Mining Software Repositories for Software Metrics (MSR-SM): Conceptual Framework

Tamanna Siddiqui, Ausaf Ahmad

Abstract: ***A number of software metrics estimate the complexity of software program by a couple of substantial software attributes and trends. Metric for measuring the reliability is extraordinary among the presently available complexity metrics by contemplating a non-physical attributes i.e. readability. Readability may be a key quality attribute for managing software source codes. Readability of the source code is mainly concerned with code maintainability which is a significant characteristic of software quality, mainly from the developers' perspective. As the code is readable, the easier it is to modify, lesser errors, copious maintainable, easy to reuse and even more reliable. Readability is employed to enhance source codes for subsequent maintenance and extensibility. But code readability is not simply computable while dealing with open source software as contributors access the code and modify its structure according to his/her style of coding. This nature of development begins problems for the new contributors to understand the code structure. To enhance code readability, In this paper, we proposed a conceptual model of mining software repositories for software metrics in which we proposed a set of metrics for readability of the code that is easier to use and helpful to understand the code structure. We also mention a mechanism to validate the proposed metrics by the data extracted from the mining software repositories and comparing it with a survey conducting from experts working in industries.***

Keywords: Mining Software Repositories, Software Metrics, Readability, Measurement.

I. INTRODUCTION:

Mining software repositories (MSR) are one of the interesting and fastest growing fields within software engineering [22]. Exact measurement is the primary stipulation for all engineering professions; software engineering is one amongst them. Engineers and researchers seek to manifest functionality of the software with numbers in the effot to assess software quality. To measure the software attributes, a large number of quality metrics have been proposed and tested. Several tools are also available to gather metrics from program delineation. This valuable collection of tools enables a user to choose the tool which is most appropriate for his requirements. However, this has assumed that most of the metrics tools measure, analyze and execute the same metrics in the same fashion.

In this paper, we describe a strategy to measure the complexity of software system by assuming their readability [24]. Chung and Yung first presented the readability metrics in 1990. Software industry makes use of metrics to determine the complexity of software to measure the development cost, development control, software testing, quality assurance as well as the maintenance [23] [3], [7], [4]. Nearly all software metrics gauge the complexity of software by one or more characteristics of the software system. Commonly software attributes classified into three categories that are used to gauge the complexity, these are size, control flow, and data flow [5]. These three categories are considered as physical activities of software development.

A.J. Albrecht and J. E. Gaffney [6] have considered the readability as excellent metrics among the available software complexity metrics and assumed a non-physical attribute. Use of readability metrics are a great practice in indicating the supplementary efforts essential for less readable software and assist to keep the software systems maintainable. However, due to a large number of metrics and complex formulation, it is exhaustive to employ readability metrics in large and complex software. In the proposed conceptual model, we formulate a simple strategy for readability measurement. In this we keep the number of required measures less for the readability. We hope that the experimental results will show that this simple strategy has the best predictive ability in determining software complexity in terms of readability and its ease of employing the architecture proposed. The exercise of applying readability metrics shows the ability of readability of software which in turn assists in retaining code readable and maintainability.

Readability attributes of software system offer something connected to software as well as its quality. In reality, readability offers judgment of peoples in terms of easier reading and understanding of particular source code. This metric possibly endorses ease of maintainability and entire quality of the code. As it is mentioned in [8], maintenance is a difficult and costly task as it takes 70% of its whole development cost.

Aggarwal et al [11] states that maintainability is the most critical task of SDLC which consumes most of the time, efforts and cost [9], [12], [14]. According to Marctty and Elshoff [13], readability of code plays a valuable role in such a manner that when some additional functionality is added during maintenance of SDLC, it is to be considered as code readability improvement phase. Knight and Myers [16] suggest that software inspection phase ought to be checking out the source code to confirm the code readability. This mechanism of development will ensure the ease of maintainability, portability as well as the reusability of source code. Haneef [17] worked out to extent to include a documentation team in his development team with the intention that well-established guidelines of readability can benefits the reviewers of the code. As explained in [8] and [20], developers have certain sort of intuition toward the concept and characteristics of the program, so readability becomes important and consequently, comments in the codes endorse readability. Dijkstra [18]

Revised Manuscript Received on August 05, 2019.

