IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Ingenierie des Systemes d'Information
Vol. 24, No. 1, February, 2019, pp. 83-87
Journal homepage: http://iieta.org/Journals/isi

An Analysis of Maintainability Index Influencing Metrics and Their Behavior on Similar Open Source Gaming Application Developed in C, C++ and, JAVA

Gokul Yenduri*, Naralasetti Veeranjaneyulu

Department of IT, VFSTR Deemed to be University, Vadlamudi 522213, Guntur-AP, India

Corresponding Author Email:yenduri.gokul@gmail.com

https://doi.org/10.18280/isi.240112

Received: 5 December 2018
Accepted: 3 February 2019

Keywords:
maintainability metrics, software quality, SDLC, MI, code smell

ABSTRACT

Gaming is a major entertainment to the world. It plays an important role in reducing the stress of many people. Constructing a game with high Quality is an important aspect. The quality of gaming software depends on many factors such as reliability, usability, maintainability, and other factors. Maintainability is a predominant factor among them as it affects the cost of open source gaming projects. It is important to forecast the consequence of such a crucial factor ahead of releasing the games as they are nonprofitable to the developers. In this paper, we collected 25 open source gaming application developed in various programming languages with the help of visualization and statistical approach to examine the maintainability of gaming applications. OSS gaming has an acceptable level of maintainability with a vast behavioral difference between metrics.

1. INTRODUCTION

Maintainability is one of the important quality factors of software. Software Maintenance has been a major issue in the software development life cycle. Since the structure oriented and object-oriented programming are being used to develop software. A lot of research is carried out towards maintainability prediction of structure and object-oriented programming. Software maintainability can be calculated based on code-level and design level metrics. Gaming software maintenance has a huge effect on cost and endeavor. As a result, gaming software systems must be maintainable. On the other hand, Several empirical research studies are done to investigate the maintainability of software. This paper considers these following important aspects.

(1). How maintainable are similar open-source gaming applications developed in various programming languages?

(2). How the behavior of various code-level metrics will be in similar open source applications developed in various programming languages?

(3). What extent MI is a superior measure to predict maintainability of various programming paradigms?

2. SOFTWARE MAINTENANCE METHODS

2.1 Maintainability Index

Maintainability Index (MI) is a measurement blended with various other metrics referred to predict the maintainability of source code. MI uses various metrics in its formula to arrive at the result, metrics include programmed Lines (LOC), Cyclomatic Complexity (G) and Halstead volume (V) [1]. MI is calculated as shown below

$$MI = 171 - 5.2 * 1n(avgV^{H}) - 0.23 * avgV^{H}(c) - 16.2 * 1n(avgLOC)$$

where

*V=(ln(N1+N2)*log((n1+n2)))*

G =(e-n+2p)

Table 1. MI ranges and maintainability of any software

MI RANGE	MAINTAINABILITY
00-10	LOW
10-20	MEDIUM
20-100	HIGH

3. ITERATURE REVIEW

Researchers from past estimated maintenance using various models and methods.

McCall et al., projected a model for software quality and distinct sub-factors for quality and categorized those factors into three diverse parts product amendment, product function, and product conversion. ease, succinctness, and modularity as the software quality sub-factors [4].

Boehm et al., proposed a quality model for software and described testability, understandability, and elasticity as the software quality sub-factors [4].

Peercy et al., defined a model to signify structure of maintainability of software depending on modularity, descriptiveness, reliability, ease, expandability and instrumentation sub-factors [5].

Sneed and Mercy projected a model and described maintainability as a gauge of factors of software [6].

Matinlassi et al., “maintainability classification is not only very important but also the scientific aspects of maintainability are significant”. This model was anticipated and described the effect of quality attributes on system, construction and module scope of the software system [7].

Hayes et al., anticipated a model that measures adaptive software maintenance attempt in requisites of variation outline of code i.e. the quantity of new, deleted and updated lines [8].

Hayes and Zhao, developed a maintainability model, which differentiates software modules as ‘simple to maintain’ and ‘complex to maintain’. The evaluation shows that different maintainability models highlight on its sub-factors during the development of such models [8]. Mattson et al., developed a structure for software maintainability by making an allowance for two aspects like artifact and practice [9]. Kazuya et al., developed a model for error detection as well as liability correction process. This model also provided an outline to assess the software maintainability [10]. Maintainability model by Heitlager et al., is a Customized version of ISO 9126 where all features are customized and system level factors are mapped to properties on the level of cause code [11].

