Annals of Science and Technology - A, Vol 7 (1): xx-xx, 2022
Copyright: An Official *Journal* of the Nigerian Young Academy
ISSN: 2 544 6320

sciendo

ARTICLE

An Evaluation of a Language Processor for an African Native Language-based Programming Language

Ezekiel K. Olatunji[1*], Stephen O. Olabiyisi[2], John B. Oladosu[2], Odetunji A. Odejobi[3]

[1]*Department of Computer Science, Bowen University, Iwo, Osun State, Nigeria*
[2]*Department of Computer Science and Engineering, Ladoke Akintola University of Technology, Ogbomoso, Nigeria*
[3]*Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria*

Received 27th November, 2021, Accepted 3rd April, 2022
DOI: 10.2478/ast-2022-0001

**Corresponding author*
Ezekiel K. Olatunji E-mail: aekolatunji@gmail.com

Tel: +234(0)8038580770

Abstract

The design and prototype implementation of a subset of an African indigenous language-based programming language has been carried out and reported. In this study, an evaluation of the processor developed for the native language-based programming language was carried out in order to assess its level of conformance to the characteristics required of a good software product as set by the international organization for standardization (ISO). The developed language processor was evaluated using some metrics for evaluating the quality of software systems including structural and time complexity. A usability test was also conducted to assess users' perception of the system concerning its relevance and ease of use. The result of the system evaluation indicated that the system contains 1558 lines of code, its cyclomatic complexity is 14 and its asymptotic time complexity is of order big oh O (n^3), where n is the size of the input to the system. Over 86% of the participants in the usability test attested to the system's relevance while the usability rating was 86%. The developed system can be inferred to be of good quality as the results of its evaluation are positively on the high side for satisfying most of the ISO criteria for adjudging a software product as being of good quality. Furthermore, the high usability rating for the system indicates that the programming language whose compiler was evaluated satisfies most of the criteria set by the Department of Defense (DOD) for assessing the 'goodness' or otherwise of a programming language.

Keywords: Language processor; Lexical item; Native language-based programming language; Software evaluation; Software product metrics.

1.0 Introduction

A subset of a computer programming language (PL) based on the lexicons of an African indigenous language, specifically the Yoruba language, has been developed. Its design, as well, as its prototype implementation has been reported (Olatunji, et al., 2018; Olatunji et al., 2019a; Olatunji et al., 2021). More than 30 million people in Nigeria, Africa have Yoruba as their first language, while the language is spoken by more than one hundred million people throughout the world (Eludiora, et al., 2015).

An important rationale for developing the PL is to show that lexical items of PLs can also be adopted from the words of indigenous African languages like the words of English and Asian languages. According to Olatunji et al. (2018), the development of PLs based on the lexicons of African indigenous languages will improve computer-based problem-solving processes by indigenous teachers and learners. This has also been shown by other research studies (Silva et al., 2020; Olatunji et al., 2019b; UNESCO 2007; Pflepsen 2011).

The developed programming language is a structured and compiled language with syntax similar to that of traditional BASIC. The processor developed for the PL is a compiler and was designed to consist of five (5) components: scanner, information table, parser, semantic analyzer, and code generation. Besides these components, three (3) other utility programs were developed in the system to enable a user/programmer to create, compile and run programs in the programming language. These utility programs are YorCompilerMain, YorC-KodeGen, and Yoreditor. In this study, the developed processor for the native language-based programming language (NLPL) was evaluated to assess the level of compliance with the characteristics required of a good software product as set by the International Organization for Standardization (Rick-Rainer 2013).

1.1 Concept of software evaluation

Software evaluation, also sometimes called software metrics, has to do with the assessment of a software product and the process by which it is being developed. Software metrics provide quantitative methods for assessing the quality of software (Debbama et al., 2013). The Evaluation of software helps in understanding, controlling, and improving the software development processes and the resulting software product (Singh et al., 2008). According to Jah (2008), the knowledge and information obtained from software evaluation can be used to manage and control the development process which will eventually lead to improvement of the resulting software product. Software evaluation is broadly divided into two categories: software process evaluation and software product evaluation.

1.1.1 Software process evaluation

Software process refers to a set of activities that are partly ordered and carried out to manage, develop and maintain software systems. It centres on the development process rather than the product's output (Acuna and Juristo, 2005). Software process evaluation involves analyzing the activities that are undertaken in an organization to produce a software system. The utmost aim of process evaluation is to improve the quality of software products (Acuna and Juristo 2005).