Dr. Tamanna Siddiqui: Associate Professor, Department of Computer Science, Aligarh Muslim University, Aligarh, UP, India

Ausaf Ahmad: Research Scholar, Department of Computer Science, Aligarh Muslim University, Aligarh, UP, India.

states that readability of source code is influenced by many factors, for example, ease of control sequences, comments, approach adapted and so on.

This paper is arranged into four sections: Section 2 describes some metrics related to readability. Section 3 describes the proposed framework used to mining software repositories for software metrics and last section concludes the proposed framework and the presented research work.

II. RELATED METRICS FOR READABILITY

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

I. The Automated Readability Index (ARI):

Sentence and word difficulty ratios are used in ARI (Automated Readability Index) [19]. 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

$$ARI = 4.71\left(\frac{Characters}{Words}\right) + 0.5\left(\frac{Words}{Sentences}\right) - 21.43 \quad (1)$$

II. SMOG: G Harry Mclaughlin in 1969 proposed the readability metric named SMOG [10]. 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.

$$SMOG = 3 + Square\ Root\ of\ Word\ Count \quad (2)$$

III. Gunning's Fog Index: This metric was proposed by Robert Gunning [1]. 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.

$$FOG = 0.4(ASL + PHW) \quad (3)$$

IV. Flesch-Kincaid Readability Index: Flesch Kincaid [15] 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

$$206.835 - 1.015\left(\frac{Total\ Words}{Total\ Sentences}\right) - 84.60\left(\frac{Total\ Syllables}{Total\ Words}\right) \quad (4)$$

V. Coleman-Liau Index: Meri Coleman and T. L. Liau [21] give another readability index like ARI. However, it was different from all others in estimating the use of text. This index emphasizes on the letters per word however not on the syllables. The Coleman–Liau index formula is:

$$SLI = 0.0588L + 0.296S - 15.80 \quad (5)$$

Although these traditional readability formulas have been widely criticized as being a weak indicator as they do not consider the comprehension skills of the reader i.e. irrespective of the reader's 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 widely used in the literature.

III. PROPOSED FRAMEWORK

Size is among the most effective feature of software systems. It influences the development cost and manpower. The size of the system also has a great influence on the maintenance of the system. Size dependent metrics show the complexity of the developing system, particularly by its size characteristics. Most of these size base metrics aid in predicting the expenses on maintenance of the software system and during the maintenance readability plays an extraordinary role. In order to measure and improve the readability of codes, we proposed a conceptual framework that is depicted in figure 1. In this conceptual framework, we categorize each necessary module which will come under the proposal of metrics and for its validation. The framework is mainly divided into three phases. Initiation phase which describes the selection of software repositories, Implementation phase which includes the code extraction process, proposed metrics and its validation with extracted source code and finally, reporting phase which deals with the comparison of outcomes of proposed metrics to industry scope, and the recommendation for future release. Each phase is discussed in detail in the following sections.

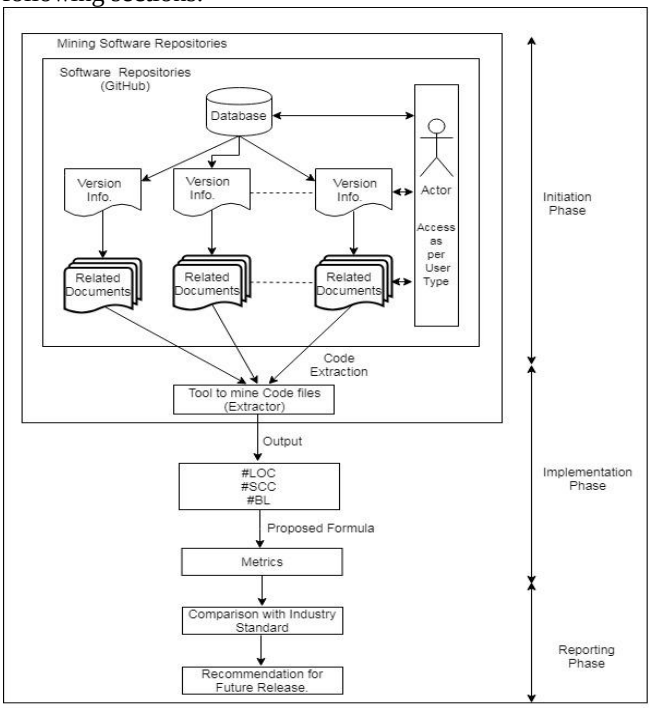

Figure 1: Mining Software Repositories for Software Metrics (MSR-SM): Conceptual Framework

3.1. Phase-I Initiation Phase

In Initiation Phase, we describe the structure of software repositories in detail and determine the selection of repository.

