Journal, Advances in Mathematical & Computational Sciences
Vol. 10 No. 2, 2022
www.mathematics-computationaljournal.info

Journal of Advances in Mathematical & Computational Sciences
An International Pan-African Multidisciplinary Journal of the SMART Research Group
International Centre for IT & Development (ICITD) USA
© Creative Research Publishers
Available online at https://www.isteams.net/ mathematics-computationaljournal.info
CrossREF Member Listing - https://www.crossref.org/06members/50go-live.html

Performance Evaluation Of The Effect Of Implementation Languages On The Sofware Complexity Of K- Means Algorithm

[1]Lala, O.G., [1]Onamade, A.A., [1]Oduwole, O.A., [1]Sunday, P., [1]Aroyehun, A.A. & [2]Olabiyisi, S.O.
[1]Department of Computer Science, Adeleke University Ede
[2]Department of Computer Science, Ladoke Akintola University of Technology, Ogbomoso
E-mails: [1]onamadeakintoye@gmail.com, [1]olusegun.lala@adelekeuniversity.edu.ng;
[2]soolabiyisi@lautech.edu.

ABSTRACT

In the world of data mining, the k-means clustering algorithm is regarded as one of the most effective and well-liked methods. Although the approach is widely used, it does have certain drawbacks, such as issues with centroids' random initialization, which might result in unforeseen convergence. Moreover, the number of clusters that must be determined in advance for this type of clustering method is what determines the distinct cluster forms and outlier effects. The inability of the k-means algorithm to accommodate different data formats is a basic issue. This work used Halstead Complexity measure to find the software complexity of k- means algorithm. K-Means algorithm was written in C++, C#, and Java programming language. The software complexity of C++, C#, and Java programming language was evaluated using Halstead Complexity measure. The result obtained was compared in order to discover the complexity of all the different implementation languages. Three different codes of K-means algorithm were written in C++, C# and Java programming language. Halstead complexity measure was used to evaluate the different implementation structures of programming languages for comparative analysis of complexity measure. Comparatively, the results showed that Java programming language performed better than C++ and C# in vocabulary of program, estimated program level, effort to generate program and programming time. In this work, it was discovered that Java has the smallest elementary mental discrimination time to construct a program which is 15.426 seconds when compare to the others. Key information about software testability, dependability, and maintainability may be predicted using complexity measurements from computerized source code assessment.

Keywords: Clustering Algorithm, Complexity, Convergence, Source Code, Software, Vocabulary

Lala, O.G., Onamade, A.A., Oduwole, O.A., Sunday, P., Aroyehun, A.A. & Olabiyisi, S.O. (2022): Performance Evaluation Of The Effect Of Implementation Languages On The Sofware Complexity Of K- Means Algorithm. Journal of Advances in Mathematical & Computational Sciences. Vol. 9, No. 2. Pp 61-74. DOI: dx.doi.org/10.22624/AIMS/MATHS/V10N2P6
Available online at www.isteams.net/mathematics-computationaljournal.

1. INTRODUCTION

The attribute or state of being complicated is called complexity, according to Ricardo (2014). To determine what it entails for a program to be complicated is the first obstacle to overcome when trying to comprehend software complexity. When anything has numerous pieces and those parts interact with one another in many ways, a higher level of emergence than the sum of the parts results, this is referred to as something being complex. The only area in which scholars concur is that there is no universally accepted definition of what constitutes "complexity." But it is feasible to characterize what is complicated (Ricardo, 2014). The basic aim of complex systems theory is the investigation of complex relationships at diverse scales. According to Basili (1980), complexity is a measurement of the resources used by a system in interaction with a piece of software to carry out a certain activity. The runtime and storage needed to complete the computation can be used to quantify complexity when the interacting system is a computer. If the interacting system is a developer, complexity is determined by how challenging it is to carry out activities like coding, debugging, testing, or software modification.

The interplay between a program and a programmer operating on the programming chores is sometimes referred to as software complexity (Basili, 1980). Software development entails building a software system based on specifications. Due to the complexity of requirements, software system projects frequently change. In order to better understand user needs or get rid of mistakes, software projects are altered or adjusted. Consequently, it is said that software systems are complicated (Arpna, *et al*., 2012; Edward *et al*., 2007). The process of creating and updating software systems is called a software life cycle. Every activity and product required to create a software application is included in the software life cycle. The intricacy of software systems makes life cycle models useful for helping developers manage it. To make software development processes more noticeable and controllable, life cycle models reveal these processes and their dependencies (Bruegge et al., 2012); Bruegge et al., 2014; Bruegge and Allen, 2010; Charles et al., 2006).