Software process metrics describe the effectiveness and quality of the process that produced the software product. These include effort, time, and the number of times defects were found during testing (Singh et al., 2008). Software process evaluation is not further described beyond this point in this paper as it is not the focus of this research.

1.1.2 Software product evaluation

Software product evaluation, the focus of this paper, can be described as the assessment of the quality characteristics of a software product (Trendowicz and Punter 2008). Software product evaluation addresses and assesses the quality of a software product. The international organization for standardization (Rick-Rainer 2013) describes six quality characteristics of a software product. These are functionality, reliability, usability, efficiency, maintainability, and portability (Jah 2008). The degree to which a software product complies with these quality characteristics will be the degree of its quality rating by customers.

This paper reports on the outcome of the evaluation of the processor developed for the PL using standard metrics for evaluating the quality of software systems. The paper is organized into sections. Section two is on the review of the literature and related work. The third section hints at the methodology employed for the research. Discussion on the results obtained is provided in section four, while the last section is on the findings of the research.

2.0 Review of literature and related work

Software evaluation is an important aspect of software engineering (Awan et al., 2015) whose ultimate goal is to find methods used for developing high-quality software products at a reasonable cost (Sacha 2005). Thus the quality of a software product is a key factor and plays a crucial role in business success (Alnanseri 2020). Software quality, as cited by Madadipouya (2018) is defined by Sacha (2005) as consisting of two things: conformance to specifications and meeting customer needs. Many metrics exist for measuring the quality of a software system.

2.1 Review of related work

Some scholars have worked on the evaluation of software products and software processes. Molner et al (2020) carried out a comprehensive evaluation of software metrics that are widely associated with software product quality. A study in software quality assurance in the case of a system called Therac-25 was carried out by Madadipouya (2018). Therac-25 is a linear medical accelerator that was used to treat cancer patients of different types. The paper pointed out that the lack of proper deployment of software quality assurance was the cause of many incidents in the system. Debbama et al (2013) carried out a review and analysis of software complexity metrics in structured testing. The work presented an analysis by which developers and testers can minimize software development costs as well as improve testing efficacy and quality. A brief overview of software quality, software metrics, and software metrics methods that could be used to predict and measure

specific quality characteristics of software was provided by Jah (2008). In his work, a sample program developed in Java was evaluated using three software product metrics: size metric, complexity, and defect metrics. Sacha (2005) described a method that was used to evaluate the expected as well as the actual quality of a huge software that was developed in the year 2003 – 2004 to support the Common Agriculture Policy of the European Union in Poland. Recommendations on what to do to enhance the quality of the product were also provided.

2.2 *Review of software system evaluation metrics*

Many metrics have been used in evaluating a software system. These include but are not limited to product size, cyclomatic complexity, time complexity, usability, mean time between failure (MTBF), mean time to recover / repair (MTTR) and application crash rate (Olatunji 2019). Only the ones employed in this research are briefly reviewed here.

2.2.1 *Product size*

The product size metric is one of the most widely used metrics for evaluating a software product (Singhal et al., 2014) while a count of source lines of code (SLOC) has been the most commonly used size metric. The SLOC is a software metric used to measure the size of a computer program by counting the number of lines in the text of a program's source code. It is sometimes expressed as kilo lines per code.

SLOC is typically used to predict the amount of programming effort that would be required to develop a program as well as to estimate the programming productivity and /or maintainability once the program is produced. In general, SLOC is calculated as the total lines of the source code excluding the blank and comment lines

2.2.2 *Cyclomatic complexity*

Another commonly used metric to evaluate a software product is the cyclomatic complexity which measures the structural complexity of the software. The metric measures independent paths through a source code (Stein et al., 2005). The metric was developed by Thomas McCabe in 1976 (Fleck 2007). The metric can informally be described as the number of decision points plus one (Fleck 2007).

