The Study of Code Reviews based on Software Maintainability in Open Source Projects

Aziz Nanthaamornphong and Thanyarat Kitpanich
Department of Information and Communications Technology, Faculty of Technology and Environment, Prince of Songkla University, Phuket Campus, Phuket, Thailand.
aziz.n@phuket.psu.ac.th

***Abstract*—Recently, open source software (OSS) applications have been widely adopting. However, the OSS projects have problems in the software quality, such as security and maintainability. Generally, software engineers focus on the software maintainability because this quality attribute can reduce the cost and increase the productivity of software development. To better understand how the OSS developers improve the source code based on a software maintenance perspective; this research aims to investigate how the developers are interested in the maintainability under the peer code review of the OSS projects. We analyzed whether the code authors changed their code based on the code review's comments related to maintenance issues by examining two OSS projects. We found that the OSS developer community tends to pay more attention to software maintainability. Finally, we expect that this research will increase the empirical evidence about the quality of OSS projects, particularly maintainability.**

***Index Terms*—Code Review; Open Source Software; Software Engineering; Software Maintenance.**

I. Introduction

Currently, commercial organizations and government agencies have been adopting the open source software (OSS) to their works widely because the OSS is developed by software engineers who are the experts and have various experiences for software system development. So, the OSS is quite popular and reliable regarding functionality's corrections, and it helps to reduce the cost of software investment. However, a previous research [1] found that the main problem of OSS development is the lacks of the systematic process or procedure and formal documents related to the system development such as requirements, designs, testing and so on. As all of above causes, some OSS projects have poor code quality [2], especially the software security and software maintainability.

Software maintainability is one of the key success factors of software development because most developers have to spend time around 40%-50% of the software development life cycle to find defects and errors during a software development process or after product delivery [3]. Also, they must pay the maintenance cost for 40%-80% (average 60%) [4]. Many OSS projects specify the ways to improve the development process and to solve the problems of software quality [5]. For example, the changed code must be approved through a process of analyzing code written (review) by a teammate or reviewers who do not develop the source code by themselves. For this procedure, it is called "Peer code review" or "Modern code review."

Peer code review is the key part of a software development process because this method is widely accepted for the software engineers in software quality assurance practices [6]. Also, the important thing of the peer code review is the comments taken from the reviewers. These comments can point to bugs or defects in the source code, suggest better alternatives of solving problems to make the developers improve the software quality, help developers submit a higher quality changed code, and improve the author's development skills, including standardizing the source code in order to help everyone be able to read and understand how the system works. However, some comments may contain incorrect information and provide comments that are not related to software quality improvement.

According to the existing literature related code reviews in OSS projects, we found that it has currently no research studying whether the OSS developers pay attention to software maintainability under the peer code review in the OSS project. To investigate how the OSS developers are interested in the software's maintainability, we analyzed the comments given by code reviewers with these following objectives: 1) to study the relationship between code review comments related to maintainability and the source code improvement based on the obtained comments and 2) to examine the comments related to the five sub-characteristics of software maintainability (modularity reusability analyzability modifiability and testability), which were addressed by code authors. In this research, we analyzed the review comments from two OSS projects, including Eclipse (https://eclipse.org/) and Qt (https://www.qt.io/).

We expect that the results of this research can provide the empirical evidence about the software quality in the OSS projects to the software engineering research community. Additionally, the results will be the guidance for software developers to realize the importance of software maintainability before modifying the source code.

The remainder of this paper is organized as follows. Section II provides background concepts related to this work. Section III presents related work. Section IV describes the research methodology. Section V shows the study results, Section VI draws conclusions and describes plans for the future work.

II. Background

This section describes the background regarding the code review in OSS projects and software maintainability.

A. Peer code review in the OSS project

The OSS is the software, which allows users or developers access to the code repository to modify or improve the source code [7]. As of this reason, the software engineering community in the OSS projects uses the peer code review to increase the software reliably. Many OSS projects have adopted code review tools, e.g., Gerrit [8], ReviewBoard [9] as the media of communication and knowledge exchange about software development. The key objective of peer code review techniques is to ensure that the changed code can decrease bugs or defects and has no effect on maintenance in the long term.