Singh et al., developed a model, it is used to review the software maintenance. The factors such as Readability of cause Code, annotations Ratio, documents quality, Understandability of Software and standard Cyclomatic Complexity were used as contribution variables in the planned model. [12].

Khan et al., developed a model for the pre-release maintenance process and to conduct post deliverance maintenance stage effectively [13].

4. PHASES OF IMPLEMENTATION

4.1 Investigation phase

In this phase, various gaming applications are downloaded from web which is developed in various programming paradigms, from them twenty-five gaming applications are selected which are similar and developed in C, C++, and java.

4.2 Preparation phase and data retrieval phase

In this, the selected gaming applications codes are given as an input to tools (CCCC and HM tools) and the output of tools are measurements; so-called metrics are obtained for all similar gaming applications developed in various paradigms.

4.3 Testing phase

In this phase, the metrics obtained are converted into a dataset. Data visualization is done all three paradigms and statistical analysis is applied to two programming paradigms. Based on this phase results are obtained which leads us to conclusions.

5. RESULTS

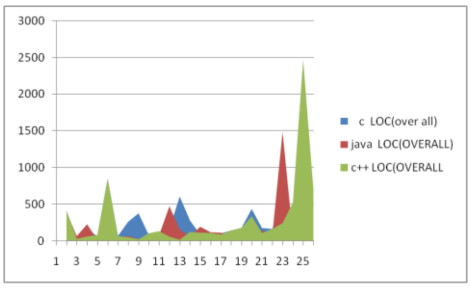

Figure 1. The behavior of lines of code (overall)

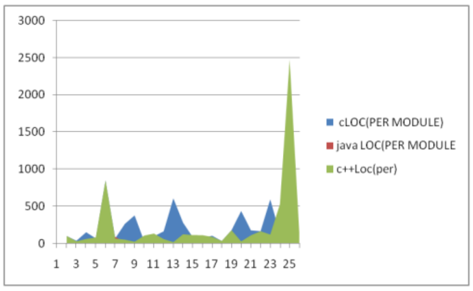

Figure 2. The behavior of lines of code (Per module)

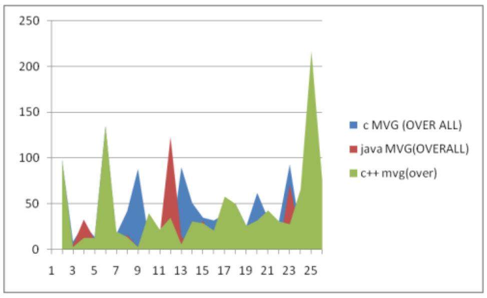

Figure 3. The behavior of Cyclomatic complexity(overall)

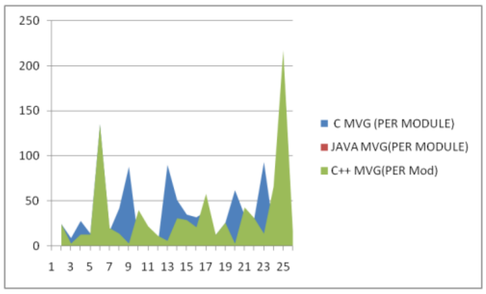

Figure 4. The behavior Cyclomatic complexity(Per module)

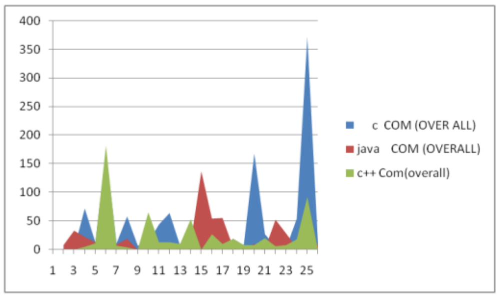

Figure 5. The behavior of comments (overall)

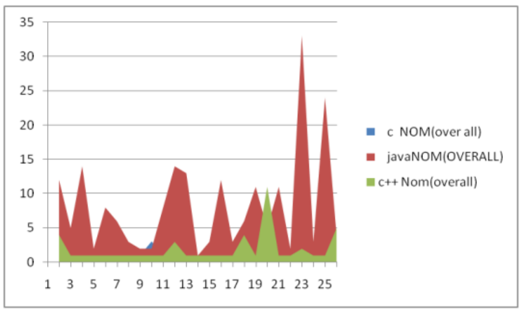

Figure 6. The behavior of the number of modules (overall)