Formally, the metric is based on a control flow representation of the program. Control flow depicts a program as a graph that consists of nodes and edges. In a graph, nodes represent processing tasks, while edges represent control flow between the nodes. Mathematically, the complexity of a program can be defined (Olabiyisi 2005) as:

$$V(G) = E - N + 2$$

where E is the number of edges and N is the number of nodes. Alternatively, the cyclomatic complexity of a program can be defined as:

$$V(G) = P + 1$$

where P is the number of predicate nodes (node that contains conditions). McCabe's recommendation is that a program being developed should be split into smaller modules if its cyclomatic complexity is more than 10.

At a later time, however, the National Institute of Standards and Technology (NIST) recommended that, in some circumstances, a cyclomatic complexity of up to 15 is acceptable (Jones and Hogenson 2021).

2.2.3 *Time complexity*

The time and memory space used by an algorithm/program are the two measures of the efficiency of the program/algorithm. More commonly the time required for executing an algorithm/ program is used to determine the efficiency or time complexity of the algorithm/program. In the analysis of an algorithm, the worst-case running time is usually estimated as the function of the input size. The time complexity of an algorithm M is the function f(x) which gives the running time of the algorithm in terms of the size x of the input data (Lipschutz and Lipson 2007). In general, it is the function that gives the worst-case running time of the algorithm in terms of the size of the input. The running time for an algorithm/program can be estimated using the method of asymptotic analysis (Olabiyisi 2005) and expressed with the big O-notation without having to implement the algorithm (Cormen, et al., 2009).

Asymptotic analysis is a technique used to estimate the running time complexity of an algorithm as its input size tends to infinity. An algorithm's growth rate or running time is usually approximated to a function that can be linear, quadratic, logarithmic, or even exponential. In general, an algorithm with the big-O notation of order O(log n) is faster than the one with O(n); likewise, the one with O(n) is more efficient than the one with $O(n^2)$, where n is the input size (Cormen, et al., 2009).

2.2.4 *Usability Test Rating*

Usability testing refers to an assessment of a product or service by testing it with prospective representative users. The test describes the level of ease with which a product like the one being reported can be used by the audience for which the system was developed. According to Okhovati et al (2016), usability testing is the second most used evaluation research method and the method that has the greatest impact in making products better. In usability testing, participants are requested to make use of the developed system and provide an assessment of their perception of the system. The participants represent "real users", doing the "real thing"; after which they can provide an assessment of their perception of the system.

3.0 Methodology

The evaluation of the developed Yoruba-based PL was carried out by using quantitative and qualitative metrics., The system was evaluated using software product metrics in the quantitative approach;, specifically size and complexity. In the qualitative approach, the system was evaluated by conducting usability testing to assess users' perception of the system concerning its relevance and ease of use.

3.1 *Product size metric*

The product size metric used for evaluating the developed Yoruba-based compiler is the count of the source lines of code (SLOC) described in Section 2.2.1.

The size of the system developed in this research was obtained by summing up the SLOC of all the functions and modules in the system. The SLOC for each module within each of the six main programs has been calculated by subtracting the blank and comment lines from the total source lines.

3.2 Cyclomatic complexity metric

Another quantitative metric used for evaluating the developed system is the cyclomatic complexity described in Section 2.2.2. While it is possible to compute the cyclomatic complexity of a program by using such tools as Cyvis (Jah, 2008), in this research, cyclomatic complexity was computed manually.

3.3 Time complexity metric

The developed system was also evaluated using the time complexity described in section 2.2.3. The method of asymptotic analysis was used in estimating the time complexity of the developed Yoruba-based programming language

3.4 Usability testing

The developed Yoruba-based compiler system was also evaluated by conducting a usability test. The goal of conducting this test is to evaluate the developed programming language (PL) and its compiler in terms of usability as well as the relevance and usefulness of the system relative to similar English-based PLs, such as BASIC, FORTRAN, and others. The selected audience for the test was requested to write simple but meaningful programs in the developed PL and then make a comparison with their experience in some similar English-based PL with which they are satisfactorily familiar. The comparison was made in terms of the following criteria:

- Ease of understanding the PL's Syntax and semantics
- Ease of use (in programming)
- User-friendliness of the user interfaces of the IDE
- Efficacy / Effectiveness/ usableness
- Relevance / Usefulness

Most of these criteria are also part of the US, Department of Defense (DOD)'s criteria for assessing and adjudging a programming language to be of good quality as far back as 1978 (Chen et al., 2005; Sebesta 2012).