In this research, we analyzed the comments from the code review tool, called Gerrit that is integrated with Git. Gerrit provides the services for the code review procedure along with the storage of data related to reviews' comments of many OSS projects. Figure 1 shows the Gerrit code review process, including the following steps. First, the developer (code author) sends a code review request to the Gerrit system. Next, the reviewer(s) reviews the code and provides comments. Then, the code author who sent the request reads the given comment(s). Lastly, the code author modifies the code, but all comments may not be modified. Note: this process repeats until the changes have been approved.

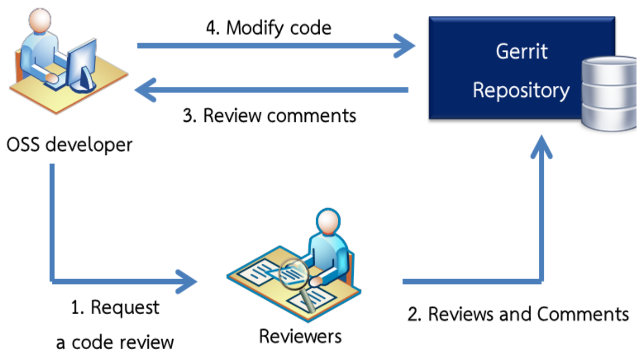

Figure 1: The overview of peer code review process in Gerrit

B. *Software Maintainability*

In this research, we focus on the maintainability, which is one of key software quality attributes to increase the software quality and reduce the expense of maintenance. ISO/IEC 25010 provides the definition of software maintainability as "the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in the environment, and in requirements" [10]. We mentioned ISO/IEC 25010 because this standard is the global standard of software products paid attention by the global business organization that concentrates on the systems and software quality requirements and the evaluation. Additionally, software maintainability has sub-characteristics, which impact on maintenance directly or indirectly.

ISO/IEC 25010 also defines the sub-characteristics of maintainability in five characteristics as follows:

i. Modularity – "Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components."
ii. Reusability – "Degree to which an asset can be used in more than one system, or in building other assets."
iii. Analyzability – "Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified."
iv. Modifiability – "Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality."
v. Testability – "Degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met."

We use all above five sub-characteristics as the initial keyword set in this research to find the comments related to software maintenance.

III. Related Works

This section describes the previous studies that related to our work.

Because several OSS projects have easily accessible resources, so many researchers studied the OSS projects in various aspects. Rigby et al. [11-12] examined code review practices in OSS development, e.g., study of practices in the Apache project [13]. Baysal et al. [14] investigated the factors affecting on rejections of program bug fixes (patch) in the WebKit project. Tao et al. [15] presented the guidance to help developers to solve defects of code to be accepted by code reviewers. They investigated patch-rejection in Eclipse and Mozilla.

Bosu et al. [16] analyzed 1.5 million review comments from five Microsoft projects that were taken by the code review tool, called CodeFlow. The researchers classified the useful comments to help the developers to be able to modify the source code according to the given comments. In Jacek et al. [17] work, they analyzed the comments through CodeFlow. The results indicated that least 50% of all comments related to the long-term code maintainability.

Moreover, we found that most of the existing studies focused on the software quality and maintenance by finding the defects during software development, or investigated code quality by examining project management regarding faults/bugs reports [18]. Several studies reported the causes of poor code, one of them is code smell. Code smell is code in the software that may cause flaws or degrade code quality [19], which may have the direct impact on software maintainability [20-21].

IV. Research Methodology

This research aims to analyze the comments made during the code review process by using a text mining technique. The main procedure consists of two parts as follows.

A. *Mining Code Review Repositories*

In this study, we have examined the accessible data repository related to the comments of OSS projects. We chose to study the Gerrit code review repository with embedded Qt and Eclipse data. The main reason for selecting these OSS projects because both projects are still under heavy development recently. They are also the projects that have gained attention and popularity in the study field of software engineering research [15, 22-24].

Figure 2 shows the process for mining code review repositories, which include the following steps: First, we reviewed the existing literature related to the Gerrit and code review (e.g., a Gerrit database structure, a review process in OSS projects). Next, we explored the OSS projects maintained in Gerrit for selecting OSS projects that allow us to retrieve comments data from reviews. Finally, we collected code review data from the OSS projects. The obtained datasets were stored in MySQL.

