

The Impact of Peer Code Review on Software Maintainability in Open-Source Software: A Case Study

Aziz Nanthaamornphong¹, Thanyarat Kitpanich²
College of Computing, Prince of Songkla University, Phuket, Thailand

Abstract—Recently, open-source software (OSS) has become a considerably popular and reliable source of functionality corrections. OSS also allows software developers to reduce technical debt in software development. However, previous studies have shown that the main problem within OSS development is the lack of systematic processes and formal documents related to system development, such as requirements, designs, and testing. This feature of OSS development causes problems in the software quality, such as those related to security and maintainability. In this research, the authors focused on the software's maintainability because this attribute has the greatest potential to reduce the cost and increase the productivity of the software development process. There is currently no existing research that examines whether OSS developers pay attention to software maintainability. To better understand how OSS developers improve software maintainability, this research aims to answer the question: "Are developers interested in software maintainability under the modern code review of open-source software projects?" To answer the research question, the authors investigated the code review process in which the OSS developers changed the code based on a review of code comments related to maintenance and collected the sub-characteristics associated with software maintainability from the existing literature. The authors examined the review comments from two OSS projects: Eclipse and Qt. The results suggest that the number of code revisions due to maintenance issues was moderate and that the OSS developers tend to improve source code quality. This direction could be observed from the increasing number of modifications on given maintenance-based comments over the years. Therefore, an implication of this is the possibility that OSS project developers are interested in software maintainability.

Keywords—Open-source software; software maintainability; code review

I. INTRODUCTION

Open-source software (OSS) has received considerable attention from users worldwide, especially among organizations and commercial industries that widely utilize OSS. Notably, OSS is often a key to organizational success for entrepreneurs. OSS can create the opportunity for organizational growth, enhance performance, increase the options for software use, and decrease operating costs. Moreover, users from around the world can use OSS freely without fees, allowing software developers to access the OSS code repository [1] and jointly develop software [2]. OSS projects are being continuously developed and improved by software developers with expertise and experience in system development. Therefore, OSS has become popular and credible in terms of effectiveness and functionality.

A previous study [3] noted that OSS development lacks systematic processes or procedures and formal documents related to system development, such as requirements, design, testing, etc. Considering the aforementioned problems, most software developers working on OSS projects solve relevant problems when they are reported by users or software developers on the same team. One frequent problem is the occurrence of a defect or bug in the system. When the source code is frequently modified, the size of the code increases, and it generally becomes more complex. Moreover, some source codes can be difficult to understand. These issues can lead to serious problems, such as increases in time and development costs. Additionally, software developers and commercial entrepreneurs who want to develop software often cannot comprehend old codes that are complex and poorly described. Thus, accessibility may be poor, and call functions may fail. In addition, poor results or inaccurate information may not meet user expectations, and possible vulnerabilities in the system may lead to the pirating of information or viruses. All of these issues can cause software project development to ultimately fail. The quality of the source code of OSS projects could be improved [4], in particular, the relevant software security and maintainability of these projects [5].

Software maintainability can improve product quality at a low cost and ensure that software development meets the intended objective. Therefore, software maintainability is one of the most important steps in the software development life cycle because most software developers spend approximately 40-50% of their time identifying defects or errors during the development process or after product delivery [6]. Notably, maintainability can solve OSS quality issues and prevent problems that may affect the system in the future. The code review process is an important part of the software development process that ensures the creation of high-quality software and the implementation of successful OSS projects [7]. Thus, many OSS projects use "peer code review" or "modern code review" [8] as a guideline for developing and improving code.

Peer code review is commonly used in software engineering for quality control [9]. Generally, a code change must be reviewed by the software developers on the same development team or reviewers other than the code editor. Generally, the code review process does not have a fixed format, but it prevents problems associated with patch files and helps avoid bugs or errors that can influence the long-term applicability of the software. Moreover, the comments from reviewers can be used to identify bugs or defects in the source code, improve the source code, modify the source code to meet certain standards,

and improve code readability [10]. However, some comments may not be directly related to software quality improvements.

A review of studies on software development in OSS projects indicated that the comments provided via peer code review have not been extensively investigated. Developers should review comments related to software maintainability in OSS projects. These comments are often collected in the Gerrit system, which provides services related to the code review process and facilitates communication and the exchange of ideas about software development among software developers. In this study, two OSS projects: Eclipse (<https://eclipse.org/>) and Qt (<https://www.qt.io/>), are investigated. These projects were selected because they have different purposes and have already been completed. Therefore, extensive comment information is available for both projects. The projects provide the opportunity for researchers in the field of software engineering to access this information and study code reviews. The main research questions posed in this study are as follows:

- 1) How many sub-characteristics are related to software maintainability?
- 2) Are the developers of OSS projects interested in software maintainability under the peer code review?
- 3) What characteristics of software maintainability have the developers considered?

