the findings of the experiment demonstrated that in sequence and Loops program structures C language has the highest lexicographical complexity followed by Java, while in Selection program structures Java is more slightly complex than C. Python on the other hand, had the least lexicographical complexity across all three essential program structures—sequence, selection, and loops during the comparative study, therefore, it is the most appropriate programming language among the three that are being studied here in terms of program complexity. Using the results of this study, we intend to use effort prediction models in the future to estimate the programming effort. We also intend to do additional experiments with the same program structures using larger program samples in the future. A replication of the study using different programming languages is also suggested.

Keywords: *Software Complexity, Software Metrics, Halstead Software Science, C Programming, Java, Python*

1. Introduction

Software complexity is always at the center of attention of every software developer. It explains how complex the components of systems are by defining how a particular set of features of the systems interrelate (Vard, Miroslaw and Ann, 2017: Madhan, Dhivakar, Anbuarasan and Thirumalai, 2017). The higher the interaction, the higher the complexity meaning that the system is complicated making it difficult to test, modify, maintain and understand (Mall, 2016: Johanna and Baharom, 2017).

Over the years, software development industries have proven that metrics are the best measure of software complexity giving advisories to software developers to develop quality software (Omri, Montag and Sinz, 2018: Horst, 2019). In software engineering, the two widely used classical measures for evaluating software complexity are Halstead metrics and McCabe's cyclomatic complexity metrics (Halstead, 1977: Hariprasad, Seenu, Vidhyagaran and Tirumala, 2017). Halstead metrics examine the mathematical correlations between the number of operands and operators in a certain code structure to assess the complexity of software. (Govil, 2020). On the other hand, McCabe's Cyclomatic complexity metrics assess a software's complexity by taking the code structure's control flow into account. (Fioravanti and Nesi, 2000: Madi, Zein and Kadry, 2013).

Some of the most well-known programming languages for scientific, object-oriented, and imperative programming are Python, Java, and C, respectively. However, it is not easy to distinguish the structural complexity of these programming languages and the existing studies have overlooked this issue.

This study therefore aims at evaluating the complexity of Object-Oriented, Scientific and Imperative Programming languages based on the composition of the program structures. Halstead metrics was considered the best metrics for the implementation since it defines the quantifiable properties of software and how they relate to one another. Halstead metrics is a software science that reflects how algorithms are implemented in various programming languages. With the existence of the right tools, the metrics have been appreciated in the effectiveness of assessing the complexity of a program's code. (Binanto, Warnars, Abbas and Sianipar, 2018; Flatter and David, 2018; Shaikh, 2020).

The remainder of this work is structured as follows: Section 2 presents literature relevant to the study; Section 3 discusses methodology; Section 4 presents results and discussion; and Section 5 provides closing thoughts and suggestions for future study.

2. Related Works

Halstead software science has been used in numerous types of research to assess the complexity of various programming languages. Hariprasad et al., (2017) conducted a study on software complexity analysis using Halstead metrics. This study used Halstead's technique to measure the complexity levels of a C++ program and a Python program. In their study, the researchers conducted an experiment using the 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. However, this study overlooked the complexity evaluation of other popular programming languages like Java and C.

The complexity of a program that checks palindromes in a total of five languages namely; JAVA, C, Python, PHP, and C++, was measured using Halstead metrics (Govil, 2020). The study's findings in descending order are as follows: 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. In this study, the researchers only consider palindromes in the five different programming languages. This study, therefore, overlooked other basic program structures on sequence, selection and loops which are fundamental in the evaluation of the complexity of a programming language.

Abdulkareem & Abboud, (2021) did a study on the evaluation of the programming languages viz Java, C++, JavaScript, and Python using Halstead Metrics' software complexity calculator. They conducted an experiment using 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, volume, truth program length, estimated program length and program time while Python had the least effort requirement. This study only considered the branching program structure of these programming languages overlooking other program structures such as sequences and loops which are essential in the contribution of the software complexity evaluation process.

Yu and Zhou, (2010) conducted a study on a survey on metrics for software complexity, in their study they noted that software complexity measurement has become an important part of software engineering. Lines of Code (LOC), Halstead Complexity measurements, and Cyclomatic Complexity Metrics are a few of the traditional and effective software complexity measurements, according to this survey. This study though did not implement any complexity evaluation of any programming language, the study only focused on the survey and presentation of these metrics.

Binanto et. al., (2018) developed an automation tool using Python programming language. The analyzer tool is used for the automation processing of Halstead metrics application results. The tool can get operands and operators faster than manual computation. However, there is no evidence that the tool was validated. In another separate study, (Binanto et. al., 2018) did a study on Halstead metrics for Statcato's multiple versions' quality evaluation. In this study, the researcher extended their work by implementing their developed tool to analyze the Statcato software's version quality.

Alfadel et. al., (2017) did a study on the measurement of Defect Density using the Halstead and Cyclomatic Complexity Metrics. These software measures showed correlation results with the defect density. During the experimental design, the researchers found a linear correlation and such empirical results were found to be statistically significant, the study concluded that the Defect Density reported by these software measurements is consistent.

3. Methodology

According to Prabowo et. al. (2018), the Halstead technique is predicated on the notion that an actual program is made up of operators and operands. Therefore, it is possible to determine some software attributes, such as program length, volume, difficulty level, and programming effort, using information about the number of operators and operands present in a program and the frequency with which those operators and operands are used in a program. Keep in mind that the measurement criteria generated by this technique only approximate the actual condition and are not a statistical estimate.

Some of the terms and definitions used in this technique are:

- Operands- these are the variables and constants that a program is composed of.
- Operators- consist of any combination of symbols that could have an impact on the value or command operand.
- The operators also include signs, arithmetic symbols (like +, -, *, and /), keywords (like if, for, do, etc.), special symbols (like =, " ", (), ==, and! =), and names of functions.

This study used Halstead metrics to measure the Lexicographical Complexity of Object-Oriented, Scientific, and Imperative programming languages based on program characteristics. The programming languages used in this study are Java for Object-Oriented, Python for Scientific, and C for Imperative programming. For each of these programming languages, sequence structures, selection structures, and loop structures were used during the experimental test (geeksforgeek, 2020: javatpoint, 2020 and Pawade, Metha, Shah & Rathod, 2015).

The programs used in this paper were developed by the researchers and then tested to ensure that they execute with no error. Java and C programs were tested in NetBeans IDE while Python was tested in the Colab cloud-based environment (Pawade et al., 2015: James, 2017).

Halstead metrics is a mathematical formulation that is used to compute software metrics. According to Coimbra, Resende & Terra, (2018) and Abdulkareem & Abboud, (2021), the measures included in this metric are:

η_1= number of unique operators

η_2= number of unique operands

N_1=total number of operators

N_2=total number of operands

Halstead metrics that can be computed from the above base measures as shown in the equations below:

Program Vocabulary (η) = sum of unique operands and unique operators

$\eta = \eta_1 + \eta_2$ (1)

Length of a program (N)= total number of operators plus all operands added together

$N = N_1 + N_2$ (2)

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

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

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

Program Difficulty (D) = $(\eta_1/2) * (N2 / \eta_2)$ (6)

Program Effort (E) = D*V (7)

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

$B = V/3000$ (8)

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

$T=E/18$ (9)

The metrics for each of these programs and the comparison are done and discussed in the results section.