Participants in the usability test comprised seven (7) computer science students in a private University: one female and six males; one 300 Level and six 400 Level students. Usability testing experts like Jakob Nielson, as quoted by McCraken (2016), recommended that 5-8 participants are sufficient for the test as they will provide 80% -95% of the usability issues of the system. Six (6) out of seven (7) of the participants were in the last semester of their graduating year and were also good in programming in some of the English-based PLs like Java and C++.

Furthermore, all the participants were fluent in spoken Yoruba and the user operating manual provided during the test was simple enough for all of them to comprehend the syntax and semantics of the Yoruba-based PL These participants were made to write, compile and run two simple programs in the programming language and then to assess the programming language and its compiler by completing a usability assessment questionnaire. In the questionnaire, participants were required to assess the Yoruba-based programming language based on the criteria specified above using a Likert scale of 1 to 5, where 1 is the least and 5 is the highest.

3.5 Sufficiency of the employed metrics

It is to be remarked that the evaluation metrics employed in this work as explained in section two are adequate in assessing the programming language and its processor. First, the developed system has been tested and reported to satisfy its functional requirements (Olatunji et al., 2021). This is also confirmed by the outcome of the usability test as will be seen in section four. Functionality is a major requirement of a good software product according to ISO standards (Rick-Rainer 2013). Secondly, cyclomatic complexity deals with the structural complexity of the system. It is a measure of the testability as well as the ease of maintainability of a system. By ISO standards, ease of maintainability ranks high among the qualities and requirements of a good software. An acceptable level of maintainability has been benchmarked by NIST (Jones and Hogenson 2021).

Usability testing measures the relevance, usefulness as well as ease of use of a system. This factor is also one of the quality characteristics of a software product as per ISO standards. Furthermore, as earlier alluded to, usability testing is the second most used evaluation research method. In addition, most of the qualities of the PL required to be assessed by participants in the usability test are part of the criteria set out in 1978 by the US, Department of Defense (DOD), Washington DC (Chen et al., 2005; Sebesta 2012) to be met by a good PL. The DODs criteria have since been a reference for PL evaluation.

4.0 Results and Discussion

The results obtained from the evaluation of the system are presented in this section.

4.1 Result of SLOC computation

The summary of the SLOC computed for the system is shown in Table 1.
The SLOC metrics represent the size of the system developed as well as the sizes of each of the main program components of the system. As can be observed in Table 1, the total actual source lines of code (SLOC) is 1558. This indicates the non-triviality of the programming effort involved in developing the system; more so that only a few constructs were used for prototype implementation. Apart from the main module of the scanner, whose cyclomatic complexity is 1 all other modules have their SLOC to be less than 60, which is the commonly recommended size of a good program module (Jones and Hogenson 2021).

4.2 Result of evaluating cyclomatic complexity of the system

The calculated results of cyclomatic complexity for the system are shown in Table 2. The module with the highest cyclomatic complexity in a program determines the cyclomatic complexity of the program.

From Table 2, the cyclomatic complexity of the system is 14, which is the cyclomatic complexity of the Semantic Checker component of the system because the semantic checker has the highest cyclomatic complexity.

According to the recommendation of MacCabe (Olabiyisi 2005), the system can be said to be moderately well structured, have moderate complexity and medium testability with moderate cost and effort. Going by McCabe's earlier recommendation that case statements should be exempted from the complexity of a program module (Jones and Hogenson 2021), the cyclomatic complexity of the semantic checker and those of other main components of the developed system is far less than 10, the acceptable degree of structural complexity. Thus by these recommendations, the developed system can be said to be well-written and structured, have high testability and requires low maintainability cost and effort. Furthermore, according to the later recommendation of NIST (Jones and Hogenson 2021) that cyclomatic complexity of up to 15 is acceptable, the developed system is very well structured and written, has high testability and requires less effort and cost in maintaining it.

4.3 Result of evaluating the time complexity of the system

The asymptotically estimated worst-case running time complexities for the program components of the system are as shown in Table 3. The module with the highest asymptotic value determines the time complexity of a program component, while the program that has the highest asymptotic value determines the time complexity of the system. From Table 3, the worst-case running time complexity of the developed system is of order $O(N^3)$, where N is the size of the input to the system. This is the worst-case running time of both the scanner and the parser. The big O-notation is used to describe the asymptotic upper bound of the size of the input to the system. This time complexity agrees with Earley Jay's algorithm as cited by Tomita (2013).

