Empirical Validation Of MSR-SM Framework

Ausaf Ahmad, Tamanna Siddiqui

Abstract: The readability of a program is concerned with its maintainability. The maintenance of software generally stands for evolving software, and even customizing existing code is a significant practice of modern software engineering. Readability also correlated with software quality, code change, and defect reporting. Code readability is a very decisive factor in software development. Code readability mainly provides a mechanism to ease of maintainability and code reusability. In order to measure the readability of code, we proposed readability metrics. In this paper, we validate our proposed MSR-SM Framework for the above purpose. An empirical study is conducted to achieve the aims of the study. We have written a code to extract the code information stored in historical repositories to collect the data for validation practice. We have extracted the data from a series of versions of the program to examine the variation in the ratio of source codes and comments between them. Contribution- This study comprises of a significant amount of empirical details related to association between code attributes and readability. We recognize that these details probably would have implications for the style of coding and its evaluation as well as for the design of programming.

Index Terms: Software Mining, Software Metrics, Mining Software Repositories, MSR-SM Framework, Maintenance, Code Readability, Software Development.

1. INTRODUCTION

Mining software repositories (MSR) is one among of the exciting and most rapid growing disciplines of software engineering. [15]. Quite a lot codebase of software projects comprises of comments that record the deployment of the codebase, and it helps the contributors to comprehend the code structure, for modification and code reuse mainly. A number of experts have conducted studies expressing that code having comments message is less complicated to understand than code without having message i.e., comments [10], [11]. Comments are the subsequent best used scripted thing for understanding the code, after the code [12]. Additionally, a source code document is also imperative in maintenance and form a mainstay of the general documentation of the software projects. Contrary to exterior documentation, comments in the codebase are a straightforward mode for contributors to preserve related documents and keep the code consistent and up-to-date. Contributors generally admit that an inadequate document creates confusion [13], and studies mentioned in [14] state that poor documentation significantly have the least capacity of maintainability of the system. While contributors ordinarily concur on the interest of software manual [12], that putting the comments in code is usually ignored mainly because of strict release deadlines along with time constraints throughout the development. In the past few decades, software engineering strategies get continuously grew to bring cultural, social, technological, and organizational improvements. This improvement is a change in development strategies and methods from document driven to agile evolution that concentrates on developing the best software system instead of developing best documents [1]. This progressive nature of changes results to scenarios wherein source code and comments usually the sole existing system artifacts having code layout, execution, and implementing the decision making. Studies and research have shown that the efficacious usages of the message in the form of comments can substantially increase the programmers' understanding about codebase [2]. Up till now, the extent of a research study committed regarding the assessment of nature, quality of in-line documentation is bounded [3].

Software development is not an independent exercise; it consists of teamwork. For maintenance of any types, developers need to comprehend their own code as well as another developers' code of their team. This indicates that the code required being written well and also understandable. This strategy, particularly pertinent to open source programming codes and projects. While fresh developers assign to an existing project may not have sufficient expertise, could be not very familiar with the current codebase, additionally they could be possibly unwilling to inquire another developer to clarify the working of code. Considering as not strange that contributors have recognized readability and understandability as one of the best particulars must have code which is challenging to examine can lead to broad-reaching outcomes. It might extend to alterations becoming sub-optimal that present possible errors. This is recognized that the maintenance of codebase turns to difficulties as the projects grow as well as the employee dimensions increase [8]. Academic experts have examined the consequences of codebase readability. Beacker et al. [33] investigate the association among the readability and comprehension. Experts from various fields also observed that comments and perfect word identifiers perform a significant segment in comprehending the codebase [14],[27],[28]. Code readability is generally supposed to influence the entire degree of excellence of the software as well as the development process. Exiguous readability not just only prevents the contributors to recognize the working of codebase nevertheless also might cause contributors to commit sub-optimal modifications and generate errors. Contributors likewise identify this threat as well as point out that readability is prime fact requires. The contributions of this study are the following: The paper is organized as follows. Section 2, discussed the related work to readability, section 3 address the impact of readability, section 4 regarding the implementation of MSR-SM framework, section 5 discussed the results and discussion and section 6 conclude the work.