As a result, achieving a high level of equality is challenging. Since it was understood that software development is a difficult process, software metrics have been a crucial tool. Software quality has been a growing need for decades as a result of its complexity, and several meanings have appeared for software quality during the course of the development of software. Several quality characteristics, including accuracy, dependability, effectiveness, efficiency, integrity, usefulness, maintainability, testability, portability, reusability, flexibility, and interoperability, should be present in a software product (Chin-Yu et al., 2012; Norma and Biemman, 2014). Software complexity is defined as "the extent to which a system or component has a design or implementation that is difficult to comprehend and confirm" (IEEE Std, 1998), meaning that the complexity of a piece of code is directly related to how easy it is to understand. Complexity is caused by all the elements that make a program challenging to comprehend. Software complexity also serves as a gauge for the time and effort required to create, comprehend, and maintain a piece of code; the more complicated the code, the longer those times and efforts will take (Kakesh and Gurvinder 2011).

Results based on actual items demonstrate a relationship between the system's complexity and the amount of failures. A software system's complexity is a measurement of the resources it uses when its component elements work together to complete a job. Complexity is connected to the amount of time and hardware capabilities needed to complete the job if the engaging entity is a computer. The complexity of an interaction depends on how challenging it is to code, test, and alter the software, assuming the interacting element is a developer (Edward 2014). It is considered that a greater level of code comprehension is necessary for creating and altering software systems. Higher comprehensibility means a lower level of program complexity, which makes testing simpler. Given that complexity is a term with several interpretations, complexity and program length have a high correlation. Maintainability is the most important aspect of software quality (Jyoti and Rajder 2017). The code should be clear to developers so they can maintain a software system effectively. In a nutshell, it is crucial to reduce complexity in order to produce high quality. Program metrics are employed to address software complexity.

The quantity of resources used during computing is guaranteed by a dynamic complexity measure, whereas a static complexity measure quantifies the size or structural complexity of an algorithm description, such as the number of nesting do loops Nouh, et al (2017). Metrics are complexity indicators since they highlight a number of flaws in a complicated software system. Software metrics are therefore essential to the software development process since they allow for the estimation of quality. A number of software attributes are quantified using metrics for software complexity. Without employing any measurements, it is often very difficult to produce high-quality programs or to streamline the development process. Multiple metrics exist, each focused on a different complexity component (http://www.stsc,hill.af.mil/resources/tech docs/gsam3). To gauge the quality of its software systems, prominent corporations like Hewlett-Packard, AT&T, and Nokia utilize a variety of criteria (Sommerville, 2004).

Software metrics are one type of measuring system that may be used to enhance software development procedures and software outcomes (Norma, 2014). They offer numerical data on the improvement and verification of software development procedures (Ignaaro et al., 2015). You cannot manage what you cannot measure, according to (DeMarco, 1986), which is why software metrics are used. Measurement is necessary to keep track of and enhance software quality. Software complexity is described by McCabe et al. as "one branch of program metrics that is centered on precise measurement of software properties, as opposed to indirect software measurements such as project milestone status and reported system faults" (McCable and Waston, 2010; Olabiyisi, *et al.,* 2013). Metrics fall under the control and predictor categories. Software processes are the subject of control metrics, whereas software products are the subject of predictive metrics. Estimates for control metrics include effort, time, and faults. Conversely, predictor metrics evaluate the amount of structures and characteristics in a code (Bernard2012).

2. METHODOLOGY

The Halstead complexity metric is used in this research to examine the software complexity of the K-Means algorithm implemented in three distinct programming languages. The following approaches will apply: K- Means algorithm was written in C++, C# and Java programming languages, evaluate of the software complexity of the different K-Means algorithm programming languages was carried out as well as the comparison of the three programming languages under consideration.