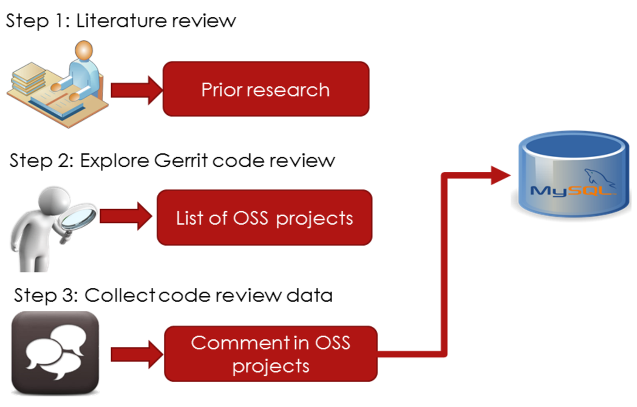
Step 1: Literature review
Prior research
Step 2: Explore Gerrit code review
List of OSS projects
Step 3: Collect code review data
Comment in OSS projects
MySQL

Figure 2: The review's comment mining process

To query and manage comments data retrieved from the OSS projects, we developed a JAVA application that can pull the data maintained in the Gerrit system. The queries were performed from the Qt and Eclipse comments made during 2012 to 2016 via REST API provided by Gerrit and then the pulled data was stored in the local database (MySQL), which Gerrit returns the results as JavaScript Object Notation (JSON). Figure 3 illustrates an overview of the data retrieving process.

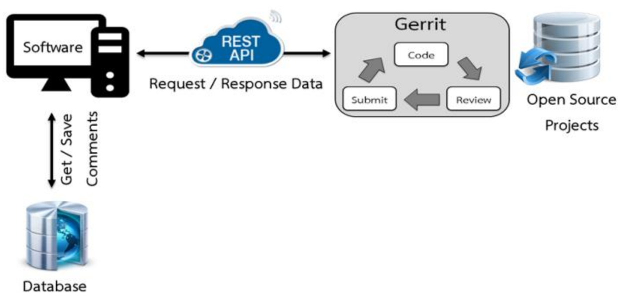

Figure 3: The overview of data retrieving process

Based on the data retrieving process, we maintained following data in our database:

i. Id – the unique number for each data entry
ii. Patch_number – the number of patch
iii. Created_on – the review request's date
iv. Uploader – the uploader id (Gerrit's user name)
v. Author – the code author
vi. Reviewer – the reviewer id (Gerrit's user name)
vii. File – the review requested file name
viii. Line – the number of changed code
ix. Message – the comment message
x. Kind – the status of changed code, such as 'Trivial Rebase', 'No Code Change' or 'Rework'

Here, we show some comments that we pulled from the Eclipse project.

"Another possibility is to get isValidThread to call isCurrentThread(), then the isValidThread can be updated to another implementation if desired without code duplication."

"Please add a similar test to IndexCPPBindingResolutionTest where the first two lines are in the header file and the third line is in the source file."

B. Analyzing a dataset

Once we obtained the datasets of reviews, we analyzed the data by using a text mining technique shown in Figure 4. The detail of each step will be described as follows:

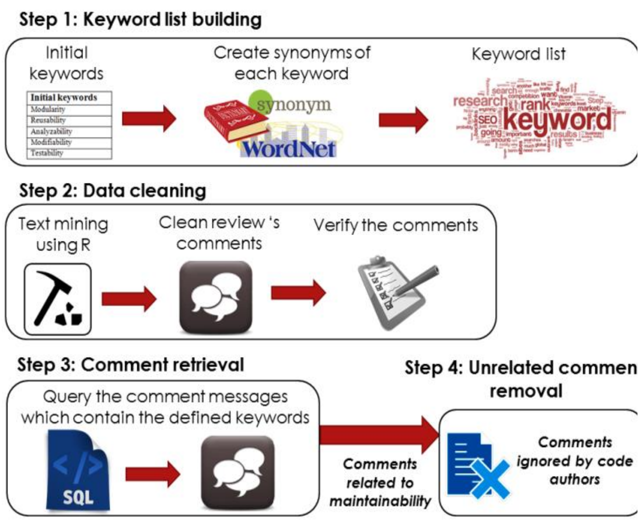
Step 1: Keyword list building
Initial keywords
Initial keywords
Modularity
Reusability
Analyzability
Modifiability
Testability
Create synonyms of each keyword
synonym
WordNet
Keyword list
keyword
Step 2: Data cleaning
Text mining using R
Clean review 's comments
Verify the comments
Step 3: Comment retrieval
Query the comment messages which contain the defined keywords
SQL
Step 4: Unrelated comment removal
Comments related to maintainability
Comments ignored by code authors

