Code Readability: A Review of Metrics for Software Quality

Ankit Pahal*[1] , Rajender S. Chillar[2]

[1]*Student, Department of Computer Science & Application, M.D. University, Rohtak (Haryana)*
Address: 1796/22 Azad Nagar, Rohtak-124001, Haryana, India
[2]*Professor, Department of Computer Science & Application, M.D. University, Rohtak (Haryana)*

Abstract: *Various software metrics evaluate the complexity of software by using some physical software characteristics. Readability Metrics is exceptional amongst the present software complexity metrics for considering a non-physical software characteristics i.e. readability. Readability should be the key quality attributes for program source codes. The readability of the software is strongly associated to its maintainability, and is thus the crucial feature in whole quality of software. More the readable code, greater the chances of having easier to modify, less mistakes, more maintainable, easy to reuse, and more reliable. Readability is used to improve source codes for future preservation and extensibility. But code readability is not simply computable with a deterministic function. In this review paper, we will study various common readability metrics present in the literature such as Flesch-Kincaid metric, Gunning-Fog metric, SMOG index and Automated Readability Index (ARI) and how to calculate readability score metrics. Then we will relate the notion of code readability and examine its relation to software quality. Lastly, based on this review study, we will classify challenging issues for the future work of the code readability.*

Keywords: *software engineering, code readability, software quality, code maintainability.*

I. INTRODUCTION

Software metrics are a mathematical description representing the entities of a software programs to numeric metrics values [1]. Moreover, we take software metrics tool as a platform which implements a set of software metrics definitions. It allows evaluating a software program according to the metrics by taking the requisite entities from the software and delivers the corresponding metrics values. In 1990, Chung and Yung introduced Readability Metrics [2]. To evaluate the complexity of software programs, Software firms use the software metrics for the cost estimation of the software projects, software reassurance, regulating the software development, for testing of the developed software, and software maintenance. Various software metrics evaluate the complexity of software by using some physical software characteristics. We typically classify the software entities that are used by software metrics for determining complexity in three categories: length, data flow, and control flow [2]. Each of the categories is associated with the physical aspects of software program. Readability Metrics is exceptional amongst the present software complexity metrics for considering a non-physical software characteristics i.e. readability. These applications of Readability Metrics are worthy for signifying the further efforts needed for less readable software programs, and helps in preserving the source code maintenance [3].

Code readability is the proficiency of software code which makes it legible and comprehensible even for a non-technical staff. Usually readability is measured by the ratio between number of lines of code and comments which are provided for the understandability of programmers and the machine do not understand them. That is, if without looking for the definitions or implementation of the language if we can understand the working of the code, the program is said to be readable. It is undoubtedly appears that readability is a characteristic related to reusability, maintainability, modifiability and robustness. In software maintenance phase, code readability is very significant. Analyzing the code by reading is first stage in software maintenance [4]. Therefore the code readability has much significance in software development cycle [5]. In this paper, we review the various code readability metrics presents in the literature and how readability effects on the development cost and how to increase code readability according to the metrics.

II. APPROACHES FOR READABILITY

In this section we will discuss traditional readability formulas which are namely ARI, SMOG, Gunning's Fog Index, Flesch-Kincaid Readability Index and Coleman-Liau Index. These are simple formulas that measure code readability on the basis of sentence length and word count or syllable count found in the text.

A. **The Automated Readability Index (ARI)**: Sentence and word difficulty ratios are used in ARI (automated readability index) [6]. Here word difficulty implies the total number of letters contained within a word whereas sentence difficulty implies the total number of words contained within a sentence. The syllable count is not reliable. The equation to compute readability with ARI is

$$4.71\left(\frac{characters}{words}\right) + 0.5\left(\frac{words}{sentences}\right) - 21.43$$

B. SMOG: G Harry Mclaughlin in 1969 proposed the readability metric named SMOG [7]. The term SMOG stands for Simple Measure of Gobbledygook. This metric evaluates the time (in years) required by any person to read the text. It is said to be an improved readability formula when compared with other existing metrics of that time.

SMOG = 3 + Square Root of Word Count

C. **The Gunning's Fog Index:** This metric was proposed by Robert Gunning [8]. The FOG metric value can be calculated by adding the average sentence length to the percentage of hard word. And the average sentence length is calculated by the ratio of words count to the total number of sentences.