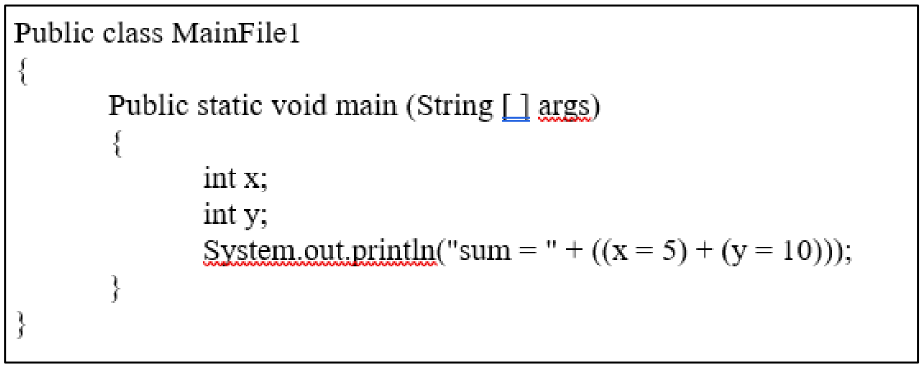
4. Results and Discussion

4.1 Computation of Lexicographical Complexity from a Sequence Program Structure

A comparison experimental test was done to compute the lexicographical complexity of source code between Java, Python and C programming language from a simple sequence program structure to add two integers as shown below. In this paper, we have used the same program structure in all three programming languages. For instance, variables are declared and initialized at the same time.

4.1.1 An Implementation in a Java Programming Language

Java was used to create a sequence program structure. To ensure that the program, as seen in Figure 1, runs effectively, it was tested and run in the NetBeans IDE.



```
Public class MainFile1
{
        Public static void main (String [ ] args)
        {
                int x;
                int y;
                System.out.println("sum = " + ((x = 5) + (y = 10)));
        }
}
```

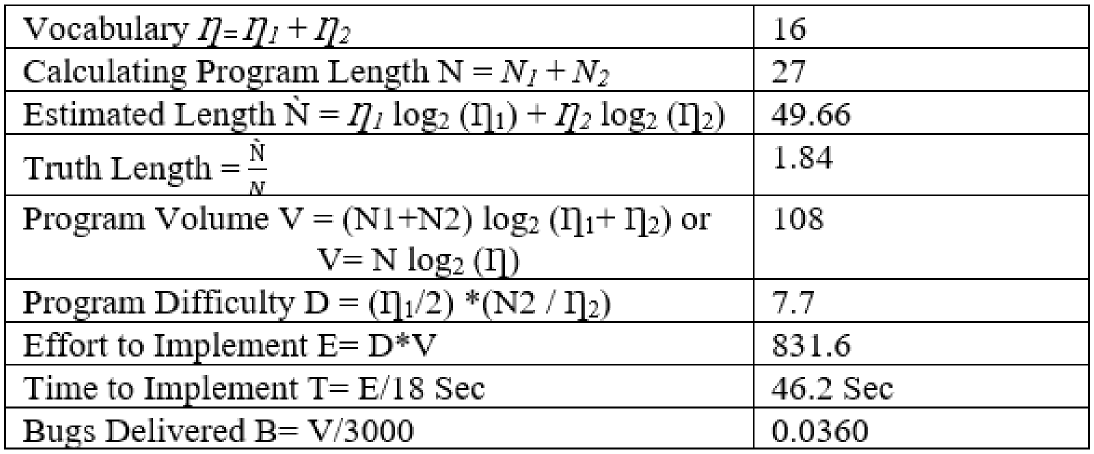
Figure 1: Sequence - Java Code

Tokenization of the Java code in Figure 1 has generated 5 distinct operands (η_2) with a frequency of 7 (N_2). The distinct operator is 11 (η_1) while the frequency is 20 (N_1). Each of the operators and operands considered in this program are summarized in Table 1. Computation of the Halstead metrics on the Java program in Figure 1 generated the following results; program vocabulary is 16, program length is 27, estimated length is 49.66, truth length is 1.84, program volume is 108, program difficulty is 7.7 effort to implement is 831.6, time to implement is 46.2 seconds while bugs delivered is equivalent to 0.0360. Table 2 shows a summary of the Halsted metrics in this program.

Table 1: Operands and Operators in a Sequence- Java code

Distinct Operators (η_1)	Frequency (N_1)	Distinct Operands (η_2)	Frequency (N_2)
=	3	x	2
+	2	y	2
String [ ]	1	sum	1
;	3	5	1
{ }	2	10	1
System.out.println( )	1		
Args	1	***$\eta_2 = 5$***	***$N_2 = 7$***
Int	2		
Public static void main( )	1		
( )	3		
“ ”	1		
$\eta_1 = 11$	***$N_1 = 20$***		

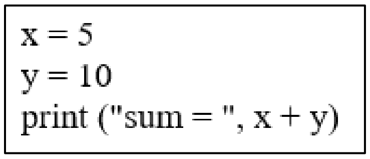
Table 2: Halstead Results of Java Sequence- Java code



Vocabulary $\eta = \eta_1 + \eta_2$	16
Calculating Program Length $N = N_1 + N_2$	27
Estimated Length $\grave{N} = \eta_1 \log_2 (\eta_1) + \eta_2 \log_2 (\eta_2)$	49.66
Truth Length $= \frac{\grave{N}}{N}$	1.84
Program Volume $V = (N1+N2) \log_2 (\eta_1 + \eta_2)$ or $V = N \log_2 (\eta)$	108
Program Difficulty $D = (\eta_1/2) * (N2 / \eta_2)$	7.7
Effort to Implement $E = D*V$	831.6
Time to Implement $T = E/18$ Sec	46.2 Sec
Bugs Delivered $B = V/3000$	0.0360

4.1.2 An Implementation in a Python Programming Language

A Python simple sequence program structure was developed to add two numbers. The program as shown in Figure 2 was developed and tested in the Colab cloud-based environment to ascertain that it executes successfully.



```
x = 5
y = 10
print ("sum = ", x + y)
```

Figure 2: Sequence - Python Code

The Python program in Figure 2 has 5 unique operators (*η_1*), the frequency of operators is 7 (*N_1*), 5 distinct operands (*η_2*) and the frequency of the distinct operands is 7 (*N_2*). Computation of the Halstead metrics for the program generated the following results: Program Vocabulary is 10, program length is 14, estimated length is 23.22, truth length is 1.66, program volume is 46.51, program difficulty is 3.5, effort to implement is 162.79, time to implement is 9.04 seconds, and Bugs delivered are equal to 0.0155. These results have been summarized in Tables 3 and 4 respectively.