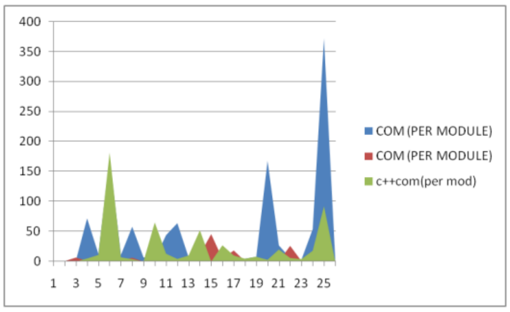

Figure 7. The behavior of comments (Per module)

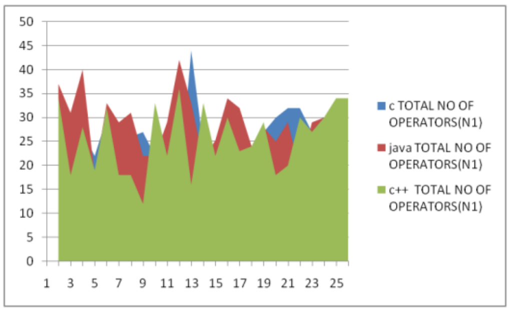

Figure 12. The behavior of total of distinct operators

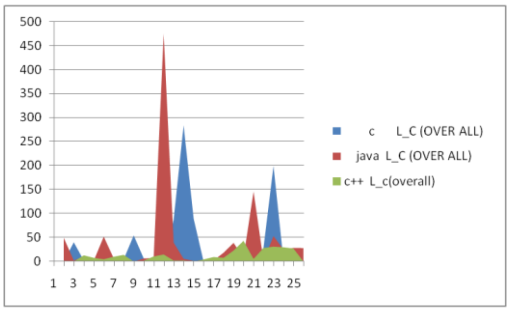

Figure 8. The behavior of comments per LOC (overall)

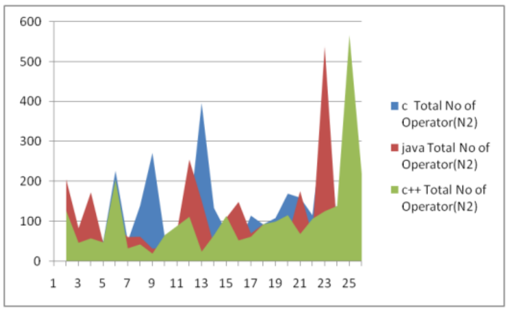

Figure 13. The behavior of total of Distinct operands

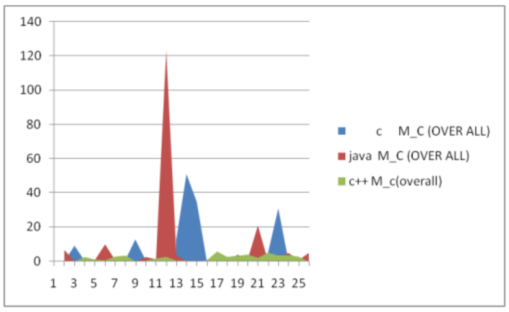

Figure 9. The behavior of comments per module(overall)