3. MOTIVATION OF STUDY

Comparing numerous criteria including effort, time, maintenance, cost, dependability, and comprehension is a common practice in software metrics. Metrics are essential for a number of reasons, including evaluating a program's readability, testability, maintainability, and development procedures. One of the most potent and well-liked data mining methods in the scientific community is the K-means clustering algorithm. Nevertheless, despite its widespread use, the technique has certain drawbacks, such as issues with centroids' random initialization, which causes unanticipated convergence. In addition, the user must first choose k (the number of nodes). Only numerical data may be handled by K-means. K-means makes the assumption that each cluster has about equal quantities of observations and that we are dealing with spherical clusters. The inability of the k-means algorithm to accommodate different data formats is a basic issue. Any clustering study must always provide the value of k, which is dependent on the K-means technique. Different k numbers for clustering will eventually provide various outcomes.

4. SOFTWARE COMPLEXITY OF C++ USING HALSTEAD COMPLEXITY MEASURE

The software complexity measure is as shown in Table 1. It contains the list of operators and operands for C++

Table 1: The list of operators and operands for C++ .

Operators	Occurrences
n1	N1
=	17
++	8
<	11
+	1
-	1
==	1
abs()	2
Operands	Occurrences
n2	N2
3	1
100000	1
10	1
1	4
0	10
25	3
100	4
K	5
I	30
J	15
numbers	7

Operators	Occurrences
kvals[]	3
prevKvals[]	1
steps	3
addition[][]	5
count	1
groups[][]	2
Min	3
groupnum	3
value	2
Sum	1
Ok	2
nums[]	5
n2 = 24	N2 = 112

This is the corresponding calculation for C++ :

N=N1+N2=153 in this case when N1=41 and N2=112.

Program vocabulary: n1+n2=7+24=31

Volume V = N* $\log_2$ n
=153*$\log_2$ 31=757 bits.

The estimated program length N of the program
= 7 $\log_2$ 7+24 $\log_2$ 24
= 7*2.81+24*4.58
= 19.67+109.92=129.59

Estimated program level

L= 2/n1*n2/N2=2/7*24/112= 0.061

V*= V*L=757*0.061=46.177

E= V/L= D*V

=757/0.061=12409.83

Therefore, 12409.83 is the elementary mental discrimination are required to construct the program.
T= E/B= 12409.83/31= 400.317 seconds = 7minutes

5. SOFTWARE COMPLEXITY OF C# USING HALSTEAD COMPLEXITY MEASURE

Table 2 provides the software complexity measurement. It includes a list of C #operators and operands.

Table 2: The list of operators and operands for C#

Operators	**Occurrences**
n1	N1
=	9
<	2
++	1
*	4
/	1
+	2
Operands	Occurrences
n2	N2
seed.Count	1
1	1
6	1
R	1
X	3
Y	3
I	3
Operators	**Occurrences**
rand	4
new Random();	1
max_r	3
num_points	2
seed_num	3
Theta	3
Seeds[]	2
N2 = 14	31

This is the corresponding calculation for C# :
The length of the program is N=N1+N2=50 where N1=19 and N2=31.
N1+N2=6+14=20 is the program's vocabulary
Volume $V = N^* \log_2 n$
$=50^*\log_2 20=216.09$ bits.

The estimated program length N of the program
$= 6 \log_2 6+14 \log_2 14$
$= 6^*2.584+14^*3.807$
$= 15.504+53.298=68.802$

Estimated program level
L= 2/n1*n2/N2=2/6*14/31= 0.150
V*= V*L=216.09*0.150=0.324
E= V/L= D*V
=216.09/0.150=1440.6

Therefore, 1440.6 elementary mental discrimination are required to construct the program.
T= E/B= 1440.6/20=72.03 seconds = 2minutes

6. SOFTWARE COMPLEXITY OF JAVA USING HALSTEAD COMPLEXITY MEASURE

Table 2 displays the software complexity metric. It includes a list of the Java operators and operands.
Table 3: The list of operators and operands for Java

Operators	Occurrences
n1	N1
<	7
+	1
=	11
{	4
}	4
++	1
;	10
,	9
Operators	Occurrences
.	7
:	1
//	1
n1 = 11	N1 = 56
Operands	Occurrences
n2	N2
Double sum	1
Centroid	1
Record	3
Distanc	4
entry se	1
get value()	1
return s	1
1000	1
n2 = 8	N2 = 13

This is the corresponding calculation for Java:
N1 is 56 here, while N2 is 13. N=N1+N2=69 is the program length.
Program vocabulary: n1 + n2 = 11 + 8 = 19

Volume $V = N^* \log_2 n$
$V=69^*\log_2 19= 293.10$ bits.