2 BACKGROUND ON SOFTWARE READABILITY

Probably the key complexities experienced by the IT industry are the hefty price of software development, particularly throughout the continuance maintenance that consumes 60% to 70% of development cost. This heavy cost of maintenance

- *Ausaf Ahmad is currently pursuing Ph.D. degree in Computer Science from the Department of Computer Science, Aligarh Muslim University, Aligarh, India. PH-+919027789755, E-mail: ausafahmad.cs@gmail.com*
- *Tamanna Siddiqui is currently working as Associate Professor at the Department of Computer Science, Aligarh Muslim University, Aligarh, India. E-mail: ja_zu_siddiqui@hotmail.com*

counties to be associated with the complexity of reading and understanding the codebase, especially code written by other developers. A newly released research on readability of programing code exerts interest on developing readable code, mainly textbase readability, as legitimate factors in programming codebase understanding [7].

2.1 Programming Code understanding referred to the procedure through which a developer develops sense and understanding of programming code. They [6] examine the consequences of the propositional organization of programming code on the code readability, i.e., simplicity of comprehension. They suggest that programming might be understood by its propositional construction. Propositions are made from propositional factors in code to create a textbase, the ordered list of propositions that symbolizes the fundamental propositional organization of the codebase.
2.2 Textbase Readability is considered as the usefulness of the related association of programed codebase for textbase understanding. Through this scene, the information ascertains script readability is not easily the linear demonstration of the code structure. Understanding the textbase stage will depend on the efficacy of the propositional enterprise is backing the readers' capability to build semantics from the word present in comment sentence of mentioned in codebase.
2.3 Code Readability of codebase is an essential matter for programmers. A codebase that is readable is mostly viewed as more maintainable; the more readable code is expected to persist convenient to read, understand and maintain in future. We consider the readability of code as a subjective impression that developers have the trouble of code as they seek to comprehend it. The association between readability and comprehending is equivalent to syntactic and semantic investigation; readability is a syntactic attribute and comprehending as a semantic attribute. Essentially, readability is supposed to a shield to understand that the programmer thinks the need to conquer before getting started working with codebase: the enough readable and comprehensible it is, lesser the shield. Elshoff and Marcotty [5] incorporated that a large number of commercial codebases were considerably hard to read and suggested the addition of a new phase to development phase that made the code extra readable. Haneef suggested that contributing devoted readability as well as proofing team in the development team that individual viewer of code might not be capable of contributing enough without confirmed and disciplined guiding principle for readability. Knight and Myers [30] recommended that the inspection segment should be added in the development phase to examine the readability of the source code to guarantee maintainability, reusability, and portability of the code. The Flesch-Kincaid Grade Level[26], the Gunning-Fog Index [27], the SMOG Index [28], and the Automated Readability Index (ARI) [29] are a quite few samples of readability metrics for plain word available in codebase. All these metrics are generally calculated on fundamental components present in the comment, for example, approximate syllables per word and approximate sentence size. These metrics might assist the development team to get some confidence that their documents reach the targets of readability quite cheaply, and turn to comprehensive for that purpose. We think that likewise metrics, focused particularly to source code and assisted with practical confirmation for performance and efficiency, can assist as relevant intent in the software engineering field. It is essential to observe that the readability is something different than complexity [17], wherefore certain available measures and metrics have been confirmed useful practically. Brooks [32] assures that complexity as "essential" attribute of any codebase; It comes from system requirement and can't be abstracted away. His model considers readability an accidental attribute simply because the problem statement does not measure it. In basic concept, software engineers focus first to manage accidental challenges and complexities. This suggested that readability tends to be tackled without difficulty compared inherent complexity. Although program complexity metrics generally consider the extent of classes, procedures as well as collaborations between them, while the program readability relies mostly on internal, line by line aspects.