Table 3: Operands and Operators in a Sequence- Python code

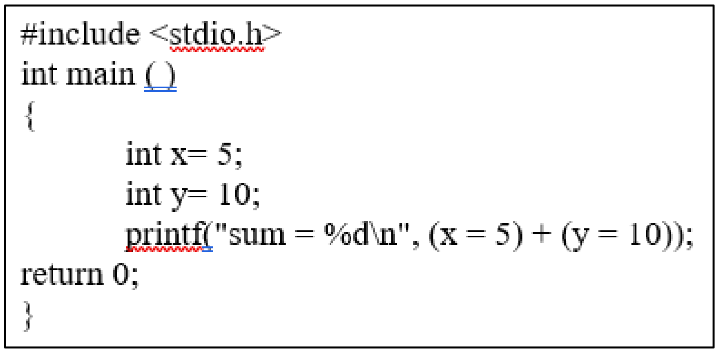
Distinct Operators (η_1)	**Frequency (N_1)**	**Distinct Operands (η_2)**	**Frequency (N_2)**
=	3	x	2
+	1	y	2
print( )	1	5	1
,	1	10	1
“ ”	1	Sum	1
$\eta_1 = 5$	***$N_1 = 7$***	***$\eta_2 = 5$***	***$N_2 = 7$***

Table 4: Halstead Results of Sequence- Python code

Vocabulary $\eta = \eta_1 + \eta_2$	10
Calculating Program Length $N = N_1 + N_2$	14
Estimated Length $\dot{N} = \eta_1 \log_2(\eta_1) + \eta_2 \log_2(\eta_2)$	23.22
Truth Length $= \frac{\dot{N}}{N}$	1.66
Program Volume $V = (N1+N2) \log_2(\eta_1 + \eta_2)$ or $V = N \log_2(\eta)$	46.51
Program Difficulty $D = (\eta_1/2) * (N2 / \eta_2)$	3.5
Effort to Implement $E = D*V$	162.79
Time to Implement $T = E/18$ Sec	9.04 Sec
Bugs Delivered $B = V/3000$	0.0155

4.1.3 An Implementation in a C Programming Language

The researchers used NetBeans IDE to develop a simple C program that adds and prints the sum of two integers as shown in Figure 3.



```
#include <stdio.h>
int main ( )
{
        int x= 5;
        int y= 10;
        printf("sum = %d\n", (x = 5) + (y = 10));
return 0;
}
```

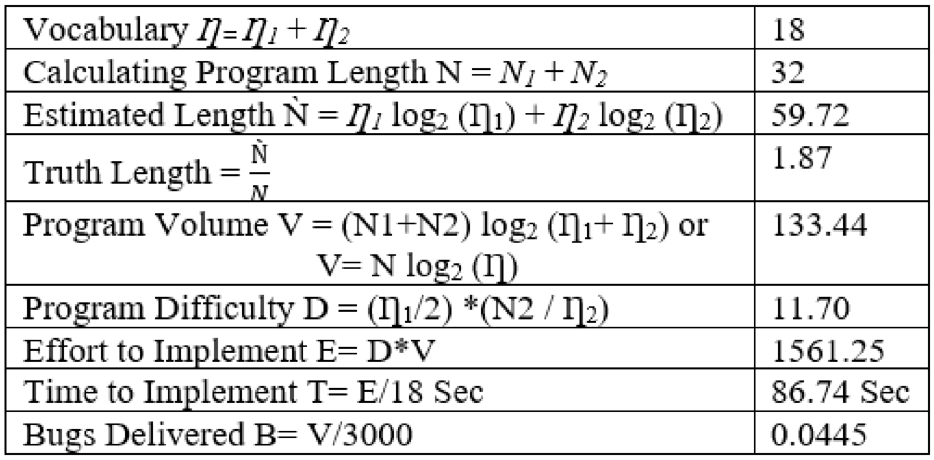
Figure 3: Sequence - C Code

The C program in Figure 3 was tokenized and it generated 13 distinct operators (η_1) with a frequency of 23 (N_1) and 5 distinct operands (η_2) with a frequency of 9 (N_2). When Halstead metric was computed from this data it generated the following results; Program Vocabulary is 18, Program Length is 32, Estimated Length is 59.72, Truth Length is 1.87, Program Volume is 133.44, Program difficulty is 11.7, Effort to Implement is 1561.25, Time to Implement is 86.74 seconds, and Bugs Delivered is 0.0445. Tables 5 and 6, respectively, provide summaries of these metrics.

Table 5: Operands and Operators in a Sequence- C code

Distinct Operators (η_1)	**Frequency (N_1)**	**Distinct Operands (η_2)**	**Frequency (N_2)**
=	5	5	2
;	4	10	2
Return 0	1	sum	1
%d	1	x	2
,	1	y	2
{ }	1		
+	1	***$\eta_2 = 5$***	***$N_2 = 9$***
" "	1		
Int	3		
printf( )	1		
main( )	1		
\n	1		
( )	2		
$\eta_1 = 13$	***$N_1 = 23$***		

Table 6: Halstead Results of Sequence - C code



Vocabulary $\eta = \eta_1 + \eta_2$	18
Calculating Program Length $N = N_1 + N_2$	32
Estimated Length $\grave{N} = \eta_1 \log_2(\eta_1) + \eta_2 \log_2(\eta_2)$	59.72
Truth Length $= \frac{\grave{N}}{N}$	1.87
Program Volume $V = (N1+N2) \log_2(\eta_1 + \eta_2)$ or $V = N \log_2(\eta)$	133.44
Program Difficulty $D = (\eta_1/2) * (N2 / \eta_2)$	11.70
Effort to Implement $E = D*V$	1561.25
Time to Implement $T = E/18$ Sec	86.74 Sec
Bugs Delivered $B = V/3000$	0.0445

A comparison of the lexicographical complexity of the three programming languages of Java, Python and C programming languages was done. The result shows that when using sequence structures, the C programming language is the most complex for all the 9 parameters of the Halstead metrics i.e. C has the most complex vocabulary, has the highest program length and volume, is the most difficult to implement and requires most of the programmer's effort. It also requires most of the implementation time as well as producing the most bugs than Java and Python, followed by Java then Python as summarized in Figure 4. This implies that, for instance, a lot of programmer effort will be required when developing software using C programming language compared to Java, while the least effort will be employed when using Python programming language.