Figure 4: The data analysis process

1) Keyword list building

We built a set of keywords from ISO/IEC 25010 with a 'maintainability' characteristic, which consists of following sub-characteristics: modularity, reusability, analyzability, modifiability, and testability. Initially, we performed the analysis on these five keywords since these keywords were defined by the global standard. Typically, the words in a sentence from the reviewer's comments might not match with our specified keywords. Therefore, the words are classified into a group of synonyms with each keyword by matching these English words from Word-net software [25] and English dictionaries databases (e.g., "change," "adjust," and "alter" are synonyms of "modify").

2) Data cleaning

This step involves converting all words to lowercase and removing unnecessary messages such as stop-word, whitespace, numbers and programming-language special character, and splitting multi-word (e.g., the comment identifier "bindingResolution" would be divided into "binding" and "Resolution"). We used R software, which is a statistical program embedded with a text mining (tm) package to clean the data and transform words into the common root (Stemming) to reduce the processing time for keyword searching. To ensure that the cleaned data can be used for the next step, we inspected the validity of review's comments throughout the cleaned data.

3) Comment retrieval

In this step, we developed Structured Query Language (SQL) scripts to query comment messages, which contain the defined keywords. Then, the queried comments were stored in the database. The examples of SQL commands are shown as follows.

*SELECT * FROM comment_detail_eclipse WHERE massage LIKE "%modify%" OR massage LIKE "% correct %" OR massage LIKE "% alter %" OR massage LIKE "% adjust %" OR massage LIKE "% qualify%"*

*SELECT * FROM comment_detail_eclipse WHERE massage LIKE "% analyze %" OR massage LIKE "% diagnose %" OR massage LIKE "% delineate %" OR massage LIKE "% anatomize %"*

4) Unrelated comment removal

We removed the comments that the code authors did not change the code having maintenance based comments obtained from the reviewer. To remove these comments, we analyzed the responses of code authors and the status of changed code.

V. RESULTS AND DISCUSSION

Based on our research objectives (described in Section I), we report the results of the analysis for each objective as follows.

A. The relationship between code review comments related to maintainability and the source code improvement based on the obtained comments

We have analyzed the comments from two OSS projects from 2012–2016. The Qt and Eclipse projects had a total of 309,396 and 115,896 comments respectively. In this research, we investigated the comments related to five types of maintainability. The result from keyword queries showed the total number of comments related to the maintainability from Qt and Eclipse projects of 29,840 and 27,527 respectively (8.9% and 25.75% in Figure 5).

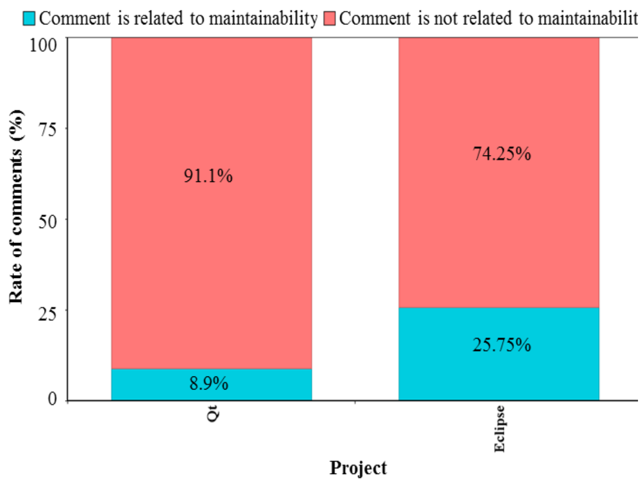

Figure 5: Rate of comments are related to maintainability and comments are not related to maintainability

Figure 6 shows the ratio of comments that the source code has been changed based on the maintenance reason each year. The analysis in Figure 6 suggests that the number of source code changes related to five maintenance sub-characteristics in the span of five years (2012–2016) period of Eclipse and Qt projects are 21.15% and 10.73% on average. In these two projects, the number of comments that the source code has been changed based on the maintenance reason is considered moderately low compared to all the maintenance comments from the reviewers.