3 IMPACT OF READABILITY

Industrial and academic experts are searching to strategies for measuring the readability code readability and their influence on quality. Buse et al. [9] proposed the first model to assess the codebase readability. They explained readability as “a human judgment of how easy a text is to understand" driven by the data observed by Survey. Readable code impact on SDLC process, development cost, understanding code easily and maintenance as well.

Impact of Readability on development cost

The issue of costly maintenance continues to point out by experts, researchers, and professionals [16, 5]. Collar jr, E. and Valerdi [7] states that most of the effort and time spend during maintenance activities on code reading and comprehension. Rugaber [8] has observed that present practice exercised, program comprehending generally comprise of code reading. Programmers working on maintenance responsibilities are relevant to that of archeologists that examine certain scenarios, attempting to know what they investigating furthermore how all of it suits collectively. To accomplish, they need to be attentive to safeguard the software artifacts they discover, admire as well as comprehend the cultural factors that created these. They recommend that the impacts of code readability on program expenses are bipolar. Definitely, our studies confirm that superior readability results in minimum efforts and time consumed on code evaluation and comprehend its structure. Accordingly, leads to less expensive cost during every phase of life-cycle. Contrarily, less readable coding practice tends to additional time required to interpretation the code leads to higher development cost.

Evolution of Code and Comments

Khamis et al. [1] studied the improvement of message comments over time to time to scrutinize the comments and find that programmers modify code without swapping the associated comments which is supposed to lead to errors. Although, the analysis discloses that programmers modify function comments continually. Likewise, Fluri et al. [4] assume that code and comments are not evidently modified simultaneously and examine how they change. The investigation indicates all comment alters together with the exact code alterations, 97% are finished, around 97% are in an identical version as the codebase alter.

4 IMPLEMENTATION OF MSR-SM FRAMEWORK

This paper is an extension of our previous work [19] where we have proposed a framework for mining software repositories for software metrics. In this paper, we have discussed all the phase and steps in detail. Algorithm for mining software repositories has been proposed and implemented. The output of this algorithm has been used as input of for another algorithm proposed, which generates different readability metrics. Our proposed framework is separated into three phases: Initiation, Implementation and reporting phase.

4.1 Initiation phase

The initiation phase of the framework discussed the structure of repositories available and the type of data available in that repository. We discussed all these in detail MSR-SM framework. The general structure

Mining Software Repositories (MSR)

MSR discipline inspects the wealthy data present in software repositories to find out the exciting and actionable knowledge about software projects and development practice. The data available are version control, mailing archive, issue tracking system and other types of development data.

Software Repository (SR)

SR is usually a storage location where a lot of software packages and its source code are maintained from where these things could be retrieved, installed on computers. When talking with a development perspective, software repositories also contained a variety of information about the software system and its development which can be retrieved, edit and used by the developers. While taking to the information contained in repositories, they contain different version of software system i.e. concurrent versions system, bugs tracking information, deployment documents, etc. from here develops extract the information of their need and utilize it for developing understanding to take further course of action. Figure 1 presented in our previous work depicts the clear structure of the SR. for implementation; we select GitHub repositories from which we extract the details of OpenSSL and Vim Editor. We used four latest versions of these two repositories. The details of the software are discussed in results and discussion section.