The estimated program length N of the program
$= 11 \log_2 11+8 \log_2 8$
$=11^*3.4594+8^*3$
$= 38.0534+24=62.0534$

Estimated program level
$L= 2/n1^*n2/N2 = 2/11^*8/13= 0.01$
$V^*= V^*L=293.10^*0.01=2.931$
$E= V/L= D^*V$
$=293.10/0.01=293.1$

Therefore, 293.1 elementary mental discrimination are required to construct the program.
$T= E/B= 293.1/19=15.426$ seconds = 1minute

7. RESULTS AND DISCUSSION

Having evaluated the software complexity of K- means algorithm using Halstead complexity measure, three (3) different codes of K-means algorithm written in C++, C# and Java programming language were analyzed. Halstead complexity measure was used to evaluate different implementation structures of programming languages for comparative analysis of complexity measure. Halstead Complexity Measure written in C++ is depicted in Table 4 whereas Halstead Complexity Measure written in C# is depicted in Table 5 and Table 6 shows the Complexity of Halstead Complexity Measure written in Java and Table 4.4: The Complexity Comparative of Halstead Complexity Measure written in C++, C# and Java.

Table 4: The Complexity of Halstead Complexity Measure written in C++

Complexity	Values
The program length	153
Vocabulary of program	31
Volume of program	757
Estimated program length	129.59
program level	0. 061
Effort to generate program	12409.83
Programming Time	400.317

Table 5: The Complexity of Halstead Complexity Measure written in C#

Complexity	Values
The program length	50
Vocabulary of program	20
Volume of program	216.09
Estimated program length	68.802
Estimated program level	0.150
Effort to generate program	1440.6
Programming Time	72.03

Table 6: The Complexity of Halstead Complexity Measure written in Java

Complexity	Values
The program length	69
Vocabulary of program	19
Volume of program	293.10
Estimated program length	62.0534
Estimated program level	0.01
Effort to generate program	293.1
Programming Time	15.426

Table 7: The Complexity Comparative Table of Halstead Complexity Measure written in C++, C# and Java.

Complexity	C++	C#	Java
The program length	153	50	69
Vocabulary of program	31	20	19
Volume of program	757	216.09	293.10
Estimated program length	129.59	68.802	62.0534
Estimated program level	0. 061	0.150	0.01
Effort to generate program	12409.83	1440.6	293.1
Programming Time	400.317	72.03	15.426

8. RECOMMENDATION

This research compares the difficulty of software developed in C++, C#, and Java using the Halstead complexity measure and the K-Means technique. It is recommended that Java programming language gives the smallest elementary mental discrimination time required to construct the program.

9. CONCLUSION

Numerous software attributes are quantified using software complexity measures. The automated analysis of the source code may be used to forecast important information about the software systems' testability, reliability, and maintainability using complexity metrics. It was found that java has the smallest elementary mental discrimination time to construct a program which is 15.426 seconds, compare to the others. While C++ has the largest or complex elementary mental discrimination time to construct a program compare to the other programming languages which is 400.317 seconds.