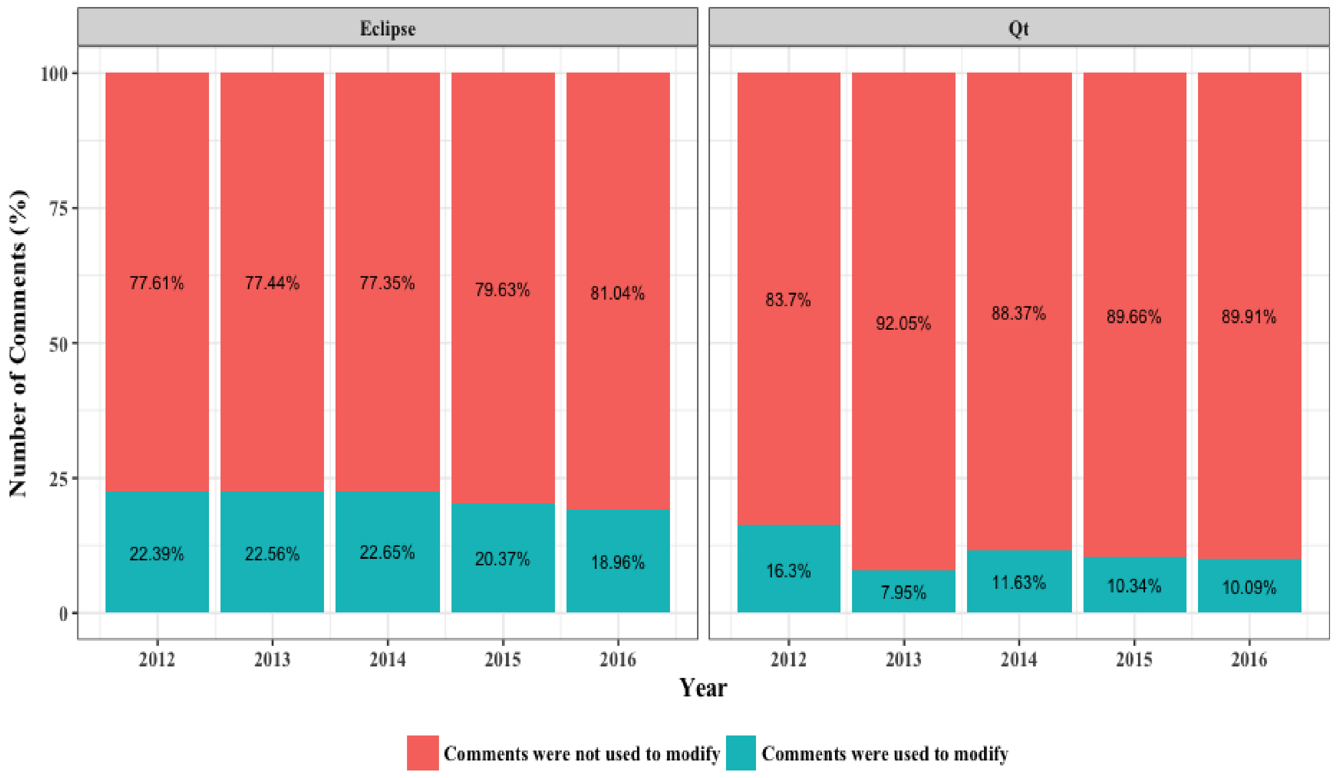

Figure 6: The use of comments to modify code

Although the number of changed code from the maintenance reason is low, the study of OSS development trends shows that the developers in the OSS community have paid more attention to the maintenance. From the observation, the number of changed code from the maintenance comments have been increasing since 2012 and continuing to grow up in the future as seen in Figure 7. Following this trend, we believe that more software change or modification can increase the complexity and risk for errors and defects in the software system. In addition, long-term system tests and software maintenance can be affected. From the observation, the modifiability and testability graphs of two projects show that the developers tend to change more source code.

Additionally, we analyzed these results using a correlation analysis to find the relationship between the data year period and the number of comments on the code change based on the given maintenance reason from two projects. The Pearson's Correlation Coefficient of Eclipse and Qt projects are 0.895 and 0.887 respectively, which illustrate that the relationship between the data year period and the number of comments on the code change, based on the maintenance reason, are strong and going in the same direction. This implies that when the number of years increased, the number of comments related to the maintenance of the software also increased.