4.2 Implementation Phase

Code Extraction

The second phase of MSR-SM is the implementation phase. It covers details of repositories used, extraction of code information from repositories and proposed metrics along with its calculation of metrics values from the extracted information from the repositories. For implementing MSR-SM framework, we use repositories of OpenSSL software and Vim Test Editor Software. The algorithm used for mining these repositories is given below

Algorithms for MSR

Our proposed readability metrics are based on mainly two physical things that codebase have, one is Lines of code and the second is Comments. Blank lines also help to increase the readability but neglect it due to consider it as non-physical things. In counting the number of comments we take both types of comments, single line comments having syntax //….and multiple line comments having syntax /*…*/. The algorithms to extract this information from software repositories have been given below.

```
Begin
Get input to base directory having all files
While (files or sub_directory)
{
Open (file_n)
{
while (EOF)
        {
        if (character = \n and next_character = \n)
                blank_line++
            else if (character = / and next_character = /)
                single_line_comment++
            exit line;
        else if (character = / and next_character = * and
        last_character_of_line = /)
                single_line_comment++
            exit line;
        else if (character = / and next_character = * and
        last_character_of_line = !/)
                multiple_line_comment++
                  find next occurance of */
        exit line;
        else
                  code_line++
        }
}
}
   End
```

The proposed metrics that are presented in MSR-SM are implemented by the algorithm given below.

```
Begin
{
Get #SCC;     // from the above algorithm
Get #LoC;     // from the above algorithm 1
CRI = (#SCC/#LoC)*100;
Avg_LoC = (#LoC/#SCC);
}
End
```

This algorithm gives the value of proposed metrics for a specific version of the software. We have applied above two algorithms on OpenSSL Repository and Vim Editor Repository. The detailed results are presented in results and discussions section.

4.3 Reposting phase

The Reporting Phase is the last phase of MSR-SM framework consists of two modules: comparison with Industry Standard in which we try to find out a common standard for readability, i.e. Industry Standard (IS), for this we conduct a survey with the people working in software industry to get a readability scale for better understanding of codebase. Then we compared the calculated results with IS and second module in this phase is concern as a recommendation for future release which is discussed in last of results and discussion section.

5 RESULTS AND DISCUSSION

To implement the MSR-SM framework we have used the following repositories

Repositories Used

We have extract software data from GitHub software repositories [18] that contain a variety of information about projects maintain online. It provides the distributed version control system and source code management facilities of Git with adding its attributes and features. It offers access control and numerous collaboration features like for example bug tracking, activity management, feature request and report of each project it contained. We select the source code of one module of Vim Editor and OpenSSl. Both these selected data sets are written in C language. The detail description of each these datasets are given below.

OpenSSL

OpenSSL is a software library that protects and secures communications over the internet averse eavesdrop. This frequently used in web-based services [20]. It offers an open source execution of SSL and TLS and enables fundamental cryptographic activities and also other various functions also. There are a series of the version of OpenSSL online maintained at [24] as open source. We select four latest versions of its fips module to measure the readability. The extracted information is given in Table 1.

TABLE 1 *DATA EXTRACTED FROM OPENSSL*

Dataset	Version	Size	Source Lines	Blank Lines	Single Lines (//…..)	Multiple Lines (/*….*/)
OpenSSl	1	2622	2319	151	0	162
	2	1798	1571	108	0	99
	3	1388	1200	88	0	67
	4	2190	2190	140	0	160

Vim (Text Editor)

Vim is a text Editor available for both for command line as well as standalone service in graphical interface [21]. It is an open source application and launched under a license. It is written in C programming language. In 2006, it had been observed a most famous text editor by the Linux Journal viewers; in 2015, stack Overflow contributors survey observed it as the third most favorite text editor [22]as well as the fifth most favorite development environment in 2018 [23].

TABLE 2 *DATA EXTRACTED FROM VIM EDITOR*

Dataset	Version	Size	Source Lines	Blank Lines	Single Lines (//..)	Multiple Lines (/*..*/)
Vim Editor	1	6162	4968	499	1	595
	2	6289	5027	535	7	615
	3	5929	4740	495	24	548
	4	6369	5136	514	1	611

Results of Metrics

Through the literature review, I found that readability is a very important factor or attribute that plays a vital role in maintenance and reengineering. The motive of our proposed metrics measures the readability of codebase of programs, based on the comments mentioned in that codebase and try to provide a balanced number of comments to understand the working of functions of that codebase easy. We have proposed two metrics namely: Code Readability Index (CRI) and Average LoC (Avg_LoC) between two Comments and base on these two metrics we try to find the Enhanced CRI and change Avg_LoC. Enhance version our metrics indicate how much readability increased from the previous version to current or how much readability should increase from the current version to next released so that the contributor/developers can easily and clearly understand the working of the codebase.