In order to answer these questions, text mining was applied to identify the sub-characteristics related to software maintainability that appear in comments provided by peer code review, and the latent Dirichlet allocation algorithm (LDA) was applied to develop new characteristics related to software maintainability. Quantitative data analysis was performed to identify the trends of developers in software maintainability for OSS projects. In addition to answering the main research questions, the results were used to determine the factors that affect the number of comments related to maintainability, and statistical analysis was conducted to answer the following research questions:

- 1) Does the size of the reviewer pool influence the comments related to maintainability?
- 2) How many comments related to maintainability are given by code reviewers each day?

The results of this study will provide empirical evidence for the research community regarding the software quality of OSS projects. Moreover, this study will shed light on the importance of software maintainability and help software developers understand sub-characteristics related to software maintainability, which usually appear in OSS projects.

The remainder of this paper is organized as follows. Section II provides the necessary background and related work. Section III describes the research methodology. Section IV shows the results of this study. Section V discusses the results. Section VI concludes the paper and outlines our future work.

II. RELATED WORK

In this section, the theories and methodologies used to conduct the research are summarized, including the literature review, which provides the basic framework of this study.

A. Peer Code Review for OSS Projects

Code review is a way to reduce the risk of error and improve the quality of software by checking the source code to identify defects or bugs that may occur during software development. Moreover, such a review can standardize the source code [11]. When organizations or projects that require software development use this code review process, it ensures that the developed code will meet the specifications or needs of the user and that the impacts of source code modifications are minimized. Therefore, code review has been a best practice in software engineering for over 35 years [12]. “Peer code review,” or “modern code review,” is applied by open-source organizations and communities to improve the quality of source code [13]. The source code is standardized by deleting duplicate functions and removing irrelevant or unnecessary code. Moreover, the code review process allows detailed comments from experienced reviewers that can improve the source code. Additionally, peer code review facilitates the sharing of knowledge about system development [11] because code review services often utilize specific communication tools. The most common tools are CodeFlow, Gerrit, Collaborator, Crucible, Review Board, and Upsource.

B. Software Maintainability

ISO/IEC 25010 is the model used by organizations around the world because it has set the software quality standard for code review based on system and software requirements. Furthermore, ISO/IEC 25010 was selected in this study because it includes both quality evaluation for a system or software and the corresponding effects on stakeholders or users. The international standards that are used to assess quality include requirement definition and quality measurement and evaluation [14].

ISO/IEC 25010 defines 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.” In a previous study on software maintainability, Ghosh et al. [15] identified the 40 sub-characteristics that impact maintainability. These sub-characteristics are used as keywords in this study to obtain comments related to software maintainability.

C. Literature Review

Bakar and Arsat [16] studied the factors that affect the quality of OSS by measuring the quality of the source code using McCall’s Quality Factor model. This model considers maintainability, accuracy, reliability, efficiency, and ease of use. The results indicated the relationships among lines of code and the complexity of the source code based on correlation analysis. The most influential factors were correctness, maintainability, efficiency, and usability. Related research on complexity measurements for source code generally uses Cyclomatic Complexity (CC) as an indicator, although there are newer indicators as well. For example, Walden et al. [17] analyzed code and used complexity metrics to predict the possible density gaps in 14 web applications. The study results could be used by software developers in OSS projects to create a simple system structure and clearly illustrate the development of software, especially security, because the number of gaps

was decreased. Norick et al. [18] studied OSS projects developed in C/C++ to determine whether the number of revisions affected the quality of the source code. This analysis used the CC of each coded function and the density of comments as representatives of source code quality. However, the study did not find a correlation between the number of software developers and the software quality, and the average software quality was satisfactory for all the studied OSS projects. Schmidt and Porter [19] studied the software development process of the “Skoll” OSS project and of closed-source projects. They found that the community of OSS developers could control long-term maintainability procedures and development costs, and these characteristics created confidence and a level of acceptance in the quality of the software. Specifically, users had confidence in the software and corresponding system. Such practices are easier to implement in OSS projects compared to closed-source projects because the disclosure of sources allows software developers who are interested in OSS projects to voluntarily participate in quality improvement tasks.