***Software Repositories**:*

Software repositories are usually a storage location where the source codes and related documents of softwares are maintained. From these repositories, we could download, source code, configuration files and related items of the software maintained in this

repository. In sync with development perspective, software repositories also contained a variety of information about the software system and its development which can be retrieved, edited and used by the developers. Mainly software publishers and software organization developed and maintain such repositories online, either in open source mode or subscription mode. These repositories contain a variety of project data which may be utilized for understanding the nature and structure of projects. Currently, this field is taking a great importance within the software researcher community.

3.2. Phase-II: Implementation

Implementation phase covers the extraction of code information from repositories and proposed metrics along with its validation with extracted data from repositories. All the modules within this phase are described in detail below.

Code Extraction:

As we mentioned above that software repositories contain valuable information which can be used for better software development by the different development personnel at different stage. The information can be requirement documents, source code, comment in the source code, bugs information, release of software etc. Researchers mine data and metadata in a software repository to extract pertinent information and/or uncover relationships or trends about various characteristic. For example, one may be interested in the growth of a system, others in change of relationship between source code entities and some in reuse of components.

Here, we have extracted the number of lines of source code, the total number of single line comments and multi-line comments and also the number of blank lines to determine the exact lines of code. This further will be investigated to get the variation between these metrics in incremental versions. This extracted information will be used to validate our proposed metrics.

Metrics Derivations:

This module refers to the derivation of several (Set of metrics) metrics. These metrics can be used to measure the readability of software and change of readability metrics of the code for the software across versions. This readability metric can help the developers and managers to easily find the best way of coding practice. Accordingly, developers can adapt the code; work on it to enhance its functionality according to their needs and maintain it easily. While taking on overall development perspective, this situation leads to time and cost saving.

Proposed Metrics:

We have worked on version control of the software system to measure Code Readability of a project based on the source line of code (LOC) and source code comments (SCC) contained and define as

$$\textbf{Code Readability Index } (CRI) = \frac{\#SCC}{\#LOC} * 100 \quad \% \qquad (1)$$

Where #LOC is the total number of lines of code, #SCC is the total number of source code comments. Here, #SCC contains the single lines of comments and multiple lines of comments excluding blank lines. #LOC contains physical lines of codes excluding blank lines and braces.

We also measure the enhanced readability index between two versions of the software project to check how readability changes from one version to the next for better readability.

$$E_n = RI_n - RI_{(n-1)} \qquad (2)$$

Here, E_n indicate the enhanced CRI RI_n and $RI_{(n-1)}$ are the code readability index of n^{th} and $(n-1)^{th}$ versions of software program respectively.

Another proposed metric to enhance the readability of the code component is to count the average number of code lines between two set of comments. We also measure the average number source code of comments in the codebase of a software system by measuring the number of code between every two consecutive comments.

$$Avg\_LoC = \frac{nCl_{1,2} + nCl_{2,3} + \ldots\ldots\ldots + nCl_{(n-1),n}}{\#SCC} \qquad (3)$$

Where $nCl_{1,2}$ indicate the number of lines of source codes between first and second comments in codebase. Similarly $nCl_{2,3}$ the number of lines of source codes between second and third comments and so on. We have also measured the change average number of source code comments from one version to its previous version by Eq. 4

$$(Ch\_Avg\_LoC)_n = (Avg\_LoC)_n - (Avg\_LoC)_{(n-1)} \qquad (4)$$

Ch_Avg_LoC indicates the number of lines of codes change from one version to next version.

We check how comments between codes added or deleted with addition and deletion of function in the release of a version with its previous one. We hope that the proposed metrics will help to measure the readability and will be popular because this is simple in computation and easy to understand. Also, these metrics will promote code reusability and ease of maintainability in cost effective mannar and in least time frame.

3.3. Phase-III Reporting Phase

Reporting phase incorporates the results calculated from the proposed metrics and its comparison with the IT industry standard by conducting a survey from the developers working on open source projects. It includes two modules i.e. comparison with industry standard and recommendation for future release.