TABLE 3 *CRI AND AVG_LOC OF OPENSSL*

Dataset	Version	CRI	Avg_LoC
OpenSSl	1	6.99	14.31
	2	6.30	15.87
	3	5.58	17.91
	4	7.31	13.69

TABLE 4 *CRI AND AVG_LOC OF VIMEDITOR*

Dataset	Version	CRI	Avg_LoC
VimEditor	1	8.34	12.00
	2	8.08	12.37
	3	8.29	12.07
	4	8.39	11.92

To entice a common conclusion for the efficiency of our proposed readability metrics, we have calculated the readability of two different applications, the details of each data set is discussed in detail in the above section. We have calculated the readability for the latest four versions of both applications. Figure 1 and figure 2 based on table 3 and table 4 clearly depicts how readability propagates from one version to another version.

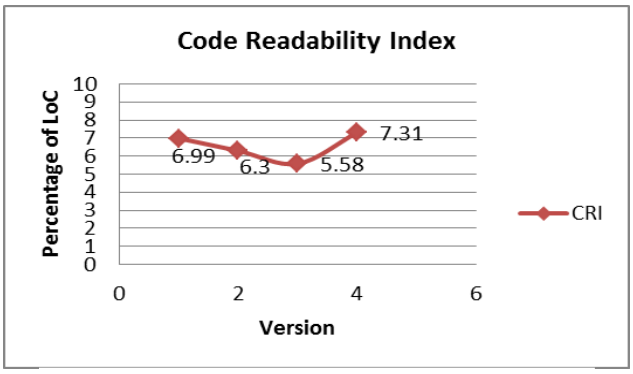

Fig. 1*. Code Readability Index for OpenSSL*

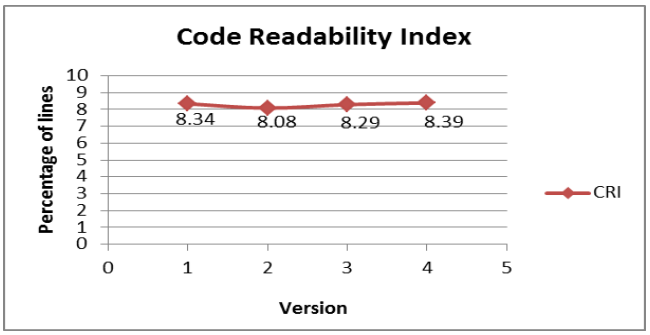

Fig. 2*. Code Readability Index for Vim Editor*

Avg_LoC metrics measure the number of lines of codes between two comments for the entire codebases. This metric offers a milestone that if we distribute the comments on average with code what should be the location of comments. From these metrics we can generate a common location for comment in the codebase. Figure 3 and figure 4 show the average code between two comments in the codebase of OpenSSL and Vim Editors respectively and also depicts how it changes from one version to the next version. In figure 3 we see that variation in average code from 1 to 4 versions while figure 4 there is slightly change in versions.

