PERFORMANCE EVALUATION OF IMPLEMENTATION LANGUAGES ON COGNITIVE COMPLEXITY OF DIJKSTRA ALGORITHM

Isah O. Mustapha [1], Stephen Olabiyisi [2], Rasheed G. Jimoh [3], Maruf O. Alimi [1]

[1] Department of Physical Sciences, College of Natural Sciences, Al-Hikmah University, Ilorin, Nigeria
[2] Department of Computer Science and Engineering, Ladoke Akintola University of Technology, Ogbomosho, Nigeria
[3] Department of Computer Science, Faculty of Communication and Information Sciences, University of Ilorin, Ilorin, Nigeria

Corresponding author: Maruf Alimi, moalimi@alhikmah.edu.ng

ABSTRACT: Maintainability is a key factor in measuring the quality of developed software and it becomes important due to dynamism of software. Partially, maintainability is a function of source code understandability on the part of developers. Therefore, cognitive complexity of software is relevant to its maintainability. In fact it is not an overemphasis to state that, quality of software in general can hardly be control if the code is complex (Banker, Datar and Zweig, 2009;francalanci and Merlo, 2010). Hence as a result of strong impact that cognitive complexity has on the software quality this research work investigates the effect of some implementation languages on cognitive complexity. Three earlier and recent implementation languages were sampled in term of Procedural Programming Languages and Object Oriented Languages then implemented on a unique algorithm and appraised using Procedural Cognitive Complexity Metric [P.C.C.M.] and Multiparadigm Cognitive Complexity Metrics [M.C.C.M.] respectively. The experiment results have shown that among the procedural programming languages, Fortran has least cognitive complexity with sixty six while among Object Oriented Languages C++ has the least with one hundred and thirty eight. Cross assessment of Fortran and C++ using both [P.C.C.M.] and [M.C.C.M.] reveal that Fortran has the least cognitive complexity among all the implementation languages used. The research results has shown that Fortran 77 is the best for implementation of Dijkstra algorithm among the selected languages to have the least cognitive complexity and has reaffirmed that some languages are more appropriate for easy understandability of source code than others.

KEYWORDS: Cognitime metric, Software Complexity, Dijdstra Algorithm, Objict Programming Language, Procedural Programming Language.

1. INTRODUCTION

Software Complexity is a measure of resources that is expended on software during development and it is factually relevant to many things like cost, processing time, amount of storage facilities required by the software, software testing, source code understandability, maintainability and future improvement of the software. Software complexity analysis tends to provide meaningful information that can be used to identify software structure, critical software components, testing deficiencies, relative risk area within software components and it is use to give an insight into modules where possible program improvements can be achieved ([L+94]).

The study of software complexity can serve as a tool for prediction of program length, program development time, number of bugs and future cost of program maintenance. There are several metrics for measuring the complexity of various software characteristics and cognitive complexity is one of it, each of the metric need to be used appropriately so that it can actually quantify those software characteristics which it meant to access.

Software Cognitive Complex refers to the degree to which a system or component has a design or implementation that is difficult to understand. It measures the ease of comprehending the source code for possible modification and improvement. Programmer need to code their design in the form that can be of least Cognitive complexity so that any other developer can have easy access for modification and improvement. This research work is an attempt to investigate may be the choice of implementation languages really has an impact on the ease of source code comprehension.

2. OBJECTIVE

To evaluate the Cognitive Complexity of Dijkstra Algorithm's source code using different implementation languages.

3. RELATED WORK

Software Complexity and The Programmer

Although characteristics of the program {program characteristics include stylistics, specific programming task, problem domain and programming environment} should be a major yard

stick for measuring specific program complexity but at the same complexity measure should be design with regards to the programmer and the source code as well, Programmer becomes important here since he perform the task of coding, debugging, testing, documentation and modification, and the ease of been able to do that, is relevant to his own expertise, experience and comprehension of the need of the user. It is not an over emphasis, to say that the experimental factor like general knowledge of programming language, techniques of algorithm, specific knowledge of one application area or more and even programming skills develop through practice will make an expert to view factor of programming task difficulty vary to that of a novices. Actually complex algorithm may result in implementation complexity but at the same time the way an algorithm may appear complex to a novice, it may not be like that in the view of an expert because of his quick comprehension.