REFERENCES

1. Ricardo, A. (2014) A Unified Complexity Theory pp.12 DOI:10.13140/2.1.2072.8963.
2. Amit, J. and Kumar R. (2015): A New Cognitive Approach to Measure the Complexity of Software. International Journal of Software Engineering and its Applications Volume 8, No. 7 pp. 185
3. Aprna, T., Dharmender S., & Arun, K. (2012): Software Change Complexity: A New Dimension for Analyzing Requested Change: IJCA proceeding on International Conference on Recent Trends in Information Technology and Computer Science 2012.
4. Basei, D., and Mistra, S. (2009): Measuring and Evaluation a Design Complexity Metric for XML
5. Boehm, B. (1978); Characteristics of software quality, I of TRW Series on Software Technology, North-Holland, Amsterdam Holland.
6. Briand, L., Emam, K. E., & Morasca, S. (1995): Theoretical and Empirical Validation of Software Metrics. International Software Engineering Research Network Technical Report ISERN-95-03.
7. Chin-Yu, H., Hareton, L., & Osamu, M. (2012): Software Quality Assurance Methodologies and Techniques.
8. Chis, F. (2008)CiteSeerX citation query". CiteSeerX: Introduction to Algorithms-The College of Information Sciences and Technology at Penn State. http://citeseerx.ist.psu.edu/showciting?cid-1910. Retrieved 2009-09-01.
9. DeMarco, T. (1986): Controlling Software Projects, Yourdon Press, New York, 1986.
10. Donald K.. (2003): The Art of Computer Programming (second edition), 3:418.
11. David, A., Bodo, M,, & Roglin, H. (2009) k-means has polynomial smoothed complexity. In Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on, pages 405–414. IEEE, 2009.
12. Eric, O., Michael, O., Francis, E., & Samuel, E. (2021) Comprehensive Review of K-Means
13. Clustering Algorithms. http://dx.doi.org/10.31695/IJASRE.2021.34050
14. Fenton, N.,and Pfleeger, S. L. (1997): "Software Metrics - A Rigorous and Practical Approach", 2nd Edition Revised ed. Boston: PWS Publishing.
15. Fischer, S. R.. (2003): A history of language, Reaktion Books, ISBN 186189080X: 205
16. Francalanci, C. and Merlo, F. (2010): The Impact of Complexity on Software Design Quality and Costs: An Explanatory Empirical Analysis of Open Source Applications.
17. Halstead, M. H. (1977): Elements of Software Science, Operating and Programming Systems Series, Elservier Computer Science Library North Holland N. Y. Elsevier North-Holland, Inc. ISBN 0-444-00205-7.
18. Horowitz, E. and Sahni, J. (1978): Fundamental of Computer Algorithms, Computer Science Press, Inc, Maryland, U.S.A.
19. H. Chen, "Comparative Study of C++ C# and Java Programming Languages", Vaasan Ammattikorakeakoulu Vasa Yrkeshogskola university of applied sciences Information Technology, 2010.
20. Nouh, A. (2017) (IJASCA) International Journal of Advanced Computer Science and Application Volume 8, No. 8, 2017.

21. Olabiyisi S. O. Adetunji A. B. and Olusi T. R. (2013): Using Software Requirement Specification as Complexity Metrics for Multi-Paradigm Programming Languages International Journal of Emerging Technology and Advanced Engineering Website: www.ijetac.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 3, Issue 3, March 2013.)
22. Schneidewind, N. F. (1992): Methodology for Validating Software Metrics, IEEE Transactions on Software Engineering, 18:410-422.
23. Steven W. Smith (2011): Execution Speed: Programming Tips Copyright © by California Technical Publishing
24. Steve, K. (2013): Complexity Metrics and Difference Analysis for Better Application Management Pg 1-33 Bill Hewlett, Hewlett-Packard.
25. Sudhir, S., Nasib, S. (2013) Analysis and Study of K-Means Clustering Algorithm. Gill Deptt of Computer Science & Applications M. D. University, Rohtak, Haryana. International journal of engineering research and technology (IJERT)
26. Tian. J., Zelokowitch, M.V (1995): Complexity Measure Evaluation and Selection, IEEE Transactions on Software Engineering 21(8): 641-650.
27. TOBE Software (2010): The Coding Standards Company. Programming Community
28. Index.www.stumbleupon.com/url/...tiobe.../index.../tpci_definition.h... – Coched
29. Veenendaal, E. V., McMullan, J. (1997): Achieving Software Product Quality, Den Bosch, UTN Publishers, Amsterdam, the Netherlands.
30. Weyuker, E. J. (1988): Evaluating Software Measures, IEEE Transaction of Software Engineering, 14:1357-1365.
31. Yahya, T, Mohammed, A., Bassam A. (2014): The Correlation among Software Complexity Metrics with Case Study: International Journal of advanced Computer Research (ISSN (print): 2249-7277 ISSN (online:7970) Volume 4, No. 2 Issue 15 June 2014.
32. Zuse, H. (1991): Software Complexity: Measures and Methods: 605, 498 figures. Berlin, New York: DeGruyter.
33. Zuse, H. “Software complexity: measures and methods. “ ,Vol. 4. Walter de Gruyter GmbH & Co KG, 2019.