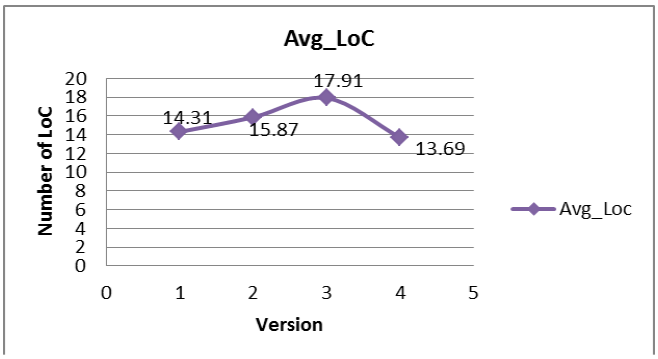

***Fig. 3**. Average Line of Codes between Two Comments for OpenSSL*

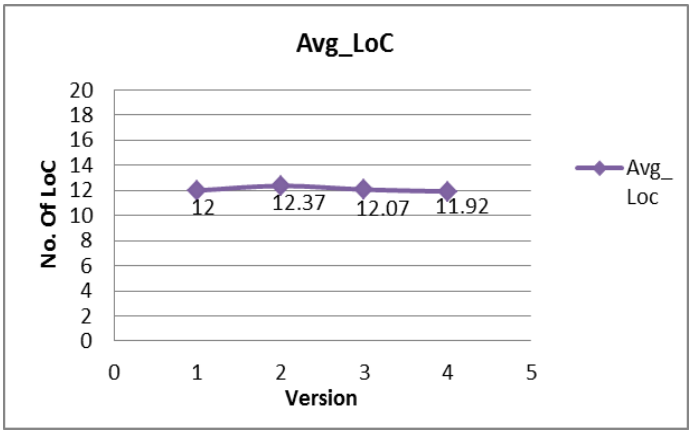

***Fig. 4**. Average Line of Codes between Two Comments for Vim Editor*

CRI and Avg_LoC metrics are nearly related to each other in an inverse manner. As the value of CRI increases, the value of Avg_LoC going to decreases. We review a set of metrics which used to measure the readability of codes, but they are in used because these metrics are hard to understand and apply. Our proposed metrics are easy to understand and apply.

Industrial Standard

Our proposed metrics provide an Index but does not a clear understanding that which types of the code is? Also, there is no scale available which provide a type of code. So we try to devise a scale that describes the type of code and accordingly developers can put the comments in a balanced manner with code. The need of this industry is discussed in detail in our preview paper [19]. Actually, our motive is to write the code that can easily understand the working of codebase because codebase having only code, can be understood by the machine. To reach this objective, we conducted a survey with software development personnel in which project managers, developers and quality analysts who are currently working in companies were involved. A number of respondents were involved from nine different reputed companies, and their details are given in the table in 3.

***TABLE 5** CHARACTERISTICS OF RESPONDENT PARTICIPATED IN THE SURVEY*

S. No.	Role	No. of Respondent	Percentage
1.	Developer	15	46.90%
2.	Quality Analyst	7	21.90%
3.	System Analyst	5	15.60%
4.	Project Manager	3	9.40%
5.	Team Leader	2	6.30%

Total 64 respondents give their responses to our asked questions related to for OpenSSL and Vim Editor each and having working experience from 1 year to 8 years. Among them 16 workings on java programing language, 9 on C/C++, 3 Perl/Python and 4 on other scripting languages. We have provided the latest version of above both data set based on that we asked about their CRI and Avg_LoC to understand the working of the codebase. In the response, we got a range of CRI from 7% to 18% and for Avg_LoC 7 to 20. For Avg_LoC metrics, only one respondent touches 20 and rest from the range 7 to 18. We divide this response range into three classes of an equal interval, i.e. 7-10, 11-14 and 15-18. The range with the highest number of responses marked as Excellent, the second highest as moderate and with least number of responses as poor type. Table: 4 shows the complete range and for readability. In which 15-18 range have the highest response 11-14 as second highest while 7-10 as least. Similarly, for the average number of lines of code (Avg_LoC), we found the range 7-10 as excellent, 11-14 as moderate and 15-18 as poor readability type. Table: 5 show the complete information about it.