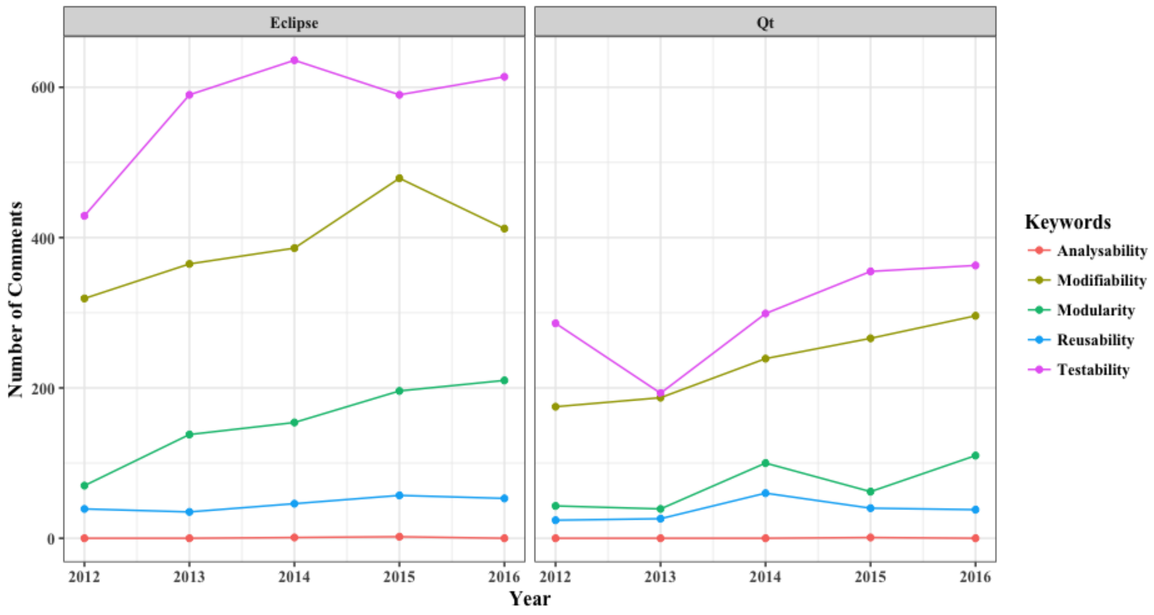

Figure 7: Comments on Sub-characteristics

We also examined the time that the reviewers reviewed the code during the course of their workdays. Figure 8 and 9 show boxplots that describe the distribution of the number of comments related to maintainability, per day of the week, over the span of five years (2012-2016) of Eclipse and Qt projects, respectively. It is not surprising that the number of comments given on business days (Monday, Tuesday, Wednesday, Thursday, and Friday) was greater than the comments given on the weekend. This evidence may imply that the reviewers in OSS projects work on business days in the commercial software development team manner.

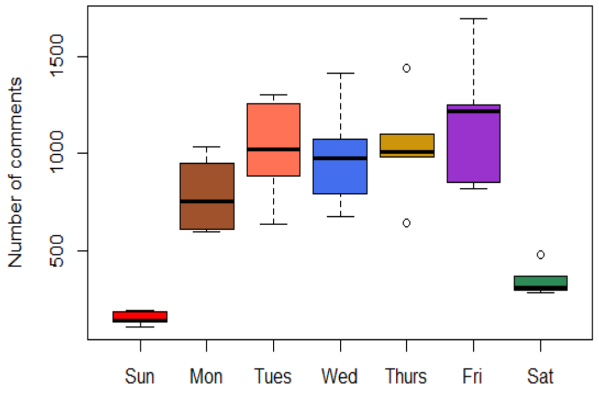

Figure 8: Number of comments related to the maintainability per day of the week in the span of 5 years (2012-2016) of Eclipse