4.4 Result of usability testing

In the questionnaire, participating users were made to provide a candid assessment of the programming language and its compiler by rating the language on the six criteria shown in the second column of Table 4. on a Likert scale of 1 to 5 where 5 is the highest and 1 the least. Furthermore, 5, 4, 3, 2, and 1 respectively stand for very good, good, average, poor and very poor. It is to be remarked that assessment criteria 1, 2 and 3 measure the system's ease of use, criteria 5 and 6 measure the system's degree of relevance, while criterion 4 measures the system's performance in terms of speed of programming. The result of the responses of the users is contained in Table 4.

From the result of the analysis in Table 4., and the users' perception of the system, the programming language's syntax and semantics are very easy to understand (because 100% of the participants attested to this); easy to use for programming and its user interface is moderately user-friendly (as indicated by 86% of the participants). Furthermore, the programming language and its compiler are also usable and effective (as attested to by 86% of the participants) while the programming language is very much desired as indicated by 100% of the participants. These percentages are the sum of 'very good' and 'good' responses.

Table 1: Summary of SLOC metric for the Yoruba Compiler

S/No	Program Name	Number of Modules	Total SLOC in QB64 Editor	Total Comment and blank Lines	Total Actual SLOC
1	Yoruba Compiler IDE	6	147	37	110
2.	Scanner	17	478	147	331
3.	Parser	13	573	131	442
4.	Info Table Builder	5	291	78	213
5.	Semantic checker	8	418	109	309
6.	Code Generator	9	197	61	136
7.	YorubaC_KodeGen	1	17	5	12
8.	YorubaEditor	1	11	6	5
	Total	60	2132	574	1558

Table 2: Summary of Cyclomatic complexity for the Yoruba Compiler

S/No	Program Name	Highest Cyclomatic Complexity
1	Yoruba Compiler IDE	5
2.	Scanner	9
3.	Parser	12
4.	Info Table Builder	10
5.	Semantic checker	14
6.	Code Generator	9
7.	YorubaC_KodeGen	1
8.	YorubaEditor	1

Table 3: Summary of Time Complexity for the Yoruba Compiler

S/No	Program Name	Highest Asymptotic value in terms of big O-notation
1.	Yoruba Compiler IDE	N
2.	Scanner	N^3
3.	Parser	N^3
4.	Info Table Builder	N^2
5.	Semantic checker	N^2
6.	Code Generator	N^2
7.	YorubaC_KodeGen	1
8.	YorubaEditor	1

Table 4: Result of Users' Evaluation of the Developed System

S/No	Criteria / Likert Items	5 Very Good	4 Good	3 Average	2 Poor	1 Very Poor	Response Mean	Response Mode
1.	Ease of understanding The PL's Syntax and semantics	5 71%	2 29%	-	-	-	4.71	5
2.	Ease of using the system for programming	4 57%	2 29%	1 14%	-	-	4.43	5
3.	User-friendliness of The user interfaces	3 43%	3 43%	1 14%	-	-	4.28	5
4.	Performance (in terms of Speed of programming)	1 14%	4 57%	2 29%	-	-	3.86	4
5.	Efficacy / Effectiveness/ usability	4 57%	2 29%	1 14%	-	-	4.43	5
6.	Usefulness / Relevance of the PL	6 86%	1 14%	-	-	-	4.86	5

5.0 Conclusion

Based on the metrics used for evaluating the developed system and the results obtained as discussed in section IV, it can be inferred that the developed system is good. This is because the system sufficiently satisfies most of the criteria for adjudging a software product as being good, especially the criteria set by ISO standards. For example, cyclomatic complexity, which among other things, measures the ease of maintaining a system is 14. This value is within the acceptable level of maintainability as benchmarked by NIST. Furthermore, the high usability rating expressed by the participants of the usability test (86% both for ease of programming and user-friendliness of the system's user interface) confirms that the PL is well-defined (syntactically and semantically), expressive, satisfactorily orthogonal and very pedagogical. These are some of the criteria set by the Department of Defense (DOD), Washington DC for assessing whether a PL is good or not (Chen et al., 2005). For future work, the developed language processor, like any other software product, could be evaluated on other software product metrics such as mean-time between failure (MTBF), meantime time to recover (MTTR) and application crash rate (Olatunji 2019). Other metrics include Halstead software science, maintainability index (Molner et al., 2017) and others.