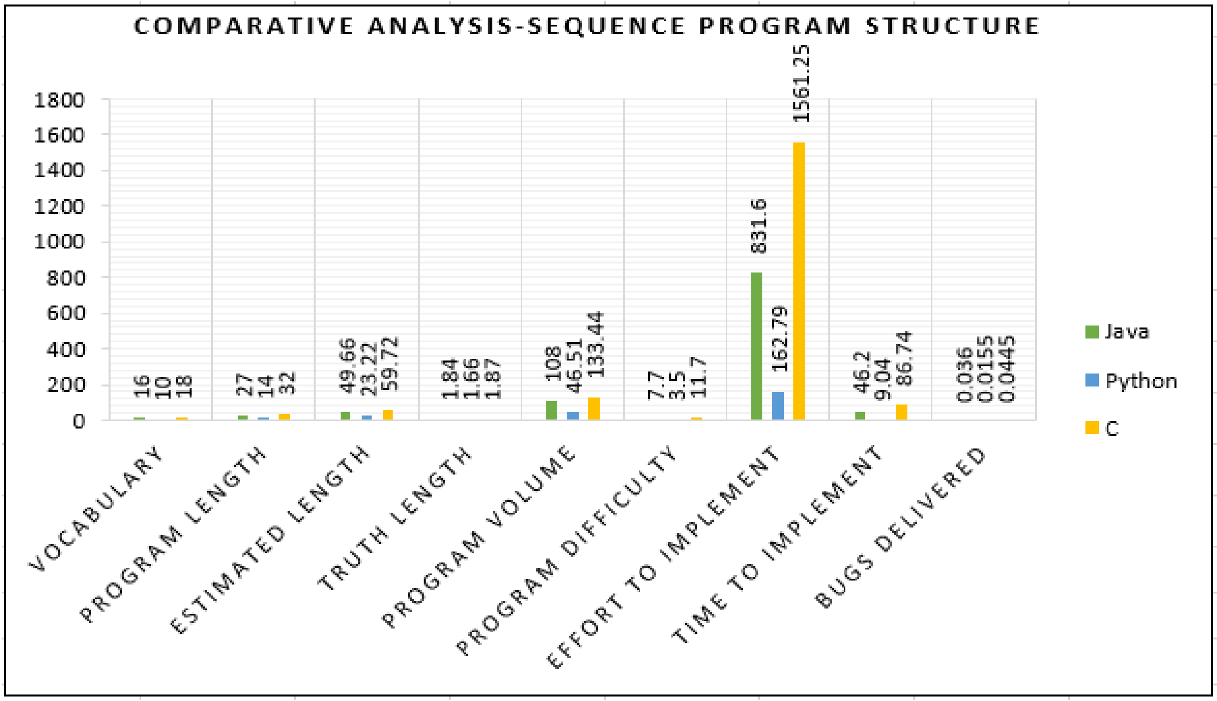


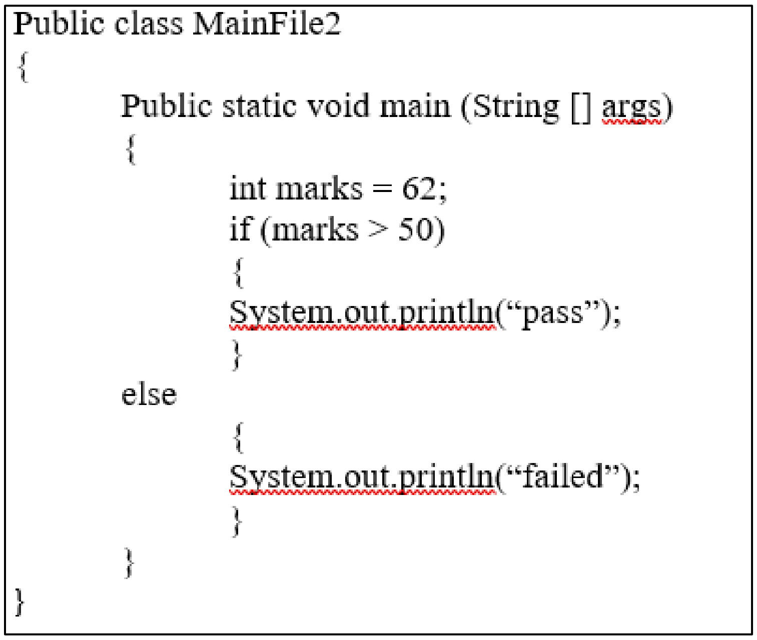
Figure 4: Comparative Analysis of Java, Python and C with Halstead Metric

4.2 Computation of Lexicographical Complexity from a Selection Program Structure

A comparison experimental test was done to compute the lexicographical complexity of source code between Java, Python and C programming language from a simple branching program structure to decide whether a student has passed or failed by accepting input as shown below.

4.2.1 An Implementation in a Java Programming Language

A selection program structure was developed using Java programming language to recommend a student as pass if the marks entered are 50 and above and fail if the marks entered are below 50. The program as shown in Figure 5 was tested and executed in NetBeans IDE to make sure that it executes successfully.



```
Public class MainFile2
{
        Public static void main (String [] args)
        {
                int marks = 62;
                if (marks > 50)
                {
                System.out.println("pass");
                }
        else
                {
                System.out.println("failed");
                }
        }
}
```

Figure 5: Selection - Java Code

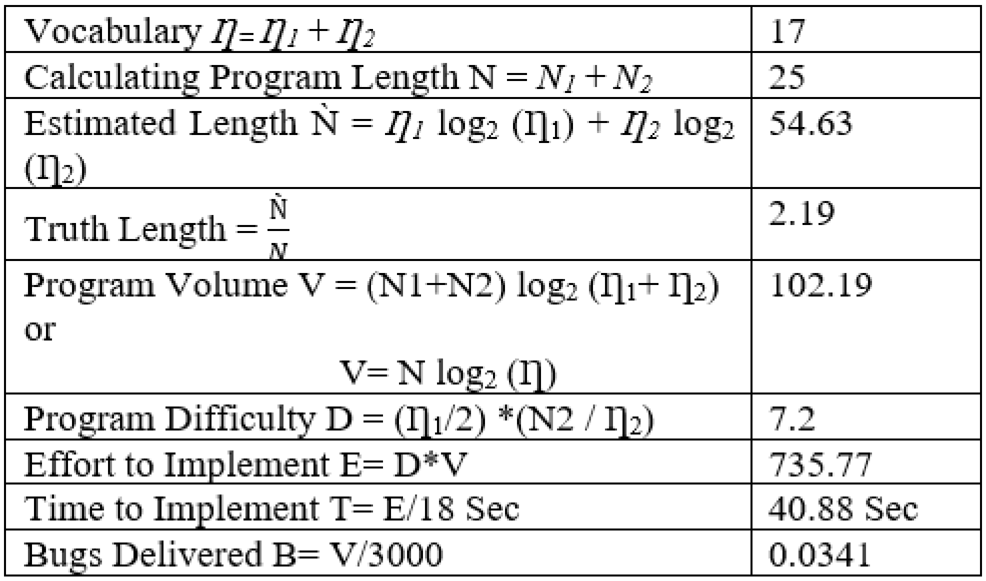
The Java program in Figure 5 has 12 distinct operators (η_1) with a frequency of 19 (N_1) and 5 distinct operands (η_2) with a frequency of 6 (N_2). Computation of the Halstead metrics gave the following results; Program Vocabulary is 17, Program Length is 25, Estimated Length is 54.63, Truth Length is 2.19, Program Volume is 102.19, Program Difficulty is 7.2, Effort to Implement is 735.77, Time to Implement 40.88 seconds, and Bugs Delivered are 0.0341. The results of this program are summarized in Tables 7 and 8 respectively.

