Asian Journal of Social Science and Management Technology
ISSN: 2313-7410
Volume 4, Issue 2, March-April, 2022
Available at www.ajssmt.com

Comparative Analysis of Complexity of C++ and Python Programming Languages

Balogun M.O[1],

[1](*Department of electrical and Computer Engineering, Faculty of Engineering and Technology/Kwara State University, Malete, Ilorin, Kwara State, Nigeria)*

ABSTRACT: Software complexity is one of the natural systems complexity which measure the rate at which a particular software is difficult to comprehend, edit, understand, manipulate and maintain. With various system interfaces and complex requirements, the complexity of software systems sometimes grows beyond control, making applications and portfolios too costly to maintain and difficult to improve. A complex software if left uncorrected can run rambling in the developed code, leaving behind bloated, cumbersome, incomprehensible and difficult to maintain applications. In this paper, complexity of two object oriented programming languages (C++ and Python) were compared using some existing complexity measures such as lines of code (LOC) and Halstead complexity measures. The results of the comparison showed that Python programming language is less complex than C++ programming language.

Keywords –Information content, Lines of code, Potential operand and operator, Program volume, Software complexity, Software metrics.

---

1. INTRODUCTION

Software complexity measures the level of difficulty in analyzing, designing, maintaining, modifying and testing software ([7]. [3] defines complexity as a measure of the resources consumed by a system while interacting with a software program to complete a task. [3] goes further by saying that if the interacting system is a computer, the complexity can be based on execution time and storage requirements, while on the programmer side, it can be based on difficulty of executing tasks such as writing, debugging, testing and updating the software. Software metric is applicable in the determination of the complexity of codes written in any programming languages. However, these metrics must be independent of their implementation and statically calculated from the code [2].

Software complexity is the all-embracing notion which are factors that decide the level of difficulty in developing software [10]. With multiple system interfaces and complex requirements, the complexity of software systems sometimes grows beyond control, rendering applications and portfolios overly costly to maintain and risky to enhance. The software engineering discipline has established some common measures of software complexity such as Lines of Code, Halstead Programming volume, cyclomatic complexity measure and so on.

Source Line of Code (SLOC): This metric is used to measure the quantitative characteristics of program source code. It is based on counting the lines of the source code. The original purpose of its development was to estimate man-hours for a project/code. LOC is usually represented as:

- kLOC: thousand lines of code
- mLOC: million lines of code

Halstead Software Measure was introduced by [6]. It is made up of suite of metrics is known as Halstead software science or as Halstead metrics. Most of the existing metrics deal with only one particular aspect of a software product. In contrast, Halstead set of metrics are applicable to software development as well as to overall production effort [2].
Halstead metrics are based on the following indices:

- n1 distinct number of operators in a program
- n2 distinct number of operands in a program
- N1 total number of operators in a program
- N2 total number of operands in a program
- n1' number of potential operators
- $n1'$number of potential operands

Halstead refers to $n1'$ and $n2'$as the minimum possible number of operators and operands for a module or a program respectively. This minimum number would occur in a programming language itself, in which the required operation already existed (for example, in C language, any program must contain at least the definition of the function main ()), possibly as a function or as a procedure; in such a case, $n1'$=2, since at least two operators must appear for any function or procedure: one for the name of the function and one to serve as an assignment or grouping symbol. $n2'$represents the number of parameters without repetition, which would need to be passed on to the function or the procedure [12].

Information Content (IC): is the scientific study of communication, quantification and storage needed for digital information. In efficient coding, information communication-coding theory based on probabilities of occurrence assigns short codes to events with little information content and long codes to events with high information content, thereby provides direct relationship of code size to amount of information content [8]. Determination of the information content of a code involves the use of information theory which is mathematical approach to the study of coding information. The three aforementioned methods were used to measure the complexity of linear search array code written in C++ and Python programming languages in this work.