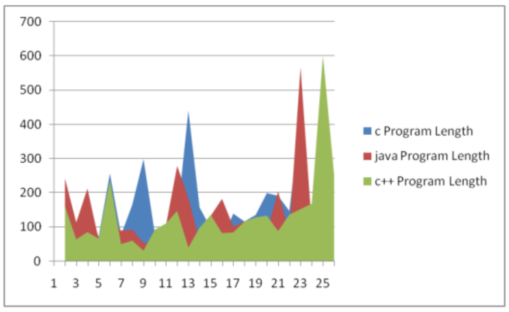

Figure 14. The behavior of program length

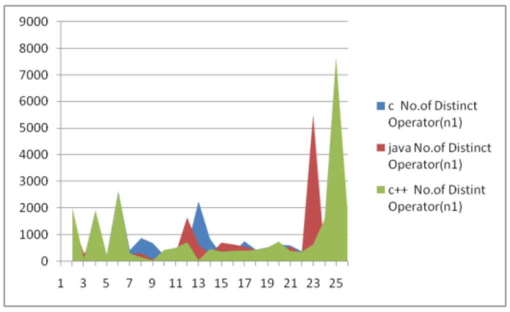

Figure 10. The behavior of distinct operators

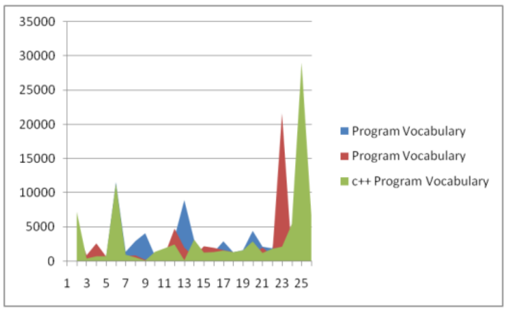

Figure 15. The behavior of program Vocabulary

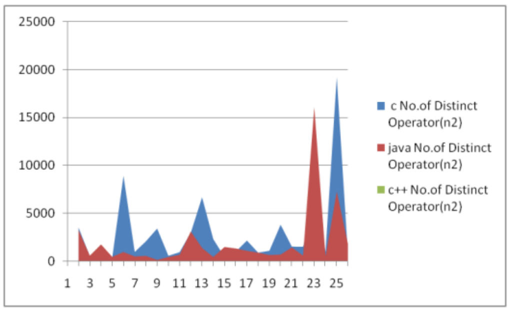

Figure 11. The behavior of distinct operands

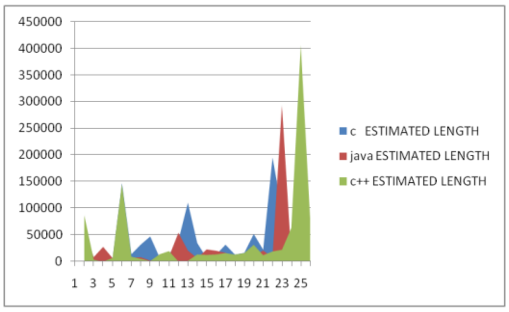

Figure 16. The behavior of estimated length

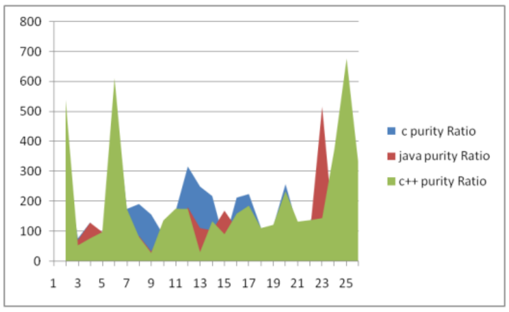

Figure 17. The behavior of purity ratio

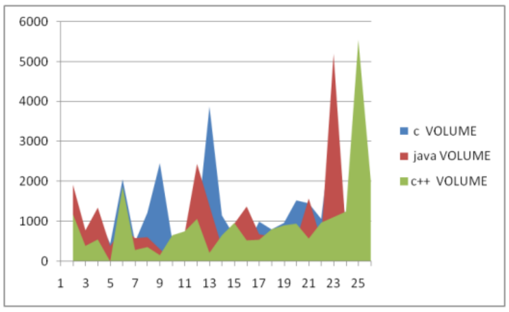

Figure 18. The behavior of Volume

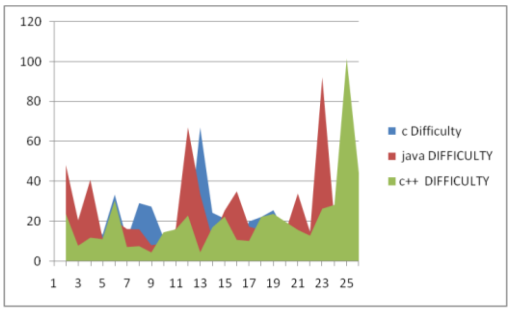

Figure 19. The behavior of program difficulty

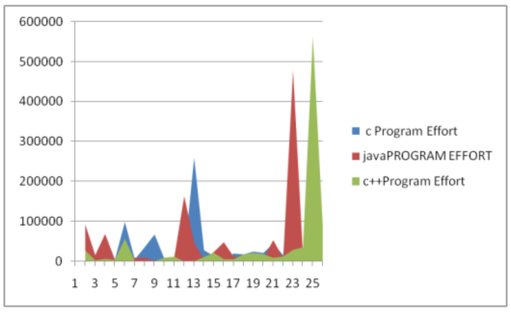

Figure 20. The behavior of program effort

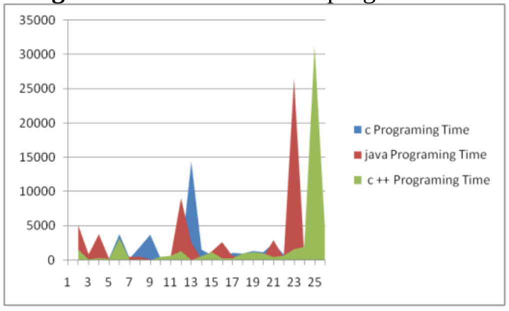

Figure 21. The behavior of programming time

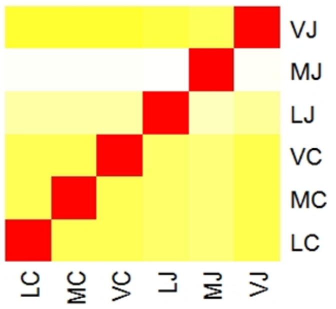

Figure 22. Heat map of pearsons correlated distance, behavior MI influencing metrics in similar gaming application developed in C and Java

6. CONCLUSION AND FUTURE WORK

Based on the above visualization results, There is a vast difference in the behavior of maintainability metrics as observed in the Figures (1-21)and statistical analysis performed related to MI metrics using pearsons correlation distance on lines of code(l), cyclomatic complexity(m) and Halsted volume (v) in C and Java with help of heat map as shown in Figure (22) leads to conclusions.metrics used to predict maintainability of gaming applications shows a vast difference between them there is no consistency between metrics and it creates confusion for predicting maintainability. Maintainability index looks promising but it uses the same metrics for both structural and object-oriented paradigm which is arguable. Based on visualization and basic statistical analysis it is hard to determine the exact behavior of different programming languages. A good statistical analysis on a huge data set can help in determining the behavior of maintainability metrics.

In the future, we aim to study the dependencies and interdependencies between the metrics and propose a new model or a metric to predict Maintainability of different programming paradigms.

REFERENCE

[1] Zhuo F, Lowther B, Oman P, Hagemeister J. (1993). Constructing and testing software maintainability assessment models. IEEE Computer Society 61-70. https://doi.org/10.1109/METRIC.1993.263800