Table 7: Operands and Operators in a Selection- Java code

Distinct Operators (η_1)	**Frequency (N_1)**	**Distinct Operands (η_2)**	**Frequency (N_2)**
{ }	4	marks	2
=	1	62	1
>	1	50	1
;	3	pass	1
if( )	1	failed	1
Else	1		

public static void main ( )	1	***η_2 = 5***	***N_2 = 6***
string [ ]	1		
Args	1		
system.out.println( )	2		
“ ”	2		
Int	1		
η_1 = 12	***N_1 = 19***		

Table 8: Halstead Results of Java Selection- Java code



Vocabulary $\eta = \eta_1 + \eta_2$	17
Calculating Program Length $N = N_1 + N_2$	25
Estimated Length $\grave{N} = \eta_1 \log_2 (\eta_1) + \eta_2 \log_2 (\eta_2)$	54.63
Truth Length $= \frac{\grave{N}}{N}$	2.19
Program Volume $V = (N1+N2) \log_2 (\eta_1 + \eta_2)$ or $V = N \log_2 (\eta)$	102.19
Program Difficulty $D = (\eta_1/2) * (N2 / \eta_2)$	7.2
Effort to Implement $E = D*V$	735.77
Time to Implement $T = E/18$ Sec	40.88 Sec
Bugs Delivered $B = V/3000$	0.0341

4.2.2 An Implementation in a Python Programming Language

A Python simple selection program structure was developed to grade a student as either pass or fail based on the marks entered. The program as shown in Figure 6 was developed and tested in the Colab cloud-based environment to ascertain that it executes successfully.

```
marks = 62
if (marks > 50):
        print (“pass”)
else:
        print (“fail”)
```

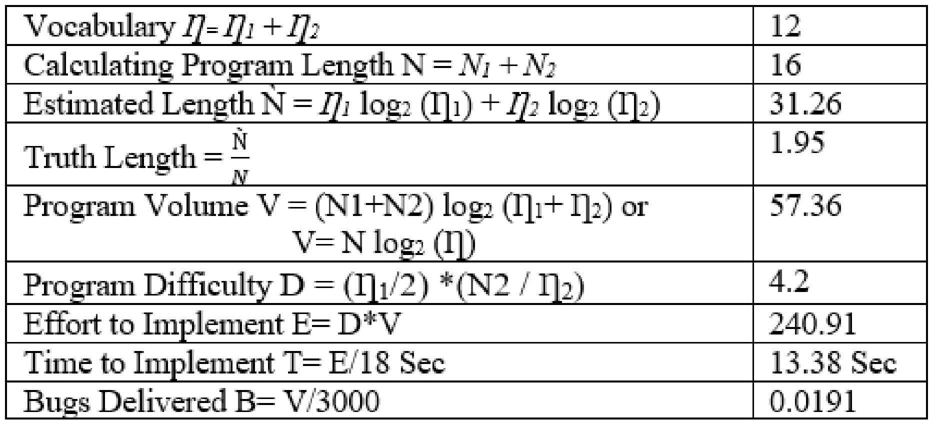
Figure 6: Selection - Python Code

The Python program in Figure 6 has 7 distinct operators (*η_1*) with a frequency of 10 (*N_1*) and 5 distinct operands (*η_2*) with a frequency of 6 (*N_2*). The Halstead metrics of the program are; Program Vocabulary is 12, Program Length is 16, Estimated Length is 31.26, Truth Length 1.95, Program Volume is 57.36, Program Difficulty is 4.2, Effort to Implement is 240.91, Time to Implement is 13.38 seconds, and Bugs Delivered is equal to 0.0191. Tables 9 and 10, respectively, contain an overview of these findings.

Table 9: Operands and Operators in a Selection- Python code

Distinct Operators (η_1)	**Frequency (N_1)**	**Distinct Operands (η_2)**	**Frequency (N_2)**
=	1	marks	2
if( )	1	62	1
Else	1	50	1
:	2	pass	1
>	1	fail	1
print( )	2		
“ ”	2	***$\eta_2 = 5$***	***$N_2 = 6$***
$\eta_1 = 7$	***$N_1 = 10$***		

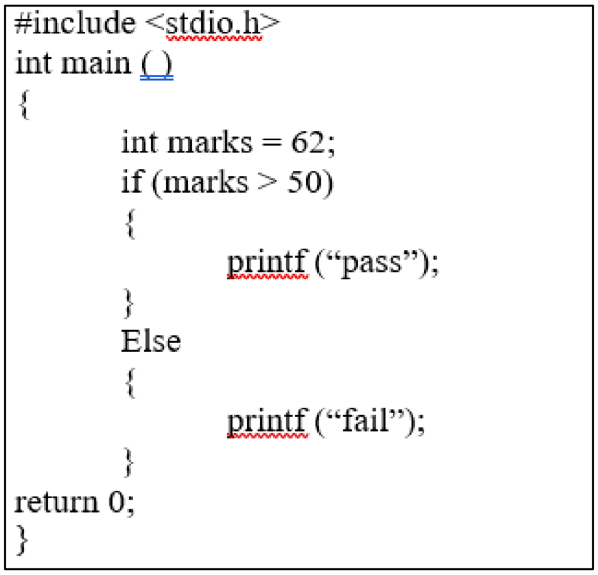
Table 10: Halstead Results of Selection - Python code



Vocabulary $\eta = \eta_1 + \eta_2$	12
Calculating Program Length $N = N_1 + N_2$	16
Estimated Length $\hat{N} = \eta_1 \log_2 (\eta_1) + \eta_2 \log_2 (\eta_2)$	31.26
Truth Length = $\frac{\hat{N}}{N}$	1.95
Program Volume $V = (N1+N2) \log_2 (\eta_1 + \eta_2)$ or $V = N \log_2 (\eta)$	57.36
Program Difficulty $D = (\eta_1/2) * (N2 / \eta_2)$	4.2
Effort to Implement $E = D*V$	240.91
Time to Implement $T = E/18$ Sec	13.38 Sec
Bugs Delivered $B = V/3000$	0.0191

4.2.3 An Implementation in a C Programming Language

NetBeans IDE was used to develop a simple selection C program that grades a student to either pass or fail by comparing the marks entered as shown in Figure 7.



```
#include <stdio.h>
int main ( )
{
        int marks = 62;
        if (marks > 50)
        {
                printf (“pass”);
        }
        Else
        {
                printf (“fail”);
        }
return 0;
}
```

Figure 7: Selection - C Code

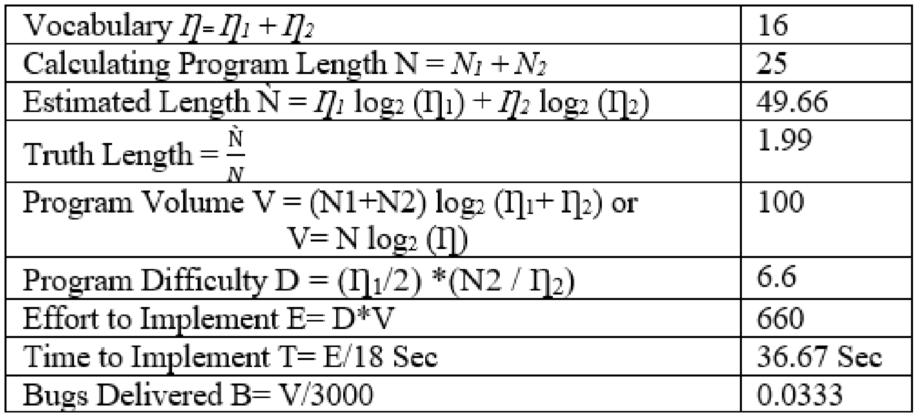
The C program in Figure 7 has 11 distinct operators (η_1) with a frequency of 19 (N_1) and 5 distinct operands (η_2) with a frequency of 6 (N_2). The Halstead metrics of the program are; Program Vocabulary is 16, Program Length is 25, Estimated Length is 49.66, Truth Length 1.99, Program Volume is 100, Program Difficulty is 6.6, Effort to Implement is 660, Time to Implement is 36.67 seconds, and Bugs Delivered is equal to 0.0333. Tables 11 and 12, respectively, presents an overview of these findings.