2. LITERATURE REVIEW

[4] carried out a thorough study to determine whether or not the automatic measurement of source code complexity is possible. A tool for automatic measurement of source code complexity is implemented and it was proved that automatic complexity measurement is achievable. Detail description about some selected metrics such as; Cyclomatic Complexity and Halstead metrics was also presented in this work.
[9] formulated a complexity metric for Python language and made assertion that since Python is an object oriented language, the formulated metric is capable to evaluate any object-oriented language. The metric was validated using; case study, comparative study and empirical validation. The case study is in Python, Java and C++ and the results proved that Python is less complex than other object-oriented languages. Later, validation of the metric empirically with a real project, which is developed in Python was also carried out.
[10] studied the current state of practice regarding deployment of software metric in software product line (SPL). The researchers pointed out that there are very few metrics that can assess quality in the contest of SPLs. It was further discussed that many metrics definitions are rather inaccurate and reuse of metrics across sub-products has barely taken placed.
[11] developed a model to examine and evaluate the Chidamber and Kemerer (CK) metrics for predictive capability for errors and degeneration. The model was developed based on the Shannon entropy and the result shows that the NASA/Rosenberg threshold risk categorization allows for high level of forecasting

[1] applied a set of common static software metrics to programs written in Rust to assess the verbosity, understandability, organization, complexity, and maintainability of the language. Nine different implementations of algorithms available in different languages were selected. A set of metrics were computed for Rust, comparing them with the ones obtained from C and a set of object-oriented languages: C++, Python, JavaScript, Type Script. To parse the software artifacts and compute the metrics, it was leveraged a tool called rust-code-analysis that was extended with a software module, written in Python, with the aim of comparing the results. The Rust code had an average verbosity in terms of the raw size of the code. It exposed the most structured source organization in terms of the number of methods. Rust code had a better, Halstead Metrics, and Maintainability Indexes than C and C++ but performed worse than the other considered object-oriented languages.

3. METHODOLOGY

In this work, the following complexity metrics were used to determine the complexity of linear search algorithm written in C++ and python programming languages

- Lines of code (LOC)
- Halstead measures
- Information contents

Lines of code (LOC): this is a method of determine the complexity of a code by counting the number of lines in a code. LOC involves counting the number of executable lines contained in a code omitting the comment lines and blank spaces. The higher the number of line contained in a code, the more difficult it is to comprehend and the more complex the code is. Halstead complexity measure was developed by Maurice Howard Halstead in 1977. Halstead measurement relies on program execution and its measures which involves analysis of the operators and operands from the source code. Halstead measure can be used to determine some important information about a code such as; testing time, vocabulary, program volume, programming time, number of bug and so on [5]. The intention of Halsted was to see each program as a group of operators with its related operands. Checking the program/code line by line with intention of distinguishing among operators and operands and also to determine number of operators and operands. Halstead complexity measure is based on the following:

$n1'$ = number of unique or distinct operators, $n2'$ = number of unique or distinct operands,
N_1 = total number of occurrences of operators, N_2= total number of occurrences of operands,
n_1= Number of distinct operators and n_1= Number of distinct operands

Information Content (IC): is the scientific study of communication, quantification and storage needed for digital information. It is determine by finding the inverse of program volume to program difficulty.