Comparison with Industry Standard:

As our proposed Code Readability metrics provides the number of comments per hundred lines of code to understand the structure and functionality of the programs. However, the industry scope provides the expected number of comments between a hundred lines of codes. We involved industry scope to keep the readability index near to threshold value i.e. taken from the developers and managers of software projects working on open source software. While *Average SCC* metrics, provide a mechanism to

write at the average number of position of code. We will check the role of *Average SCC* in enhancement of code readability by involving the developers from the industry.

The need for involvement of Industry Standard:

Andrey Nikishaev a developer, states in his article [2] that comments shows the bad code. Good code can be understood without comments. But his practice arises problem to new developers when the size of the software system becomes complex. Writing comments between codes explaining what and how the code works are deemed to be a good practice. This practice will help to understand the code in less time. It gives chance to the new developers to effectively understand the code and the better his implementation of code and maintenance. It will be a good initiative for global code documentation.

When we talking about the open source software, the term "open source" refers to something people can copy, study, modify according to their need and share, because its nature is publicly accessible. In addition to this, when someone contributing in any software release they add comments according to their personal interest and understanding which may be large or may be too small depending on the developers coding style. This insufficient number of comments may raise difficulties to new contributor/developers. Same difficulties may arise with a huge number of comments. So, to maintain a balanced ratio between source codes and comment we involve industry experts. Here, we try to suggest a standard architecture to code structure which provides better readability of code, better possibility of understanding, maintaining and reusability of codes becomes higher.

Recommendation for further Release:

The recommendation module is an important part of the proposed framework which provides a guideline to maintain the number of comments per hundred of lines of codes. This module tries to give a standard to code so that new developers can easily understand, work on it and maintain it easily. By accepting these recommendations, the contributors/developers can keep the ratio of code and comments in sync with the software industry best practices. We suppose that the results of this practice would help to provide a standard to code particularly in the open source environment.

IV. CONCLUSION

Software metrics measure the software attributes from starting of its development to final product from different perspective (development personnel's and user perspective) that can be quantified or is countable. Imposing software metrics in development as well as in maintenance is good practice for many reasons, including software performance, planning work stuff, productivity, and other involved activities. Readability is an additional valuable attribute of software that provides an extensive impact on software maintainability. The software systems with less readable source code are recognized as more difficult to maintain as compare to those with more readable source code. In this paper, we have proposed a conceptual framework of mining software repositories for software metrics. This model is divided into three phases - initiation phase, implementation phase, and reporting phase. Each phase is described in detail in this paper. We have also proposed a set of metrics related to the readability of codebase of software which helps the developers to easily understand the code structure, reuse the codes and maintain it. We have also proposed a mechanism to validate the proposed metrics with the industry standard. The extension of this paper will be the implementation of our proposed framework.