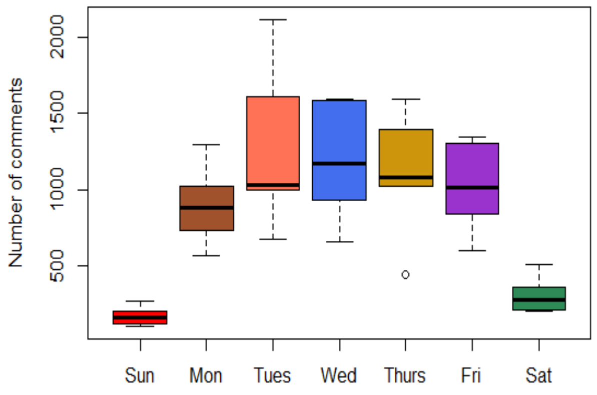

Figure 9: Number of comments related to the maintainability per day of the week in the span of 5 years (2012-2016) of Qt

With a t-test analysis, we found that there is no statistical difference among business days, but there is a statistically significant difference between the comments on Saturday and Sunday. Thus, this evidence may indicate that the reviewers frequently reviewed the code during the workday instead of doing it on the weekend.

B. The comments related sub-characteristics of software maintainability which were addressed by code authors

We found that the code authors paid special attention to the testability related comments with the highest percentages of addressed comments in both Eclipse (49.1%) and Qt (46.7%) projects (shown in Figure 10). As a result, in our opinion, the code authors might place emphasis on the testability since it can test persistence and quality in the system, such as whether the execution of a program can execute three billion instructions per second. The popular secondary sub-characteristic that the code authors have paid attention to is 'modifiability' in both two projects. The possible explanation here is that the code authors are likely to improve software quality or mitigate defects. Table 1 shows the number of comments related to each sub-characteristic.

Table 1
Addressed comments related to sub-characteristics

Sub-characteristics	Number of comments	
	Eclipse	Qt
Testability	2,859	1,496
Modifiability	1,961	1,163
Modularity	768	354
Reusability	230	188
Analyzability	3	1

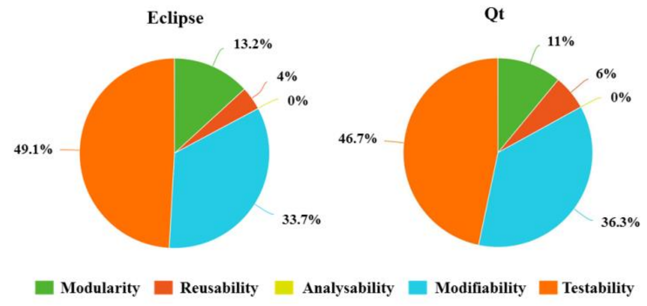

Figure 10: The percentages of addressed comments related to sub-characteristics

Based on the findings, we identify the limitations of this study as follows. First, as we studied only two OSS projects, the results of this study may not be generalized to all OSS projects because each OSS project may have a different review process and diverse experiences of developers. However, we believe that the results of this study provide a useful idea for other similar studies on the quality of OSS projects. Second, we only utilized the R program to process data. To use other text mining applications may return different results. Finally, the data selection process might introduce bias problems because each author manually read half the total number of comments. We attempted to reduce bias by reviewing excluded comments together several times till we ensured that those comments were neither related to maintainability nor useful.

VI. Conclusions and Future Works

This research aims to analyze the comments in the Eclipse and Qt projects under the code review process. The analytical results suggested that the number of changed based on the maintenance reason is pretty small when we compared to all of the comments. However, the developers in the OSS community tend to improve the quality of source code more in our study. This trend could be observed from the increasing number of changed code based on the comments related to software maintenance each year.

We suggest that the OSS developers should focus on software maintenance to prevent the impact of code modifications during software development and facilitate future maintenance, which can reduce time and costs in the software development process.

In the future, we plan to increase the number of keywords from the existing initial keyword set. The main five sub-characteristics of maintainability can be used to find new additional keywords by applied a Latent Dirichlet Allocation (LDA) technique, which is an algorithm for discovering the hidden topics in a large document of texts. Building a set of keywords is more evidence to analyze the comments related to maintenance.

We hope that this research can be a path to find additional features related to the software maintenance capabilities found in existing OSS projects. In addition, the future research can increase the empirical evidence to cover the definition of "maintainability," including the study of other OSS projects with a longer period of research time in order to receive more accurate code reviewers' trends and comments. However, the information presented in this paper is sufficient to guide the development and improvement of the quality in OSS.

References

[1] V. Tiwari, and R.K. Pandey, "Open source software and reliability metrics," *The International Journal of Advanced Research in Computer and Communication Engineering*, vol. 1, no. 10, pp. 808-815, Dec. 2012.