Conflicts of Interest

All authors have declared that there are no conflicts of interest.

Individual Authors Contribution

Conception; [E.K.O, J.B.O, S.O.O]
Design: [E.K.O, J.B.O, O.A.O]
Execution: [E.K.O]
Interpretation: [E.K.O, J.B.O, O.A.O, S.O.O]
Writing the Paper: [E.K.O]

References

Acuna, S.T. and Juristo, N. (2005). Software Process Modelling, Boston, Springer, 2005, pp 111-139 DOI: https://doi.org/10.1007 /b104986

Alnanseri, M. (2020). User Authentication in Public Cloud Computing Through Adoption of Electronic Personal Synthesis Behaviour. PhD Thesis, , http://doi.org/10.13140 /RG.2.235475 .71202

Awan, S., Malik F., and Javed, A. (2015). An Efficient and Objective Generalized Comparison Technique for Software Quality Models. In: International Journal of Modern Education and Computer Science, 7(2); 57-64. https://doi.org/10.5815/ijmecs.2015.12.08

Chen, Y; Dios, R., Mili, A. and Wu, L. (2005). An Empirical Study of Programming Language Trends, IEEE SOFTWARE, IEEE Computer Society, 22(3); 72-79. DOI: https://doi.org/10.1109/MS.2005.55 Available online at www.computer.org/software. Retrieved on 17-04-2014

Cormen, T.H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2009). Introduction to Algorithms, 3ed, Massachusetts, The MIT Press, pp 30–35

Debbama, M.K., Debbama, S., Debbama, N., Chakma, K., and Jamatia, A. (2013). A Review and Analysis of Software Complexity Metrics in Structural Testing. International Journal of Computer and Communication Engineering, 2(2);129-133. https://doi.org/10/7763/IJCCE.2013.v2.154

Eludiora, S.I., Agbeyangi, A.O. and Fatusin, O. I. (2015). Development of English to Yoruba Machine Translation System for Yoruba verbs' Tone Changing. International Journal of Computer Applications, 129(10);12-17.

Fleck, D. (2007). Cyclomatic Complexity. Available online at http://cs.gmu.edu/~dfleck/ Retrieved on 02-02-2018

Jah, M. (2008). Software Metrics – Usability and Evaluation of Software Quality. M.Sc.Thesis, Department of Technology, Mathematics and Computer Science, University West, SWEDEN, pp 95-101.Retrieved on 02-02-2018 from https://pdfs.semantic scholar.org/

Jones, M. and Hogenson, G. (202)1. Code Metrics – Cyclomatic Complexity. Retrieved on 14-03-2022 from https://docs.microsoft .com

Lipschutz, S and Lipson, M. (2007). Discreet Mathematics, New York, Mcgraw-Hill Companies, Inc, ISBN 978-0-07-161586-0[LLVM], pp 605 http://llvm.org/ Retrieved 11-02-2015

Madadipouya, K. (2018). Importance of Software Quality Assurance to Prevent and Reduce Software Failures in Medical Devices: Therac-25 Case Study, pp 1-18. https://doi.org/10.6084/m9.figshare.3362281/5

McCraken, C. (2016). How to conduct Usability Testing from Start to Finish. Retrieved online from http://uxmastery.com/ on 24-05 - 2017.

Molnar, A J., Neamtu A., Motogna S. (2020). Evaluation of Software Product Quality Metrics. In: Damiani E., Spanoudakis G., and Macaszek L. (eds) Evaluation of Novel Approaches to Software Engineering. ENASE 2019. Communications in Computer and Information Science, Springer, Cham, Vol 1172; 163-187. https://doi.org/10.1007/978-3-030-40223-5-8

Molnar, A.; and Motongna, S. (2017). Discovering Maintainability Changes in Large Software Systems. In: Proceedings of the 27th International Workshop on Software Measurements and 12thth International Conference on Software and Process and Product measurement, pp 88-93. IWSM Mepsure '17, ACM, New York, NY, USA (2017). https://doi.org/10-1145/31434344.3143447

Okhovati, M., Karami, F. and Khajouei, R. (2016). Exploring the Usability of the Central Library Websites of medical Sciences. Journal of Librarianship and Information Science, 49(3); 23-26. https://doi.org/10.1177/096100050932