***TABLE 6** PROJECT READABILITY TYPE SCALE BASED ON CRI*

S. No.	Type of Projects	Range	Mean of Responses	No. of Respondent
1.	Excellent	15-18	15.51 %	26
2.	Moderate	11-14	12.49 %	24
3.	Poor	7-10	9.2 %	14

***TABLE 7** PROJECT READABILITY TYPE SCALE BASED ON AVG_LOC*

S. No.	Type of Projects	Range	Mean of Responses	No. of Respondent
1.	Excellent	7-10	8.50	45
2.	Moderate	11-14	11.82	14
3.	Poor	15-18	15.00	5

We have also calculated the mean for responses for each type of class to get a common point for readability. Our table 4 and table 5 both are based on open source data sets because software companies does not share their codebase. In the

past, when the concept was not popular, such types of research were conducted with collaboration with NASA, DRDO, etc. Due to this, we tried to bring the software development experts close to a common consensus for code readability and based on this consensus we proposed above scale.

Comparison of Calculated Results to Proposed Scale

The calculated value of CRI metrics and Avg_LoC metrics shown in figure 1, figure 2 and in figure 3, figure 4 respectively. It is clear from figure 1 that the readability of OpenSSL is 7 % for version 1 and decreases to version 3rd i.e. 5.60 % and then increases to 7.30 % for 4th version that comes under poor category by comparing with the scale referred in table 4. Similarly, the readability of Vim Editor is near about 8% that also comes under the poor category of readability. When we compare the calculated value of Avg_LoC of OpenSSL for 1st and 4th versions mention in figure 3 to the proposed scale we found it under the moderate category of readability while its version 2nd and 3rd come under the poor category of readability. Similarly, for Vim Text Editor, all four versions have same value referred to figure 4, and it lies in the poor category.

Recommendation for Future Release

For better readability of codebase its recommendation to keep the CRI and Avg_LoC in range of Excellent. The excellent range of CRI is 15-18 %. To reach this, we can use eq. 3 which determines the CRI require to add in the existing code base for the readable codebase. Similarly, for better readability, it is necessary to keep the Avg_LoC between 7-10, which means put the comment at an average of 9 lines of codes as a milestone. Eq 4 assists the number of lines after that a comment will require for better readability. For OpenSSL, from version 4 it is required to enhance 7 % in its current CRI to reach this it is necessary to add comment at every 9 lines of codes as milestone comment.

6 CONCLUSION

Code readability is a very critical factor in software development and its maintenance. It plays a great role in saving budget and resource saving during maintenance. Keeping the code readable from its initial development is good practice and surely helps the existing maintainers as well as for new maintainers of the project in understanding the working of the code. In order to measure the readability, we proposed two metrics namely CRI and Avg_LoC. From these metrics, we measure the enhanced CRI and incremental Avg_LoC for next release from current version or current release to its previous release. We also find out a scale of readability for both metrics and to found it we conducted a survey with development staff. Based on response gathered we suggest the range of scale and the type of codebase. We also empirically validate our proposed work with genuine arguments and hope that our work wills the software development personnel in developing human readable code and try to reduce maintenance efforts.

REFERENCES

[1] N. Khamis,R Witte,J. Rilling. "Automatic quality assessment of source code comments: the JavadocMiner." In International Conference on Application of Natural Language to Information Systems, pp. 68-79. Springer, Berlin, Heidelberg, 2010.

[2] E. Nurvitadhi, W.W. Leung, C. Cook, "Do class comments aid java program understanding?." In 33rd Annual Frontiers in Education, 2003. FIE 2003., vol. 1, pp. T3C-T3C. IEEE, 2003.

[3] Y Padioleau, L Tan, Y Zhou. "Listening to programmers taxonomies and characteristics of comments in operating system code." Proceedings of the 31st International Conference on Software Engineering. IEEE Computer Society, 2009.

[4] B Fluri, M Wursch, HC Gall, "Do code and comments co-evolve? on the relation between source code and comment changes." 14th Working Conference on Reverse Engineering (WCRE 2007). IEEE, 2007.