Table 11: Operands and Operators in a Selection - C code

Distinct Operators (η_1)	**Frequency (N_1)**	**Distinct Operands (η_2)**	**Frequency (N_2)**
{ }	3	pass	1
=	1	fail	1
;	4	marks	2
if( )	1	62	1
>	1	50	1
Else	1		
return 0	1	***$\eta_2 = 5$***	***$N_2 = 6$***
Int	2		

main( )	1	
“ ”	2	
printf( )	2	
$\eta_1 = 11$	***$N_1 = 19$***	

Table 12: Halstead Results of Selection - C code



Vocabulary $\eta = \eta_1 + \eta_2$	16
Calculating Program Length $N = N_1 + N_2$	25
Estimated Length $\grave{N} = \eta_1 \log_2(\eta_1) + \eta_2 \log_2(\eta_2)$	49.66
Truth Length = $\frac{\dot{N}}{N}$	1.99
Program Volume $V = (N1+N2) \log_2(\eta_1 + \eta_2)$ or $V = N \log_2(\eta)$	100
Program Difficulty $D = (\eta_1/2) * (N2 / \eta_2)$	6.6
Effort to Implement $E = D*V$	660
Time to Implement $T = E/18$ Sec	36.67 Sec
Bugs Delivered $B = V/3000$	0.0333

The lexicographical complexity analysis was done to the three programming languages of Java, Python and C for selection program structure. Following the Halstead metrics, the results ranked the lexicographical complexity of the three programming languages and the outcome shows that in selection or branching program structures, Java is the most complex, followed by C and Python is the least complex of the three in regards to the 9 Halstead metrics. The summary of the comparative analysis for the three programming languages is represented in a histogram as illustrated in Figure 8.

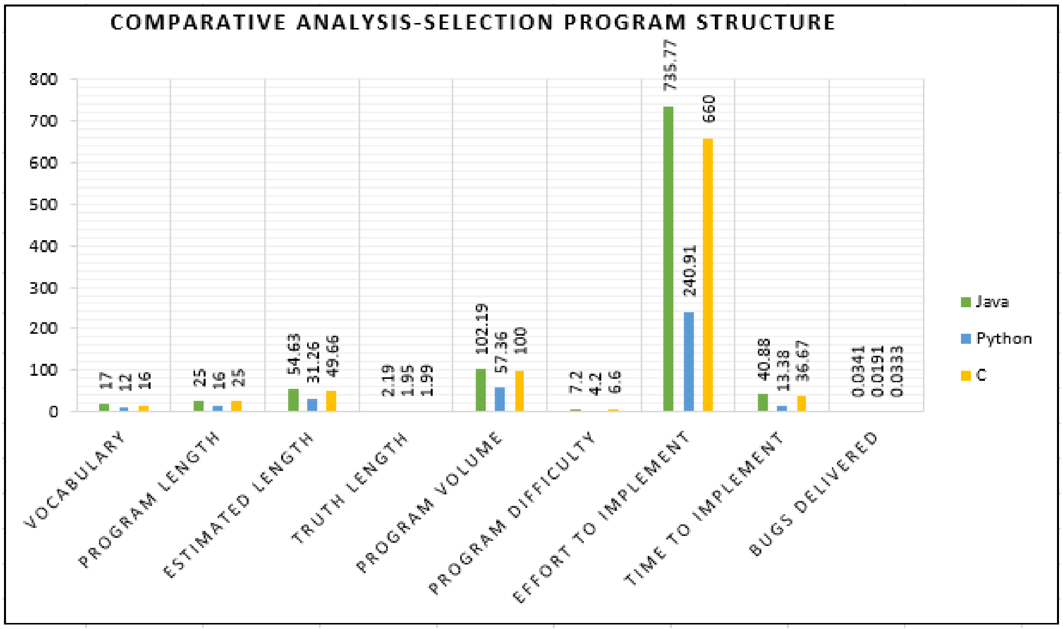


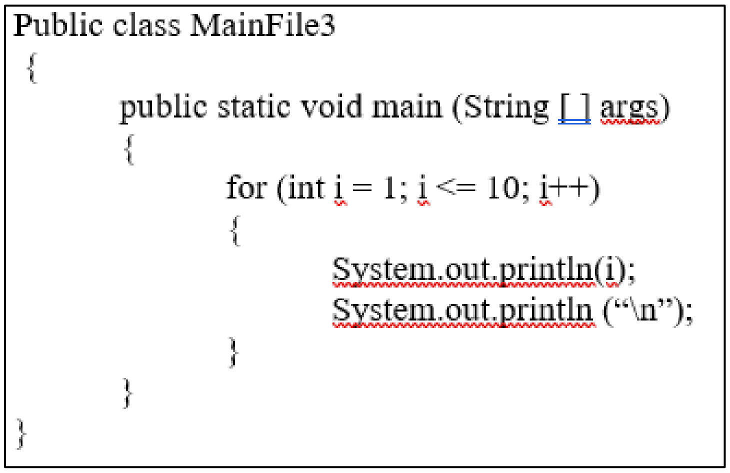
Figure 8: Comparative Analysis of Java, Python and C with Halstead Metric

4.3 Computation of Lexicographical Complexity from a Loop Program Structure

A comparison experimental test was done to compute the lexicographical complexity of source code between Java, Python and C programming language from a simple loop program structure to print the first 10 integers using a FOR statement as shown below.

4.3.1 An Implementation in a Java Programming Language

The first 10 integers were printed using a Java programming language loop program structure. The program as shown in Figure 9 was tested and executed in NetBeans IDE to make sure that it executes successfully.



```
Public class MainFile3
 {
          public static void main (String [ ] args)
          {
                    for (int i = 1; i <= 10; i++)
                    {
                              System.out.println(i);
                              System.out.println ("\n");
                    }
          }
}
```

Figure 9: Loop - Java Code

The tokenization of the Java program in Figure 9 reveals that the program has 13 distinct operators (η_1) with a frequency of 19 (N_1) and 3 distinct operands (η_2) with a frequency of 6 (N_2). The Halstead metrics of the program are computation gave; Program Vocabulary is 16, Program Length is 25, Estimated Length is 52.86, Truth Length 2.11, Program Volume is 100, Program Difficulty is 13, Effort to Implement is 1300, Time to Implement is 72.22 seconds, and Bugs Delivered is equal to 0.0333. Tables 13 and 14 present a summary of these findings.