Olabiyisi, S. O. (2005). Universal Machine for Complexity Measurement of Computer Programs. Unpublished PhD Thesis, Ogbomoso – Nigeria, Ladoke Akintola University of Technology

Olatunji, E. K., Oladosu, J. B., Odejobi, O. A. and Olabiyisi, S. O. (2021). Design and Implementation of an African Native Language-based Programming Language. International Journal of Advances in Applied Sciences (IJAAS), 10(2); 171-177. https: //doi.org/10 .11591/ijaas.v2.i2.pp171-177

Olatunji, E. K. (2019). Development of a Programming Language with Yoruba Lexicons. Unpublished PhD Thesis, Ogbomoso-Nigeria, Ladoke Akintola University of Technology.

Olatunji, E. K., Oladosu, J. B., Odejobi, O. A. and Olabiyisi, S. O. (2019a). Design of an African Native Language-based Programming Language. University of Ibadan Journal of Science and Logics in ICT Research (UIJSLICTR), 3(1); 72-78

Olatunji, E. K.,Oladosu, J. B., Odejobi, O. A. and Olabiyisi, S. O. (2019b). A Needs Assessment for Indigenous Language-based Programming Language, Annals of Science and Technology (AST) - E, 4 (2); 1-5 DOI: https://doi.org/10.2478/ast-2019-0007

Olatunji, E.K., Oladosu, J.B., Odejobi, O. A and Olabiyisi, S.O. (2018). Towards Development of an Indigenous African Language-based Programming Language. FUOYE, Journal of Engineering and Technology (FUOYEJET), 3(2); 61-64. https://doi.org/10.46792/

Pflepsen, A. (2011). Improving Learning Outcomes through Mother Tongue-Based Education, MTB-MLE Network. Available online at https://www.eddataglobal.org Last Accessed on 30-07-2014, stored online as eddata_ii_mother_tongue _instruction.pdf 1(2); 148-159.

Rick-Rainer, L. (2013). Software Characteristics in ISO 9126. Retrieved on 14-03-2022 from https://rickrainerludwig .wordpress.com/2013/

Sacha, K. (2005). Evaluation of Software Quality. In Software Engineering: Evolution and Emerging Technologies, pp 381 – 388, Amsterdam, the Netherlands

Sebesta, R.W. (2012). Concepts of Programming Languages. New Jersey, Pearson Education Inc.

Silva, G., Santos, G., Canedo, E.D., Rissoli, V., Praccano, B. and Andrade, G. (2020) Impact of Calongo Language in an Introductory Computer Programming Course. In 2020 IEEE Frontiers of Education Conference, pp 1-9. DOI: https://doi.org/10.1109/FIF44824.2020.9274150

Singh Y., Kaur A. and Suri B. (2008). An Empirical Study of Product Metrics in Software Testing. In: Iskander M. (eds), Innovative

Techniques in Instruction Technology, E-learning, E-assessment, and Education. Springer, Dordrecht. https://doi.org/10.1007/978-1-4020-8739-4_12

Singhal, S., Suri, B. and Gaur, G. (2014). Overview of Software Engineering Metrics for Procedural Paradigm. In Proceedings of IEEE on IT in Business, Industry and Government (CSIBIG), Indore, pp 1-5. DOI: https://doi.org/10.1109/CSIBIG.2014.7056967. Retrieved on 15-03-2022 from https://www.researchgate net/publication/

Stein, C., Cox, G., and Etzkorn. (2005). Exploring the Relationship between Cohesion and Complexity. Journal of Computer Science, 1(2); 137-144. https://doi.org/10.3844/jcssp.2005.137.144

Tomita, M. (2013). Efficient Parsing for Natural language: A fast Algorithm for Practical Systems. (Vol 8),
Springer Sciences and Business Media, pp. 105

Trendowicz, A. and Punter, T. (2008). Quality Modelling for Software Product Lines. Retrieved On 14-03-2022 from https://www.reserachgate.net/publication/

UNESCO (2007). Mother Tongue-based Literacy Programmes: Case Studies of Good Practice in Asia. Bangkok:
UNESCO Bangkok, Thailand, ISBN 92-9223-113-8 Available online at www.unesdoc.unesco.org Retrieved on 30-07-2014