FOG = 0.4 (ASL + PHW)

D. **Flesch-Kincaid Readability Index:** Flesch-Kincaid [9] check results specifies the reading ease of the given code, for a high value readability is high and for less value that implies code is hard to read.

$$206.835 - 1.015\left(\frac{total\ words}{total\ sentences}\right) - 84.6\left(\frac{total\ syllables}{total\ words}\right)$$

E. **Coleman-Liau Index:** Meri Coleman and T. L. Liau [10] give another readability index like ARI however different from all others to estimate the use of text. This index emphases on the letters per word however not on the syllables. The Coleman–Liau index formula is following:

$$CLI = 0.0588L - 0.296S - 15.8$$

Where L and S are average number of letters and sentences.

Although these traditional readability formulas have been widely criticized as being weak indicator as they do not consider the comprehension skills of the reader i.e. irrespective of the readers ability to comprehend the given text snippet, the calculation is completely based on the text structure. However, due to the simplicity of readability formulas, these are the widely used in the literature.

III. REVIEW AND DISCUSSIONS

In 1997, Chung Yung [11] proposed an approach that assimilates software metrics with the complexity of code, regarding the readability of the implemented algorithm. For the assessment of source code of the program, they proposed four different metrics that are; the unique number of operators, the unique number of operands, the number of total operator occurrences, and the number of total occurrences. The operators are the signs or groupings of signs which affect the values or serializing of operands, and the operands are the constants or variables. The inspiration is the association of code readability and software maintainability.

In 2006, Emilio and Valerdi [3] highlighted the role of code readability on software development cost. They analyze various software development activities and found that code readability has a widespread influence on the cost of software development and is not affected by the size of software. Furthermore, they discover the relations between software readability and domain knowledge of programming. Their conclusions determine that enhanced readability results in less reading time which subsequently means lower costs during every stage of the life cycle. Contrariwise, lesser readability results in more time spending on code reading which leads to higher cost. Increasing the code readability may improve the probabilities of reusing the code. The cost of redeveloping may be saved as the readability of existing code increases.

In 2010, Raymond Buse et al. [1] proposed a readability tool that calculates readability value. Relationship between errors or faults and evaluation with the code readability were studied. They establish a relationship between the size of code and the code readability as the size of code straightly effects the code readability; as it is easier to read short code as compared to large code. To justify the metric some java codes snippets were selected and the readability was tested with the proposed metric as well as human annotators. Readability metric value from their proposed metric was compared with the results obtained from the experts. The accuracy of the metric was found to be 80%.

In 2011, Daryl Posnett et al. [12], argue that Buse readability scores [1] were not collected from developers which were given an exact task, however from students who have no contribution in the code. More exactly they were not being evaluated on their aptitude skills to read. Though this, as such, cannot essentially have influenced the legitimacy of Buse's scores, it's somewhat imaginable that code readability means something different to a developer. Considering this, they specify that readability of code depends very much on the information containing in the source code. They proposed a model which depend on two main constraints size and entropy. Where entropy is measured from the counts of terms (tokens or bytes) in addition to the no. of unique terms and bytes. The more the entropy of the snippet greater the readable the code is. Up to a given entropy level, a rise in length of code certainly increase the readability. The simpler model of code readability proposed by them is:

$$z = 8.87 - 0.033\ V + 0.40\ Lines - 1.5\ Entropy$$

In 2011, X Wang et al. [13] specify the role of source code readability in the development of software quality. They emphasize on the fact that code readability is essential also in the later phases of Software development life cycle mainly in the maintenance phase. As maintenance phase affects the most of the software development cost. They asked some expert programmers to rate the code's complexity from open source snippets pertaining on the commands, statements, keywords, loops etc. which is then compared with the metric value of their developed tool which evaluate the readability of code. The tool proves out to be a really efficient than the human judgment.

In 2012, P. Sivaprakasam et al. [14] presented a programed system to enhance code readability in the program. They argue that blank lines could be added in source code to improve the code readability and the points could be located from the inner documentation. They presented a tool for the proposed method, which takes java methods itself as input and returns a readable source code by inserting blank lines after every valid code blocks. In this way code readability can be increased and it also helps for deciding the suitable place for the internal comments. Experimental results proves that the computerized insertion of blank line is as effective as blank lines added by human annotators.

ARI metric for code readability measurement is specified in [6]. The two characteristics were stated for ARI (automated readability index) for measuring the readability of the test snippet. First characteristic is the sentence difficulty which can be measured by calculating words per sentence [15]. And second characteristic is the word difficulty that can be measured by calculating the letters per word. By using the formula, the readability of the source code can be achieved.

SMOG, Simple Measure of Gobbledygook was suggested in 1969 by G Harry McLaughlin [7]. SMOG is used for computing code readability. This metric provides an expected level for reading and comprehending a snippet of code. SMOG results are measured by adding 3 in square root of the polysyllable count. It was said to be an improved readability formula when compared with other existing metrics of that time.