[2] I. Stamelos, L. Angelis, A. Oikonomou and G. L. Bleris, "Code quality analysis in open source software development," *Information Systems Journal*, vol. 12, no. 1, pp. 43-60, Jan. 2002.

[3] G. S. Walia and J. C. Carver, "Using error information to improve software quality," in *Proceedings of the IEEE 24th International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2013, pp. 107.

[4] R. L. Glass, "Frequently forgotten fundamental facts about software engineering," *IEEE Software*, vol. 18, no. 3, pp. 112-111, May 2001.

[5] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozill," *Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 309–346, Jul. 2002.

[6] R. Baker, "Code reviews enhance software quality," in *Proceedings of the ACM/IEEE 19th International Conference on Software Engineering (ICSE)*, 1997, pp. 570–571.

[7] "What is free software?" Available at https://www.gnu.org/philosophy/free-sw.html. [Accessed: 12-Jun-2017].

[8] "Gerrit." Available at http://code.google.com/p/gerrit/. [Accessed: 12-Jun-2017].

[9] "Review board." Available at https://www.reviewboard.org/. [Accessed: 12-Jun-2017].

[10] ISO/IEC 25010, *Systems and Software Quality Requirements and Evaluation (SQuaRE)*. ISO/IEC 25010, ed. IEC, 2011.

[11] P. Rigby, B. Cleary, F. Painchaud, M. Storey, and D. German, "Open source peer review–lessons and recommendations for closed source," *IEEE Software*, pp. 56-61, Nov. 2012.

[12] P. C. Rigby and M.-A. Storey, "Understanding broadcast based peer review on open source software projects," in *Proceedings of the ACM/IEEE 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 541–550.

[13] P. C. Rigby, D. M. German, and M.-A. Storey, "Open source software peer review practices: A case study of the apache server," in *Proceedings of the ACM/IEEE 30th International Conference on Software Engineering (ICSE)*, 2008, pp. 541–550.

[14] O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey, "The influence of non-technical factors on code review," in *Proceeding of the IEEE 20th Working Conference on Reverse Engineering (WCRE)*, 2013, pp. 122-131.

[15] Y. Tao, D. Han, and S. Kim, "Writing acceptable patches: an empirical study of open source project patches," in *Proceeding of the IEEE 30th International Conference on Software Maintenance and Evolution (ICSME)*, 2014, pp. 271-280.

[16] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: an empirical study at Microsoft," in *Proceedings of the ACM/IEEE 12th Working Conference on Mining Software Repositoriesd (MSR)*, 2015, pp. 146-156.

[17] J. Czerwonka, M. Greiler and J.Tilford, “Code reviews do not find bugs : how the current code review best practice slows us down,” in *Proceedings of the ACM/IEEE 37th International Conference on Software Engineering (ICSE)*, 2015, pp. 27-28.

[18] R. Rana and M. Staron, “When do software issues and bugs get reported in large open source software project?,” in *Proceedings of the 25th International Conference on Software Measurement (IWSM)*, 2015, pp. 1-14.

[19] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[20] B. C. Wagey, B. Hendradjaya and M. S. Mardiyanto, “A proposal of software maintainability model using code smell measurement,” in *Proceedings of the 2nd International Conference on Data and Software Engineering (ICoDSE)*, 2015, pp. 25-30.

[21] A. Yamashita and S. Counsell, “Code smells as system-level indicators of maintainability: An empirical study,” *Journal of Systems and Software*, vol. 86, no. 10, pp.2639–2653, Oct. 2013.

[22] M. B. Zanjani, H. Kagdi and C. Bird. “Automatically recommending peer reviewers in modern code review,” *IEEE Transactions on Software Engineering*, vol. 42, no. 6, pp. 530-543, June 2016.

[23] K. Hamasaki, R. G. Kula, N. Yoshida, A. E. C. Cruz, K. Fujiwara, and H. Iida, “Who does what during a code review?: datasets of OSS peer review repositories,” in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*, 2013, pp. 49–52.

[24] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K. Matsumoto, “Who should review my code? A file location-based code-reviewer recommendation approach for modern code review,” in *Proceeding of the 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2015, pp. 141-150.

[25] G. A. Miller, “Wordnet: A lexical database for English,” *Communications of the ACM*, vol. 38, no. 11, pp. 39-41, Nov. 1995.