Software complexity metric should be able to serve as a tool for prediction of program length, program development time, number of bugs, difficulty that is likely to be involved in comprehension of the program and future cost of program maintenance. Hence for developer to be able to use complexity metric to predict future cost of program maintenance, there is need for perfect comprehension of the source code. Like we know life is dynamic and there may be need for constant redesign of an algorithm to meet the environmental challenges which may call for recoding. In such situation, developers need to use less complex source code that any other developer can easily understand and readjust. An Algorithm, which is complex to implement, requires skilled developers, longer implementation time and has a higher risk of implementation errors. Moreover, complicated algorithms tend to demand specialist and they do not necessarily work well when the problem changes ([AN00]).

In order to achieve ease of software source code comprehension, researchers in this domain have continue to search for various parameters that are relevant to source code comprehension and are improving day in day out to ensure the possibility of having unified metric for calculation of cognitive complexity. In that regard, Chhabra, Aggarwal & Singh, ([CAS03]) uses the analysis of distance between module definition and modules call to investigate human cognitive effort for source code understandability. He explained the fact that when module definition is far away from modules call then mental searching is greater than when modules call is at a shorter distance of line of code to module definition (spatial analysis). He proposed a code cognitive complexity by combining structural design of control statement complexity of source code with code"s spatial complexity. By comparism, his research work shows a better analysis of difficulty of comprehension of source code than lines of code metric. Although his metric needs to be validated over cross section of many cognitive metrics for a collection of large program.

Vivanco and Jin ([VJ07]) also carried out a case study research to determine a group of source code metric that can be used to improve the performance of predictive model for detection of component or modules that are likely to have high cognitive complexity or that are probably going to be problematic. The contribution of the research is in the area of making project managers and developers to plan ahead and focus on corrective measure towards software components that may be faulty.

Cafer, ([Caf10]) also formulated a cognitive complexity metric which was used in this research work, the model takes into consideration the characteristics of object oriented programming language as well as that of procedural programming language hence the metric is applicable to both and even applicable to multiparadigm programming language. Comparative analysis of the research work revealed that the metric perform well than some early cognitive complexity metric since it was like an hybridize of the early metric.

4. EXPERIMENT WITH DIJKSTRA ALGORITHM

Dijkstra's Algorithm is about finding the shortest distance connecting two nodes (source and destination node) in a graph. In this situation if we order all the nodes in a graph with respect to their closeness to the starting node then the shortest would have been a member of the ordered list. Consequently each of the ordered list can then be compare to detect the one with minimum weight.

Summarily, Dijkstra's algorithm get a solution to the single-source shortest-paths problem on a weighted, directed graph(digraph) K=(N, E) where the weight of all edges are nonnegative. It is applicable to many practical situation in life, for instance in Network Routine, Marketing and Distribution of goods, Transportation system of an organization, etc. Therefore as a result of its applicability in resource management and its relevance in decision-making, this research uses this algorithm for implementation to determine the appropriateness of programming languages in addressing the issue of cognitive complexity as stated above.

The complexity of Dijkstra Algorithm implementation is researched in to, using six programming languages. C++, C#, and Java were selected from Object Oriented Language and Pascal, C, and FORTRAN 77 were evaluated from

Procedural programming Language. The least complex source code among object oriented language was later recompare with the source code that has the lowest complexity among the procedural languages. The complexity appraisal were done using M.C.C.M., P.C.C.M. and C-procedural Metric.

4.1 METRIC INVOLVED

With reference to Cafer (2010), **M.C.C.M** was calculated by adding the complexity of inherited class (CIclass), complexity of distinct class (CDclass) together with C-procedural Metric
i.e

MCM = CIclass + CDclass + C-procedural Metric

4.1.1 Calculations of Class Complexity (Cclass):

Calculations of Cclass was done base on the weight or number of attributes, variables, objects, methods, structures and Cohesion involved in the class.
i.e

Cclass = W(attributes) + W(variables) + W(structures) + W(Object) - W(Cohesion)

i.e

Cclass = W(att) + W(var) + W(str) + W(Obj) - W(Cohesion)

4.1.2 Weight of Structure:

The value of structure was derived from Basic Control Structure (BCS) of cognitive weight unit (CWU) as follows:

Table 1. BCS for PCCM

Category	BCS	CWU
Sequence	Sequence	1
Condition	If-else Switch	2 2
	sub-if (in nested conditions)	1
Loop	For	3
	for…in while/do…while sub-loop in nested loop	3 3 2
Functional Activity	functional-call alert/prompt/throw event recursion	2 2 2 3
Exception	try…..catch	1

Table 2. BCS for MCCM

Category	CWU
Sequence	1
Condition	2
Nested sub-condition	1
Loop	3
Nested sub-loop	2
Module call Recursion	2 3
Exception	1

The value for Cohesion is derived from Number of Method that uses an Attribute (MA) and Number of Attributes that is used in a method (AM) as follows:

Cohesion = MA / AM

Calculation of CIclass:
Super Class was multiplied by the sum of the classes that are derived from it.
While CDclass was calculated and added.