Robbert Gunning [8] presented another one readability metric called FOG. To calculate a FOG metric average length of sentences is added to the hard word's percentage. The average length of sentences can be calculated by dividing no. of words by the no. of sentences.

IV. CONCLUSION

Various software metrics are used in software development businesses to evaluate the software programs complexity for finding the software maintenance cost. In the paper we reviews the notion of code readability and study its significance in software quality and the software maintenance cost. The metrics of Code Readability have been outstanding in the present complexity metrics of source code for considering a non-physical software characteristics i.e. readability. The applications of Code Readability metrics suggests the extra efforts requisite for software systems that are less readable, and thus provide assistance in retaining the software systems maintainable. Yet the plentiful metrics and the complex formulas for the code readability commonly make it tiresome to relate Readability Metrics to huge scale software systems. Therefore various simplified readability metrics were proposed from time to time. One of our objectives is to review a readability model that indicates the code readability and the software systems' complexity by bearing in mind the readability difference in various different employments. Such that, we have a standard metric to specify the readability of the software code so as to keep it maintainable, and a benchmark for calculating the complexity of software program regarding their readability. There are some elements which make a program code easy to read, for example appropriate comments; whereas some others make a program code difficult to read, for example poorly defined variables. One of the normally familiar characteristics which make source code less readable is the very composite expressions. In such situations, dividing the highly composite expressions commonly aids in making the program highly readable. The applications of Readability Metrics support in retaining the source code of software programs readable for the software programs to be maintainable in the future stages of the software development life-cycle. Furthermore, we observes that readability displays an important level of association with another traditional metrics of software quality, for example defects etc. Additionally, we discussed how considering the aspects which affect readability has tendency to improve the programming practice relating to this significant aspect of software quality.

REFERENCES

[1]. R. P. L. Buse and W. R. Weimer, "Learning a metric for code readability", Software Engineering, IEEE Transactions, doi:10.1109/tse.2009.70, vol. 36, no. 4, **(2010)**, pp. 546-558.

[2]. C. M. Chung, W. R. Edwards, and M. G. Yang, "Static and Dynamic Data Flow Metrics," Policy and Information, Vol. 13, No. 1, pp. 91-103, June 2010.

[3]. E. Collar and R. Valerdi, "Role of software readability on software development cost", Proceedings of the 21st Forum on COCOMO and Software Cost Modeling, Herndon, VA, (2006) October.

[4]. R. Fitzpatrick, "Software quality: definitions and strategic issues", Reports, (1996), pp. 1.

[5]. R. Land, "Measurements of software maintainability", Proceedings of ARTES Graduate Student Conference, ARTES, (2002), pp. 1-7.

[6]. "The Automated Readability Index (ARI)", http://www.readabilityformulas.com/automatedreadability-index.php

[7]. "The smog readability formula", http://www.readabilityformulas.com/smog-readability-formula.php

[8]. "The Gunning's Fog Index (or FOG) Readability Formula", http://www.readabilityformulas.com/gunning-fog-readabilityformula.php

[9]. "Flesch-Kincaid Readability Index" http://www.mang.canterbury.ac.nz/writing_guide/writing/flesch.shtml

[10]. "Coleman-Liau Index" http://en.wikipedia.org/w/index.php?title=Meri_Coleman&action=edit&redlink=1

[11]. C. Yung, "Simplified readability metrics", Information Systems Working Papers Series, (1997).

[12]. D. Posnett, A. Hindle & P. Devanbu "A Simpler Model of Software Readability" MSR '11, Waikiki, Honolulu, USA Copyright 2011.

[13]. X. Wang, L. Pollock and K. Vijay-Shanker, "Automatic segmentation of method code into meaningful blocks to improve readability", Reverse Engineering (WCRE), 2011 18th Working Conference on IEEE, (2011).

[14]. P. Sivaprakasam and V. Sangeetha, "An accurate model of software code readability", International Journal of Engineering Research and Technology. ESRSA Publications,(2012)August, vol. 1, no. 6.

[15]. R. Namani1 and J. Kumar, "A New Metric for Code Readability", IOSR Journal of Computer Engineering, vol. 6, Issue 6, (2012) November-December.

[16]. K. Aggarwal, Y. Singh, and J. Chhabra. An integrated measure of software maintainability. In Reliability and Maintainability Symposium, 2002. Proceedings. Annual, pages 235-241. IEEE, 2002.

[17]. P. Sivaprakasam and V. Sangeetha, "An accurate model of software code readability" International Journal of Engineering, vol. 1, no. 6, (2012).

[18]. C. M. Chung, and C. Yung, "Measuring Software Complexity Considering Both Readability and Size," Infomration and Communication, Tamkang Univ., Taiwan.