Table 13: Operands and Operators in a Loop- Java code

Distinct Operators (η_1)	**Frequency (N_1)**	**Distinct Operands (η_2)**	**Frequency (N_2)**
=	1	i	4
< =	1	1	1
;	4	10	1
{ }	3		
for( )	1	***η_2 = 3***	***N_2 = 6***
+ +	1		
\n	1		
“ ”	1		
System.out.println( )	2		
public static void main( )	1		
String [ ]	1		
Args	1		
Int	1		
η_1 = 13	***N_1 = 19***		

Table 14: Halstead Results of Java Loop - Java code

Vocabulary $\eta = \eta_1 + \eta_2$	16
Calculating Program Length $N = N_1 + N_2$	25
Estimated Length $\grave{N} = \eta_1 \log_2 (\eta_1) + \eta_2 \log_2 (\eta_2)$	52.86
Truth Length $= \frac{\grave{N}}{N}$	2.11
Program Volume $V = (N1+N2) \log_2 (\eta_1 + \eta_2)$ or $V = N \log_2 (\eta)$	100
Program Difficulty $D = (\eta_1/2) * (N2 / \eta_2)$	13
Effort to Implement $E = D*V$	1300
Time to Implement $T = E/18$ Sec	72.22 Sec
Bugs Delivered $B = V/3000$	0.0333

4.3.2 An Implementation in a Python Programming Language

A Python simple loop program structure was developed using a FOR statement to output 10 integers starting with 1. The program as shown in Figure 10 was developed and tested in the Colab cloud-based environment to ascertain that it executes successfully.

```
for i in range (1,11):
        print(i)
```

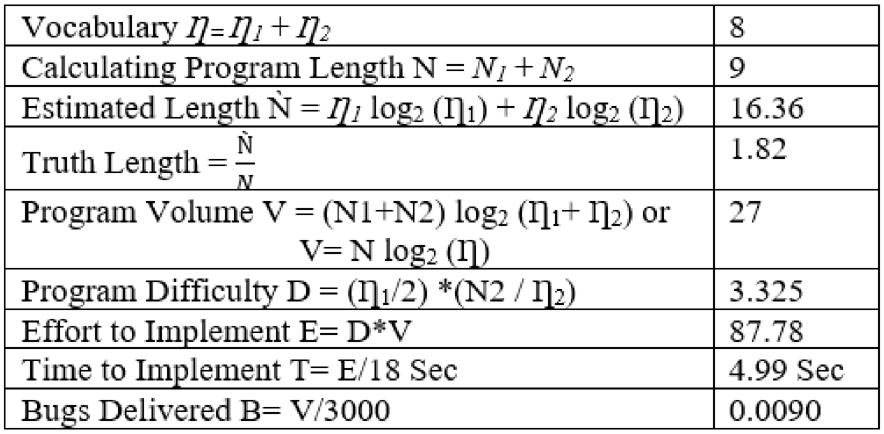
Figure 10: Loop - Python Code

The tokenization of the Python program in Figure 10 reveals that the program has 5 distinct operators (η_1) with a frequency of 5 (N_1) and 3 distinct operands (η_2) with a frequency of 4 (N_2). The Halstead metrics of the program are; Program Vocabulary is 8, Program Length is 9, Estimated Length is 16.36, Truth Length 1.82, Program Volume is 27, Program Difficulty is 3.325, Effort to Implement is 87.78, Time to Implement is 4.99 seconds, and Bugs Delivered is equal to 0.009. Tables 15 and 16, respectively, offer an overview of these findings.

Table 15: Operands and Operators in a Loop - Python code

Distinct Operators (η_1)	**Frequency (N_1)**	**Distinct Operands (η_2)**	**Frequency (N_2)**
:	1	i	2
for	1	1	1
,	1	11	1
in range( )	1		
print( )	1		
$\eta_1 = 5$	***$N_1 = 5$***	***$\eta_2 = 3$***	***$N_2 = 4$***

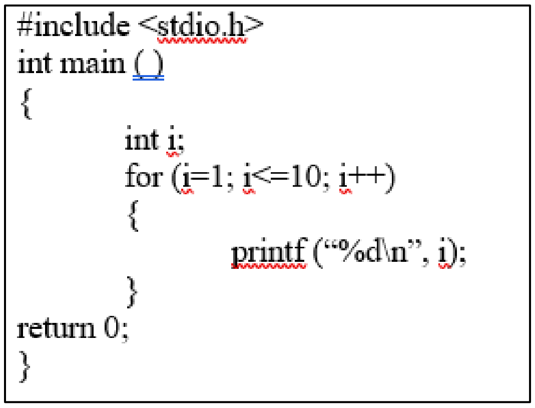
Table 16: Halstead Results of Loop - Python code



Vocabulary $\eta = \eta_1 + \eta_2$	8
Calculating Program Length N = $N_1 + N_2$	9
Estimated Length $\dot{N} = \eta_1 \log_2(\eta_1) + \eta_2 \log_2(\eta_2)$	16.36
Truth Length = $\frac{\dot{N}}{N}$	1.82
Program Volume V = (N1+N2) $\log_2(\eta_1 + \eta_2)$ or V= N $\log_2(\eta)$	27
Program Difficulty D = $(\eta_1/2) * (N2 / \eta_2)$	3.325
Effort to Implement E= D*V	87.78
Time to Implement T= E/18 Sec	4.99 Sec
Bugs Delivered B= V/3000	0.0090

4.3.3 An Implementation in a C Programming Language

NetBeans IDE was used to develop and test a loop program structure in C programming language that outputs the first 10 integers as shown in Figure 11.



```
#include <stdio.h>
int main ()
{
        int i;
        for (i=1; i<=10; i++)
        {
                printf ("%d\n", i);
        }
return 0;
}
```

Figure 11: C-Loop

The tokenization of the C program in Figure 11 reveals that the program has 14 distinct operators (η_1) with a frequency of 20 (N_1) and 3 distinct operands (η_2) with a frequency of 7 (N_2). The Halstead metrics of the program are giving; Program Vocabulary is 17, Program Length is 27, Estimated Length is 58.05, Truth Length 2.15, Program Volume is 110.36, Program Difficulty is 16.31, Effort to Implement is 1799.97, Time to Implement is 100 seconds, and Bugs Delivered is equal to 0.0368. Tables 17 and 18 show the summary of these findings, respectively.

Table 17: Operands and Operators in a Loop - C code

Distinct Operators (η_1)	**Frequency (N_1)**	**Distinct Operands (η_2)**	**Frequency (N_2)**
=	1	i	5
< =	1	1	1
{ }	2	10	1
+ +	1		
;	5		
,	1	$\eta_2 = 3$	$N_2 = 7$
%d	1		
return 0	1		
“ ”	1		
printf( )	1		
\n	1		
for( )	1		
Int	2		
main( )	1		

η_1 = 14	***N_1 = 20***	

Table 18: Halstead Results of Loop - C code

Vocabulary $\eta = \eta_1 + \eta_2$	17
Calculating Program Length N = $N_1 + N_2$	27
Estimated Length Ǹ = $\eta_1 \log_2(\eta_1) + \eta_2 \log_2(\eta_2)$	58.05
Truth Length = $\frac{\grave{N}}{N}$	2.15
Program Volume V = (N1+N2) $\log_2(\eta_1 + \eta_2)$ or V= N $\log_2(\eta)$	110.36
Program Difficulty D = $(\eta_1/2)$ *(N2 / η_2)	16.31
Effort to Implement E= D*V	1799.97
Time to Implement T= E/18 Sec	100.00 Sec
Bugs Delivered B= V/3000	0.0368

A comparison study of the three programming languages was done to analyze their lexicographical complexity. The complexity of each programming language in each of the Halstead metrics is as follows ranked from highest to lowest. The result shows that C is slightly more complex than Java programming language considering the Halstead metrics for the Loops program structure. However, Python programming language has the least complexity for all the 9 parameters of the Halstead metrics as summarized in Figure 12.