[2] Coleman D, Ash D, Lowther B, Oman P. (1994). Using metrics to evaluate software system maintainability. Computer 27(8): 44-49. https://doi.org/10.1109/2.303623

[3] https://blogs.msdn.microsoft.com/zainnab/2011/05/26/code-metrics-maintainability-index

[4] Samadhiya D, Wang SH, Chen DJ. (2010). Quality models: Role and value in software engineering. 2nd International Conference on Software Technology and Engineering 25: 3-5. https://doi.org/10.1109/ICSTE.2010.5608852

[5] Peercy E. (1981). A software maintainability evaluation methodology. IEEE Transactions on Software Engineering 7(4): 343-351. https://doi.org/10.1109/TSE.1981.234534

[6] Sneed H, Mercy A. (1985). Automated software quality assurance. IEEE Trans. Software Eng 11Bi 9: 909-916.

https://doi.org/10.1109/TSE.1985.232548

[7] Matinlassi M, Niemelä E, Dobrica L. (2002). Quality-driven architecture design and quality analysis method: A revolutionary initiation approach. Tech. Report VTT Technical Research Centre of Finland.

[8] Hayes JH, Zhao L. (2005). Maintainability prediction: A regression analysis of measures of evolving systems. In 21st IEEE International Conference on Software Maintenance (ICSM'05) 21: 26-29. https://doi.org/10.1109/ICSM.2005.59

[9] Mattsson M, Grahn H, Frans F. (2006). Software architecture evaluation methods for performance, maintainability, testability, and portability. Second International Conference on the Quality of Software Architectures. https://doi.org/10.1007/11921998

[10] Kazuya S, Koichiro R, Tadashi D, Hiroyuki O. (2007). Quantifying software maintainability based on a fault-detection/correction model. 13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007), pp. 17-19. https://doi.org/10.1109/PRDC.2007.57

[11] Heitlager I, Kuipers T, Visser J. (2007). A practical model for measuring maintainability. 6thinternational conference on the quality of information and communications technology (QUATIC 2007). IEEE https://doi.org/ 10.1109/QUATIC.2007.8

[12] Singh Y, Bhatia PK, Sangwan OP. (2009). Predicting software maintenance using fuzzy model. ACM SIGSOFR SEN 34(4): 1-6. https://doi.org/10.1145/1543405.1543425

[13] Khan AS, Kajko-Mattsson M, Tyrberg T. (2009). Comparing the EM3 pre-delivery maintenance model with its industrial correspondence. (IMCSIT), 573-582. https://doi.org/10.1109/IMCSIT.2003.5352783