Calculation of **P.C.C.M**
The following parameters were used for the calculations of PCCM :

a. Number of arbitrarily named variable [ANV] has four units weight.
b. Number of meaningfully Named variable [MNV] has one unit weight
c. Number of operators has one unit per each.
d. Cognitive weights Unit [CWU] of Basic Control Structures [BCS].

Then P.C.C.M adopted from Misra & Akman ([MA10]) is given as

PCCM = ∑∑ (((4*ANV + MNV) + Operator) * CWU)

5. FINDINGS AND DISCUSSION

a) Analysis of Complexity of Dijkstra Algorithm Using Object Oriented Programming Language with the use of M.C.C.M is as follows:

The following tables display the analysis of Cognitive Complexity of C++ implementation language on Dijkstra Algorithm.

Table 3: Complexity Of Dijkstra Algorithm with respect to C++

Class	Att	Str	var	obj	MA	AM	Cohesion	Complexity	TOT Complexity
Class main	5	44	39	0	4	3	1.333	6.67	87

NON Class	Att	Str	var	obj	Function	Cohesion	Complexity	TOT Complexity
C procedural		44	9	0	17	1.889	51	51

FOR C++:
MCM= C.I + C.D + C procedural
MCM= 0 + 87 + 51=**138**
The outcome of the above complexity analysis shows that the cognitive complexity of the source code for dijkstra algorithm is 138 when c++ is used for implementation. It can also be seen that C++ has less complexity because only one class was used.

The following tables display the analysis of Cognitive Complexity of C# implementation language on Dijkstra Algorithm.

Table 2: Complexity Of Dijkstra Algorithm With Respect To C#

Class	Att	Str	var	4	MA	AM	Cohesion	Complexity	TOT Complexity
Class main	0	15	2	1	0	0	0	18	
Class Path	8	91	36	0	5	5	1	134	152

NON Class	Att	Str	var	obj	Function	Cohesion	Complexity	TOT Complexity
C procedural		106	12	1	37	3.08	116	116

FOR C#:
MCM= C.I + C.D + C procedural
MCM= 0 + (18+134) + 116 = 152 + 116 = **268.**
The outcome of the above complexity analysis shows that the cognitive complexity of the source code for Dijkstra Algorithm is 268 when c# is used for implementation. Although C# uses two classes in it implementation but has less attributes and string when compare with java.

The following tables display the analysis of Cognitive Complexity of JAVA implementation language on Dijkstra Algorithm.

Table 5: Complexity Of Dijkstra Algorithm With Respect To JAVA

Class	Att	Str	Var	obj	MA	AM	Cohesion	Complexity	TOT Complexity
Class main	0	15	3	2	0	0	0	20	
Class Path	9	93	39	1	9	9	1	141	161

NON Class	Att	Str	var	Obj	Function	Cohesion	Complexity	TOT Complexity
C procedural		108	15	3	30	2	154	154

FOR JAVA:
MCM= C.I + C.D + C procedural
MCM= 0 + (20+141) + 154 = 161 + 154 = 315**.**

The outcome of the above complexity analysis shows that the cognitive complexity of the source code for dijkstra algorithm is 315 when java is used for implementation. It also shows that Java has the largest number of attributes and string when compare with other object oriented languages used here.

Table 6: Result Of M.C.C.M On Object Oriented Programming Languages

Programming Language	M.C.C.M results
C ++	138
C #	268
JAVA	315

This table shows that out of all the three Object Oriented Programming Languages, C++ has the least complexity and that C# has almost double the complexity of C++ while Java is almost three times the complexity of C^{++} . This is as a result of the fact that C# and Java source code in the implementation involved the use of a distinct class, this is evidence from the calculation.

b) Analysis of Cognitive Complexity of Dijkstra Algorithm on Procedural Programming Languages is as follows:

Table 7: Result of M.C.C.M on Procedural Programming Languages is as follows

Programming Language	M.C.C.M results
FORTRAN	66
PASCAL	67
C	77

Table 5 shows the summary of the computational analysis of **M.C.C.M** on Procedural Programming Language, this is to say that it depict what Table 4 has shown interm of Object Oriented Language for procedural programming language.