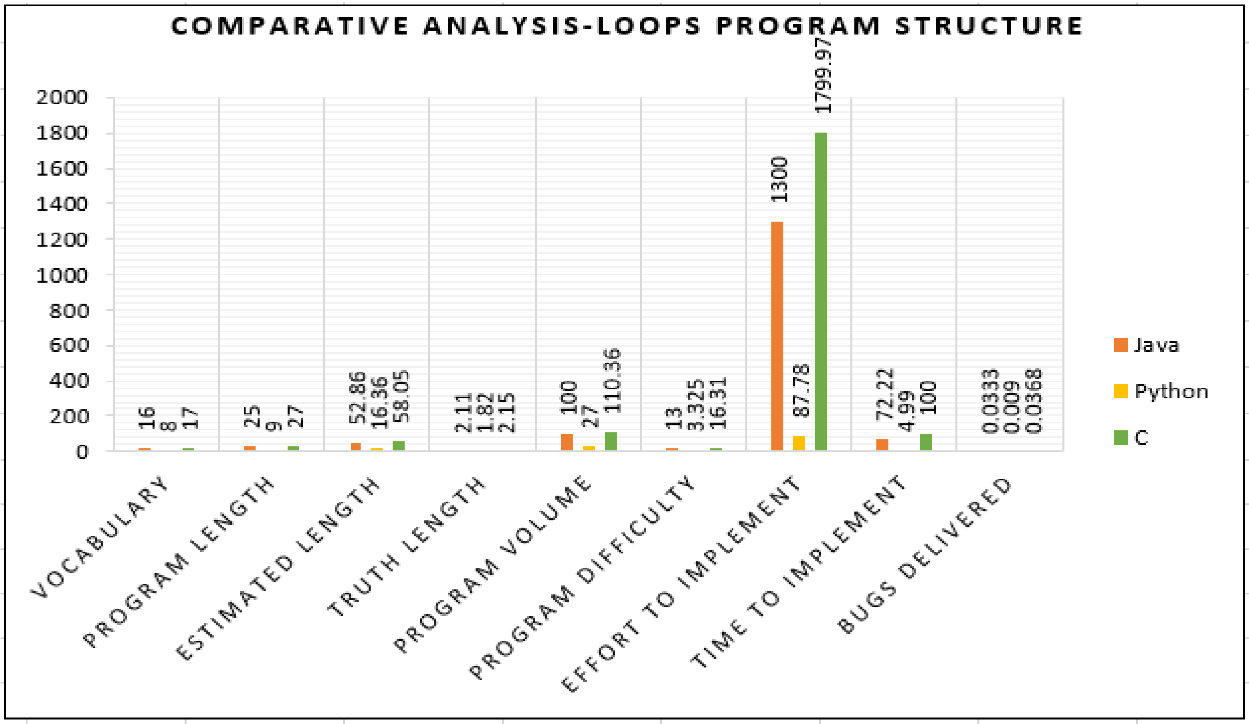


Figure 12: Comparative Analysis of Java, Python and C with Halstead Metric

5. Conclusion and Future Works

Software complexity is one of the most important aspects that every software developer would like to understand before choosing the programming language to use or even developing software.

In this paper, a comparison study was done on the evaluation of software lexicographical complexity levels of three programming languages viz Java, Python and C using Halstead metrics. The results showed that in sequence and Loops program structures, C programming language is the most complex programming language considering all the parameters of the Halstead metrics, for instance, it will require the largest time to implement and more of the programmer effort followed by Java. However, Java proved to be the most complex for selection or branching program structures when subjected to Halsted metric, followed by C. Python on the other hand proved to be the least difficult and also the least complex in all the other Halstead complexity measures.

The findings of this study can help software developers to make important decisions regarding software costing, software quality assurance, and software maintenance, among others.

Future works can focus on predicting software costs using the findings of this study. Also, in the future researchers can consider larger program samples and compare the complexity levels of languages in the different programming paradigms to inform on the choice of a programming language to perform a given task.

References

Abdul Rehman Shaikh. (2020, 18 March) “Applying Halstead Metrics in Your Programs”, https://www.academia.edu/23024048/Applying_Halstead_Metrics_in_Your_Programs/ last accessed on 18 March 2020.

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.

Ahmad Johanna, and Salmi Baharom. (2017). Comparison of Software Complexity Metrics in Measuring the Complexity of Event Sequences. In International Conference on Information Science and Applications, pp. 615-624. Springer, Singapore.

Alfadel, M., Kobilica, A., & Hassine, J. (2017, May). Evaluation of Halstead and cyclomatic complexity metrics in measuring defect density. In 2017 9th IEEE-GCC Conference and Exhibition (GCCCE) (pp. 1-9). IEEE.

Antinyan Vard, Staron Miroslaw, and Sandberg Anna. (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.

Binanto, I., Warnars, H. L. H. S., Abbas, B. S., & Sianipar, N. F. (2018, September). Automation processing Halstead metrics application's results. In 2018 Indonesian Association for Pattern Recognition International Conference (INAPR) (pp. 1-4). IEEE.

Binanto, I., Warnars, H. L. H. S., Abbas, B. S., & Sianipar, N. F. (2018, September). Halstead Metric for Quality Measurement of Various Version of Statcato. In 2018 5th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE) (pp. 276-280). IEEE.

D. Y. Pawade, M. Metha, K. Shah, and J. Rathod. (2015). “Python Based Software Complexity Calculator using Halstead Metrics”, Int. J. Adv. Found. Res. Comput. 2, no. Special Issue (NCRTIT 2015), pp. 390-394.

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.

Flater, David, and David Flater. (2018). Software Science Revisited: "Rationalizing Halstead's System Using Dimensionless Units". US Department of Commerce, National Institute of Standards and Technology.

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.

https://www.geeksforgeeks.org/software-engineering-halsteadssoftware-metrics/last accessed on 27 March 2020.

https://www.javatpoint.com/software-engineering-halsteads-softwaremetrics/last accessed on 27 March 2020.

Iwan Binanto, Harco Leslie Hendric Spits Warnars, Bahtiar Saleh Abbas, and Nesti Fronika Sianipar. (2018). "Halstead Metric for Quality Measurement of Various Version of Statcato", In 2018 5th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE), pp. 276-280. IEEE.

James Cooper. (2017). "Java Design Patterns": A Tutorial.

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

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

Maurice Halstead. (1977), Elements of Software Science. Amsterdam: Elsevier North-Holland, Inc. ISBN 0-444-00205-7.

Prabowo, Y. D., Warnas, H. L. H. S., Gaol, F. L., Abdurachman, E., & Soewito, B. (2018, March). Initial research on Halstead's technique for pattern similarity relationship study. In 2018 International Conference on Information and Communications Technology (ICOIACT) (pp. 773-777). IEEE.

Rajib Mall. (2016). "Fundamentals of Software Engineering", Fourth Ed., PHI Learning Private Limited.

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

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.

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.

Yu, S., & Zhou, S. (2010, April). A survey on metric of software complexity. In 2010 2nd IEEE International Conference on information management and engineering (pp. 352-356). IEEE.

Zuse, H. (2019). Software complexity: measures and methods (Vol. 4). Walter de Gruyter GmbH & Co KG.