3.1 LINEAR SEARCH ARRAY (ARR) FOR BINARY NUMBERS Written in C++ Language

```
/* C++ Program to search an element
in a sorted and pivoted array*/
#include <bits/stdc++.h>
using namespace std;
/* Standard Binary Search function*/
int binary Search(int arr[], int low,
int high, int key)
{
if (high < low)
return -1;

int mid = (low + high) / 2; /*low + (high - low)/2;*/
return mid;
```

```
if (key > arr[mid])
return binary Search (arr, (mid + 1), high, key);

// else
return binarySearch(arr, low, (mid - 1), key);
}
/* Function to get pivot. For array 3, 4, 5, 6, 1, 2
it returns 3 (index of 6) */
int findPivot(int arr[], int low, int high)
{
// base cases
if (high < low)
return -1;
if (high == low)
return low;

int mid = (low + high) / 2; /*low + (high - low)/2;*/
if (mid < high andand arr[mid] > arr[mid + 1])
return mid;

if (mid > low andand arr[mid] < arr[mid - 1])
return (mid - 1);

if (arr[low] >= arr[mid])
return findPivot(arr, low, mid - 1);

return findPivot(arr, mid + 1, high);
}
/* Searches an element key in a pivoted
sorted array arr[] of size n */
int pivoted Binary Search(int arr[], int n, int key)
{
int pivot = find Pivot(arr, 0, n - 1);

// If we didn't find a pivot,
// then array is not rotated at all

if (pivot == -1)
return binary Search(arr, 0, n - 1, key);

// If we found a pivot, then first compare with pivot
// and then search in two subarrays around pivot
if (arr[pivot] == key)
return pivot;

if (arr[0] <= key)
return binary Search(arr, 0, pivot - 1, key);

return binary Search(arr, pivot + 1, n - 1, key);
}
```

```
/* Driver program to check above functions */
int main()
{
// Let us search 3 in below array
int arr1[] = { 5, 6, 7, 8, 9, 10, 1, 2, 3 };
int n = sizeof(arr1) / sizeof(arr1[0]);
int key = 3;

// Function calling
cout << "Index of the element is : "
<< pivoted Binary Search(arr1, n, key);

return 0;
}
```

Table 1: The Various Operators and Operands Used in Coding the Linear Search Array for Binary Numbers Using C++ Programming Language

Operators	**Occurrences**	**operands**	**Occurrences**
Int	20	X	7
()	1	N	7
,	45	I	-
[]	5	J	-
If	13	Key	14
*	14	Low	16
<	6	2	6
;	25	1	21
For	1	0	6
=	8	3	5
–	1	4	2
{	5	5	2
}	4	6	3
<=	1	7	1
==	4	8	1
//	8	9	2
Return	16	10	1
<<	2	-	-
Pivot	19	-	-
Mid	18	-	-
“	2	-	-
and&	2	-	-
Arr	29	-	-
n_1=22	**N_1=249**	**n_2=15**	**N_2=94**

Halstead Measures are calculated below:

The program length (N)

$$N = N1 + N2$$

$$N = 249 + 94$$

= 343m

The vocabulary (n)

$n = n1 + n2$

$n = 22 + 15$

= 37

The Program difficulty (D)

$D = (n1 \div 2) * (N2 \div n2)$

$= (22/2) * (94/15)$

= 68.933

Program Volume (V)

$V = N\ log2\ n$

$V = 343\ log2\ (37)$

$= 343 * 5.209$

=1786.687

=1786.7m^3

The Effort (E)

E = D * V

= 68.9333 * 1786.7

= 123162.591j(s)

Programming Time T(s)

$T = E/18$

$= 123162.591 / 18$

= 6842.36s

BUGS (B)

$= V/3000$

$= 1.7867 /3000$

= 0.595(m)

The Information Content (IC) is calculated as follows:

$IC = V/D$

$IC = 1786.687/68.933$

= 25.919

3.2 Linear Search Array (Arr) for Binary Numbers Written in Python Programming Language.

```
# Python Program to search an element
# in a sorted and pivoted array

# Searches an element key in a pivoted
# sorted array arrp [] of size n
def pivoted Binary Search (arr, n, key):

pivot = findPivot (arr, 0, n-1);

# If we didn't find a pivot,
if pivot == -1:
return binary Search (arr, 0, n-1, key);
# If we found a pivot, then first
# compare with pivot and then
# search in two subarrays around pivot
if arr [pivot] == key:
return pivot
if arr[0] <= key:
```

```
return binary Search (arr, 0, pivot-1, key);
return binary Search (arr, pivot + 1, n-1, key);

# Function to get pivot. For array
# 3, 4, 5, 6, 1, 2 it returns 3
# (index of 6)
def findPivot (arr, low, high):

# base cases
if high < low:
return -1
if high == low:
return low

# low + (high - low)/2;
mid = int ((low + high)/2)

if mid < high and arr [mid] > arr [mid + 1]:
return mid
if mid > low and arr [mid] < arr [mid - 1]:
return (mid-1)
if arr [low] >= arr [mid]:
  return findPivot (arr, low, mid-1)
  return findPivot (arr, mid + 1, high)
  # Standard Binary Search function*/
  def binarySearch (arr, low, high, key):

  if high < low:
  return -1
    # low + (high - low) /2;
    mid = int ( (low + high) /2)

    if key == arr [mid]:
    return mid
    if key > arr [mid]:
    return binarySearch (arr, (mid + 1), high, key);
    return binarySearch (arr, low, (mid -1), key);

    # Driver program to check above functions */
    # Let us search 3 in below array
    arr1 = [5, 6, 7, 8, 9, 10, 1, 2, 3]
    n = len (arr1)
    key = 3
    print ("Index of the element is : ",
    pivotedBinarySearch (arr1, n, key) )
```