REFERENCES

1. The Gunning's Fog Index (or FOG) Readability Formula", http://www.readabilityformulas.com/gunning-fogreadabilityformula.php, retrieved December, (2018).
2. Andrey Nikishaev, "13 Simple Rules for Good Coding", https://hackernoon.com/few-simple-rules-for-good-coding-my-15-years-experience-96cb29d4acd9.Retrieved January, (2019).
3. J. E. Ganey, Metrics in software quality assurance, Proceedings of the ACM CSCER '81 conference, pp 126-130, (1981).
4. D. N. and W. W. Agresti. "Comments on Resolving the Software Science Anomaly." J. Syst.and Software 7 (1), pp. 83-84, (1987)
5. Muhamediyeva, D. T., Abduraimov, D., & Primova, K. A. (2016). Development of software complexes of text data recognition. *BEST: International Journal of Management, Information Technology and Engineering (BEST: IJMITE); ISSN (Print): 2348-0513; ISSN (Online): 2454-471X; Impact Factor (JCC): 1.5429; Index Copernicus: 3.0*, *4*, 27.
6. R. S. Pressman, Software Engineering: A Practitioner's Approach,McGrawHill, (1996).
7. Albrecht, A. J. and J. E. Gaffney, Jr. "Software Source Lines of Code, and Development Effort Prediction: A Software Science Validation ".IEEE Trans. Software Eng. SE-9, 6 (Nov. 1983)
8. Arthur, L. J. Measuring Programmer Productivity and Software Quality. New York: John Wiley, (1985).
9. B. Boehm and V.R. Basili, "Software Defect Reduction Top 10 List," Computer, vol. 34, no. 1, pp. 135-137, Jan. (2001).
10. L.E. Deimel, Jr., "The Uses of Program Reading," ACM SIGCSE Bull., vol. 17, no. 2, pp. 5-14, (1985).
11. The SMOG Readability Formula, a Simple Measure of Gobbledygook. http://www.readabilityformulas.com/smog-readability-formula.php, Retrieved January, (2019).
12. MOHAMED, S. I. INNOVATIVE SOFTWARE DELIVERY FRAMEWORK TO WARDS SOFTWARE APPLICATIONS MODERNIZATION.
13. K. Aggarwal, Y. Singh, and J.K. Chhabra, "An Integrated Measure of Software Maintainability," Proc. Reliability and Maintainability Symp., pp. 235-241, Sept. (2002).
14. D.R. Raymond, "Reading Source Code," Proc. Conf. Center for Advanced Studies on Collaborative Research, pp. 3-16, (1991).
15. Lavrischeva, E. M. (2018). The Scientific basis of Software Engineering. *International Journal of Applied and Natural Sciences (IJANS)*, *7*(5), 15-32.
16. J.L. Elshoff and M. Marcotty, "Improving Computer Program Readability to Aid Modification," Comm. ACM, vol. 25, no. 8, pp.512-521, (1982).
17. S. Rugaber, "The Use of Domain Knowledge in Program Understanding," Ann. Software Eng., vol. 9, nos. 1-4, pp. 143-192, (2000).
18. Flesch-Kincaid Readability Index "http://www.mang.canterbury.ac.nz/writing_guidewriting/flesch.shtml. Retrieved December, (2018).
19. J.C. Knight and E.A. Myers, "Phased Inspections and Their Implementation," ACM SIGSOFT Software Eng. Notes, vol. 16, no. 3,pp. 29-35, (1991).
20. N.J. Haneef, "Software Documentation and Readability: A Proposed Process Improvement," ACM SIGSOFT Software Eng. Notes, vol. 23, no. 3, pp. 75-77, (1998).
21. E.W. Dijkstra, A Discipline of Programming. Prentice Hall PTR, (1976).
22. TAMIMI, M., ALGHAMDI, F., & YASEEN, A. A SYSTEMATIC SNAPSHOT REVIEW OF CUSTOM-MADE SOFTWARE ENTERPRISES FROM THE DEVELOPMENT PERSPECTIVES.
23. .The Automated Readability Index (ARI),

http://www.readabilityformulas.com/automated-readability-index.php , retrieved January, (2019).
24. P.A. Relf, "Tool Assisted Identifier Naming for Improved Software Readability: An Empirical Study," Proc. Int'l Symp. Empirical Software Eng., Nov. (2005).
25. Coleman-Liau Index" http://en.wikipedia.org/w/index.php?title=Meri_Coleman&action=edit&redlink=1, Retrived, December, (2018).
26. T. Siddiqui and A. Ahmad. "Data mining tools and techniques for mining software repositories: A systematic review." In Big Data Analytics, pp. 717-726. Springer, Singapore, (2018).
27. Ausaf Ahmad, Tamanna Siddiqui, Najeeb Ahmad Khan, "A Detailed Phasewise Study on Software Metrics: A Systematic Literature Review ", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN: 2456-3307, Volume 3, Issue 3, pp.1696-1705, (2018).
28. T. Siddiqui and A. Ahmad, "Complexity Clarification through Code Metrics" Proceedings of the 12th INDIACom; INDIACom-2018, 5th International Conference on "Computing for Sustainable Global Development", pp.3746-3749, March, (2018).

AUTHORS PROFILE

Dr. Tamanna Siddiqui is presently working as Associate Professor in the Department of Computer Science, Aligarh Muslim University, Aligarh (UP). She obtained her B.Sc. (Hons) and MCA from AMU, Aligarh, and Ph.D. (Computer Science) from Jamia Hamdard, New Delhi. Her Research Interest includes data mining, big data, Software engineering, cloud computing, soft computing, etc. She has rich 20 years Teaching experience which includes national and international universities like Jamia Hamdard (New Delhi), University of Dammam (KSA) and Aligarh Muslim University (AMU). She has performed different administrative responsibilities apart from teaching and research. She has published many books and she has a rich number of research papers in well reputed international journals.

Ausaf Ahmad is currently working as a research scholar in the department of computer science, Aligarh Muslim University, India. His research interests are in software engineering and mining software repositories. He received his MCA and B.Sc.(Hons) degree from Aligarh Muslim University, Aligarh, India.