[5] JL Elshoff, M Marcotty, "Improving computer program readability to aid modification." Communications of the ACM 25. Vol 8, pp. 512-521, 1982.

[6] TM Shaft, I Vessey. "The relevance of application domain knowledge: The case of computer program comprehension." Information systems research 6.3, pp. 286-299, 1995.

[7] E Collar Jr, R Valerdi, Role of software readability on software development cost. 2006.

[8] S Rugaber, "The use of domain knowledge in program understanding." Annals of Software Engineering 9.1-2, pp. 143-192, 2000.

[9] RPL Buse, WR Weimer, "A metric for software readability." Proceedings of the 2008 international symposium on Software testing and analysis. ACM, 2008.

[10] T Tenny, "Program readability: Procedures versus comments." IEEE Transactions on Software Engineering 14.9, pp. 1271-1279, 1988.

[11] SN Woodfield, HE Dunsmore, VY Shen, "The effect of modularization and comments on program comprehension." In Proceedings of the 5th international conference on Software engineering, pp. 215-223. IEEE Press, 1981.

[12] SCB de Souza, N Anquetil, KM de Oliveira, "A study of the documentation essential to software maintenance." Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information. ACM, 2005.

[13] CS Hartzman, CF Austin, "Maintenance productivity: Observations based on an experience in a large system environment." Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering-Volume 1. IBM Press, 1993.

[14] BP Lientz, Issues in software maintenance. CALIFORNIA UNIV LOS ANGELES GRADUATE SCHOOL OF MANAGEMENT, 1983.

[15] T Siddiqui, A Ahmad. "Data mining tools and techniques for mining software repositories: A systematic review." Big Data Analytics. Springer, Singapore, pp. 717-726, 2018.

[16] A Ahmad, T Siddiqui, NA 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.

[17] 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.

[18] GitHub, https://github.com/ retrieved, 26 September,2019.
[19] T. Siddiqui and A. Ahmad, "A Conceptual Framework to Mining Software Repositories for Software Metrics." International Journal of Innovative Technology and Exploring Engineering (IJITEE) Vol. 8(10), pp. 4173-4177. 2019.
[20] OpenSSL, https://en.wikipedia.org/wiki/OpenSSL, retrieved, April 10, 2019.
[21] Vim text_editor, https://en.wikipedia.org/wiki/Vim_(text_editor), retrieved, September 10, 2019.
[22] Stack Overflow Developer Survey 2015 § IV. Text Editor". Stack Overflow. Retrieved July 25, 2016.
[23] Vim documentation: Uganda, (Report), 2019.
[24] Openssl, https://github.com/openssl/openssl/, retrieved, September 15, 2019.
[25] Vim, https://github.com/vim/vim/, retrieved, September 20, 2019.
[26] R.F. Flesch, “A New Readability Yardstick,” J. Applied Psychology, vol. 32, pp. 221-233, 1948.
[27] R. Gunning, The Technique of Clear Writing. McGraw-Hill, 1952.
[28] G.H. McLaughlin, “Smog Grading—A New Readability,” J. Reading, vol. 12, no. 8, pp. 639-646, May 1969.
[29] J.P. Kinciad and E.A. Smith, “Derivation and Validation of the Automated Readability Index for Use with Technical Materials,” Human Factors, vol. 12, pp. 457-464, 1970.
[30] 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.
[31] UA Mannan, I Ahmed, A Sarma, "Towards understanding code readability and its impact on design quality." Proceedings of the 4th ACM SIGSOFT International Workshop on NLP for Software Engineering. ACM, 2018.
[32] FP Brooks, NS Bullet. "Essence and accidents of software engineering." IEEE computer, vol. 20, no. 4, pp. 10–19, 1987.
[33] S Becker, F Plasil, RH Reussner. “Quality of Software Architectures Models and Architectures” 4th International Conference on the Quality of Software Architectures, QoSA 2008, Karlsruhe, Proceedings. Vol. 5281. Springer, 2008.