Table 8: Result of P.C.C.M for Procedural Programming Languages

Programming Language	PCCM
FORTRAN	784
PASCAL	815
C	1130

P.C.C.M result has shown in table 6 above, when compare with table 5 where M.C.C.M is used, shows that more values are attached to each procedural programming languages than has depicted by M.C.C.M in table 5, this imply that PCCM gives more details about some other feature of the procedural languages than has depicted in table 5. Although the result of the two metrics still consider Fortran to be the least in the complexity analysis.

Table 9: Comparism of the source code of Fortran 77 and C++ Language Cognitive Complexity

Programming Language	M.C.M result
Fortran	66
C++	138

After the use of M.C.C.M to analyse the cognitive complexity of Object Oriented Programming Languages and P.C.C.M for Procedural Programming Languages. Fortran 77 which has the least complexity in procedural languages was compared with C++ that also has the least complexity among Object Oriented Programming Languages. Using M.C.C.M for both Fortran 77 and C++ , Table 7 depict the result of their comparative analysis.

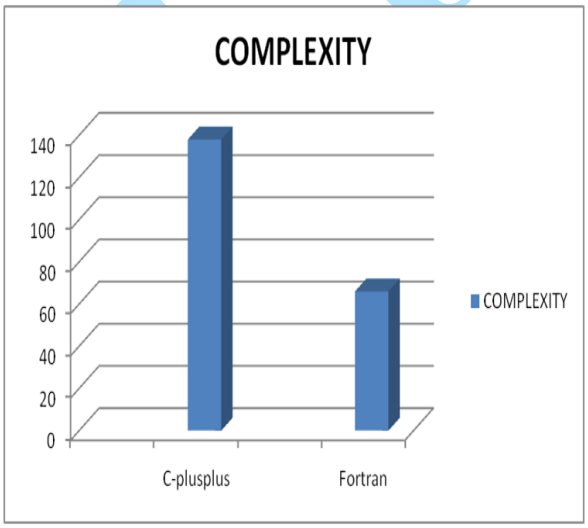

Figure 1: Graph Showing Software Cognitive Complexity between C++ and Fortran

At a glance, table 7 and figure 1 distinctly depict that Fortran source code even when compare with object oriented languages is the one that has the least Cognitive Complexity.

6. CONCLUSION

This research work evaluated the Cognitive Complexity of Dijkstra Algorithm using software complexity measure on six different programming languages implementation and make a conclusion from the results that the choice of programming language affects the cognitive complexity of Dijkstra Algorithm because Fortran language according to result is the best for implementation of Dijkstra algorithm to have the least cognitive complexity.
Generally, Fortran 77 and Pascal though they are old languages and not commonly used in large companies today but by this research work they are highly efficient in implementing Dijkstra Algorithm. This research has also revealed the relevance of earlier implementation languages in the recent software development since some earlier algorithms are still in use today.

REFERENCES

[AN00] **Akkanen J., Nurminen J. K.** - *Case-study of the evolution of routing algorithms in a network planning tool*, J. Syst. Software. 2000, 58, 181-198.

[BDZ10] **Banker R. D., Datar S. M., Zweig D.** - *Software Complexity and Maintainability*, CiteSeer Scientific Literature Digital Library and Search Engine. 2010.

[Caf10] **Cafer F.** - *An Estimating Complexity of software codes*. M.Sc. Thesis, Atilim University, 2010.

[Chh11] **Chhabra J. K.** - *Code Cognitive Complexity*. Proceedings of the World Congress of Engineering 2011 Vol II, WCH 2011, London, U.K., 2011.

[CAS03] **Chhabra J. K., Aggarwal K. K., Singh Y.** - *Code and data spatial complexity. Two important software understandability*, Measures, information and software Technology, 2003,Vol 45, no 8, pp. 539-546.

[FM10] **Francalanci C., Merlo F.** - *The Impact of Complexity on Software Design Quality and Cost: An Exploratory*

Empirical Analysis of Open Source Application (last accessed 16.30, 2010). Available at: http://is2.lse.ac.uk/asp/aspecis/20080122.p, df

[L+94] **Lee A. T., Gunn T., Pharm T., Ricaldi R.** - *Software Analysis Handbook: Software Complexity and Software Reliability Estimation and Prediction*. 1994.

[MA10] **Misra S., Akman I.** - *Unified Complexity Metric: A measure of Complexity*, Proceeding of National Academy of Sciences Section A. 2010.

[VJ07] **Vivanco R., Jin D.** - *Improving Predictive Models of Cognitive Complexity Using An Evolutionary Computational Approach: a case study*. National Research Council Canada. 2007.