Table2: The various operators and operands used in coding the linear search array for binary numbers Written in Python programming language.

operators	occurrences	Operands	Occurrences
Def	3	N	7
()	-	Key	14
,	45	Low	16
[]	1	2	6
If	13	1	17
<	5	0	4
;	8	-	-
For	1	-	-
=	9	-	-
–	1	5	2
<=	1	High	12
==	4	6	3
Return	15	7	1
Pivot	17	8	1
Mid	18	9	1
#	17	10	1
Arr	25	-	-
n_1=14	**N_1=145**	**n_2=13**	**N_2=85**

The program length (N)

$$N = N1 + N2$$

$$N = 145 + 85$$

$= 230_m$

The vocabulary (n)

$$n = n1 + n2$$

$$n = 14 + 13$$

n = 27

The program difficulty (D)

$$D = (n1 \div 2) * (N2 \div n2)$$

$$= (14/2) * (145/85)$$

= (7 * 1.706)

= 11.942

Program Volume (V)

$$V = N \log 2\, n$$

$$V = 230 \log 2\, (27)$$

$$= 230 * 4.755$$

$= 1093.65m^3$

The effort (E)

$$E = D * V$$

$$= 11.942 * 1093.65$$

E = 13,060.37

Programming Time (T)

$$T(s) = E/18$$

$$= 1471.11/18$$

T(s) = 81.72

= 1hr.22 s

BUG (B)

$B = V/3000$

$= 1093.65/3000$

= 0.365

Information Content (IC)

$IC = V/D$

$= 1093.65/11.942$

=91.580

4. Results and Discussion

Table 3: Calculation of the Complexity of Linear Search Array Written in C++ Using Line of Code (LOC)

Complexity Measure	**Value (s)**
Length (in lines)	79
LOC without comments	67
LOC plus comments	13
Blank lines	14

Table 4**:** The Result Implementation of Some Metrics Using C++ Programming Language

Complexity Measures	**Values**
Program vocabulary	1787
Program length	343
Program difficulty	68.933
Program Effort	123,162.591
Bugs	0.595
Total Lines of codes (LOC)	79
Program Volume	1786.7
Information content	25.919

Table 5: The Result Implementation of the Lines of Codes Metrics Using Python Programming Language

Complexity Measures	Values
Length Of the code	64
LOC without comments	35
LOC plus comments	15

Table6: The Result of Implementation of Some Metrics Calculated in the Linear Search Array Python Programming Language

Complexity Measures	Values
Program Vocabulary	27
Program Length	230
Program Difficulty	11.942
Program Effort	13,060.37
Bugs	0.365
Total Lines of Codes (LOC)	64
Program Volume	1093.65
Information Content	91.580

Table 7: Comparison of the Complexity Measures of C++ and Python Programming Languages

Complexity Measures	C++	Python
Program Vocabulary	1787	27
Program Length	343	230
Program Difficulty	68.933	11.942
Program Effort	123,162.591	13,060.37
Bugs	0.595	0.365

Total Lines of Codes (LOC)	79	64
Program Volume	1786.7	1093.65
Information Content	25.919	91.580

Tables 3 and 5 show the complexity values gotten using LOCs, it can be seen from the tables that python programming language has less lines of code than C++. Tables 4 and 6 also show the complexity values gotten using Halstead complexity measures. Table7 gives the summary of the complexity values gotten from linear search array written in C++ and Python programming languages. It can be seen from the table that the total lines of code for C++ and Python are 79 and 64 respectively. The lower the lines contained in a code, the less complex the code. The program vocabulary produced by C++ is 1787 which is far more than that of Python that is just 27. It can also be observed from the table that C++ generated 343, 68.933, 123,162.591 and 1786.7 for program length, program difficulty, program effort and program volume respectively, while, that of Python are 230, 11.942, 13,060.37, 1093.65 respectively. The higher complexity values produced by C++ programming language shows that it is more complex than Python programming language.

5. Conclusion

Software complexity is a very crucial characteristic that must be taken into consideration by the software engineers and end users. A too complex software will be difficult to evaluate and maintain. Therefore, when embarking on software development, it is advisable to develop the source code using a less complex programming language for better resource management and ease of maintenance of the source code. This work compares the complexity of two object oriented programming languages (C++ and Python), in which the complexity values produced by Python programming language is far lesser than those produced by C++, it is concluded that Python programming language is less complex than C++ programming language.

6. References

[1] Ardito, L., Barbato, L., Coppola, R., and Valsesia, M. Evaluation of Rust code verbosity, understandability and complexity, *Peer Journal of Computer Science, 7, e406*, 2021.

[2] Basci, D., and Misra, S. Measuring and Evaluating a Design Complexity Metric for XML Schema Documents, *Journal of Information Science and Engineering*, *25*, 2021, 1415–1425.

[3] Basili V. R. Qualitative software complexity models: A Summary in Tutorial on Model and Methods for Software Management and engineering, *IEEE Computer Society Press, Los Alamitos, Calif.,* 1980.

[4] Bhatti, H. R. *Automatic Measurement of Source Code Complexity*, A Master Thesis in Computer Science and Engineering, urn:se:Itu:diva-46648, 2011.

[5] Chandra Segar Thirumalai, Hariprasad T, Vidhyagaran G and Seenu K. Software Complexity Analysis Using Halstead Metrics; *International Conference on Trends in Electronics and Informatic*, 2017, 1109 – 1112.

[6] Halstead M. H. Element of Software Science, Operating and Programming Systems Series, *Elsevier North-Holland, Inc. ISBN 0-444-00205*-7, 1977.

[7] Horst Zuse, Criteria for Program Comprehension Derived from Software Complexity Metrics. *IEEE Second Workshop on Program Comprehension*, 1993, 8-16.

[8] Jack Belzer, Information theory as a Measure of information Content; *Journal of the American Society for Information Science*. V 24 (4), 1973, 300-304.

[9] Misra, Sanjay, and Cafer, F., Estimating complexity of programs in python language, *Technical Gazette*, 18(1), 2011, 23–32.

[10] Sascha E-S, Nozomi Y-E, Klaus S., Metrics for Analyzing Variability and its Implementation in Software Product Lines: A systematic Literature Review, *Inform Softw Technol.*, 2019, 106: 1-30.

[11] Selvarani R., Gopalakrishnan Nair TR, Ramachandran M. and Prasad K., software metrics evaluation based on entropy. *In:IGI Global*, 2010, 139-151.

[12] Misra Sanje, Ibrahim Akman and Ricardo Colomo-Palacios, Framework for Evaluation and Validation of software Complexity Measures, *IET Software 6 (4),* 2012, 323-334.

INFO

Corresponding Author: ***Balogun M.O,*** ***Department of electrical and Computer Engineering, Faculty of Engineering and Technology/Kwara State University, Malete, Ilorin, Kwara State, Nigeria.***

How to cite this article: ***Balogun M.O, Comparative Analysis of Complexity of C++ and Python Programming Languages, Asian. Jour. Social. Scie. Mgmt. Tech.2022; 4(2): 01-12.***