Successful OSS projects usually participate in peer code reviews to assess the quality of the software [20]. The code review process begins with the software developers creating patch files, which add value to a project. Rigby [21] stated that using supportive tools for peer code review could affect the progress of OSS, and there was a subsequent push for software development that facilitates code review. As noted by Bosu and Carver [7], review tools have been widely used to study comments via platforms such as ReviewBoard. The researchers analyzed the number of participants in the project, the number of requests for code review, and the response time after a request for code review was submitted by comparing MusicBrainz Server and Asterisk. However, Bacchelli and Bird [22] studied motivation and challenge in searching for defects in Microsoft projects using the CodeFlow tool, which records data from code reviews. The comments aiming to guide code modification during the peer code review process were investigated. The results showed that reviewers with expertise in system development and reading and reviewing source code provided the most useful comments for developers.

Baysal et al. [10] studied the factors that affect the modification of patch file defects by extracting code review information for the WebKit project from Bugzilla. Most reviewers denied large patch files because developers modified or developed source code that was too complex, included unnecessary components, and/or affected other functions or modules. Tao et al. [23] proposed a method for developers to code patch files that were accepted by reviewers. They verified the causes of rejected patch files by collecting rejection information for Eclipse and Mozilla patch files. The information was collected and stored based on an online survey of 246 software developers. Considering the issues that caused a code fix to be accepted or rejected, the community can determine how to optimize the peer code review process.

fBosu et al. [10] searched for ways to improve the efficiency of peer code review, which was useful for system development. They analyzed 1.5 million comments from Microsoft projects to identify the characteristics of useful and useless comments. Some comments that software developers received contained incorrect information or comments unrelated to system development, and software developers took time to

respond to these questions and discuss why they did not modify the source code accordingly. Moreover, the study identified the factors that were correlated to beneficial comments, such as review experience, organizational experience, and being on the same team as the code developers. Czerwonka and Greiler [24] found that up to 50% of the comments provided via the peer code review process were related to software maintainability and that very large patch files decreased the benefit of reviewer comments and increased the review time by six hours to a week. Nonetheless, they found that reviewer comments are beneficial and reflect the overall view of software development. Such a review system promotes cooperation in problem resolution and expedites the development and improvement of software. Such systems provide a standardized framework for researchers to better understand peer code review.

Maintainability is a very important factor because software maintenance accounts for 66% of the life cycle and 40-80% of the cost of development [25]. Code smells are among the most important problems that must be urgently solved. A code smell is any characteristic in a source code, such as a poorly designed structure or poor organization, that can lead to larger issues. These factors decrease the quality of software maintainability. Wagey et al. [26] presented a model of software maintainability and applied it to six OSS projects. The results showed that the model improved the structures of the OSS projects. Refactoring [27], a process that involves shortening lines of code or combining similar functions to reduce the complexity of the code [28], has also been used to improve software functionality. Rizvi and Khan [29] used regression analysis to develop a class-level maintainability estimation model called MEMOOD, which facilitated development and reduced the time required to improve or edit code before delivery. This model used the features of understandability and modifiability, which were calculated based on object-oriented metrics, to estimate the maintainability of the code considering its size and structural complexity. Khan and Khan [30] developed a model that was used to measure the quality of analyzability. Multiple linear regression was conducted for the structural design of a system in which complexity might affect the quality of software maintainability, especially its analyzability. However, they found that other factors, such as the size of a file or program, may also make maintainability difficult.

III. RESEARCH METHODOLOGY

This study analyzes the comments related to software maintainability from the peer code review of OSS projects using text mining techniques. The main procedure consists of two parts, as described in the subsequent subsections.

A. Development of Software for Searching and Storing Data from Gerrit

Software was developed for searching and storing data associated with Eclipse and Qt comments that were collected by Gerrit from 2017 to 2021. Before storing the data from Gerrit, tables were designed in the database to support the data used in this study. The stored information is as follows:

- ID - the unique number for each data entry;
- Patch_numbers - the number of patches;

- Created_on - the review request date;
- Uploader - the uploader's ID (Gerrit user name);
- Author - the code author;
- Reviewers - the reviewer's ID (Gerrit user name);
- File - the name of the file with the review request;
- Line - the number of changed lines of code;
- Message - the comment message; and
- Kind - the status of the changed code, such as trivial rebase, no code change or rework.

B. Analyzing a Dataset

In this step, the sub-characteristics related to software maintainability that appear in comments provided via the peer code review of OSS projects were searched using a text-mining technique. This technique also verifies that the comments were related to maintainability. This step is shown in Fig. 1, and each step is shown as follows.

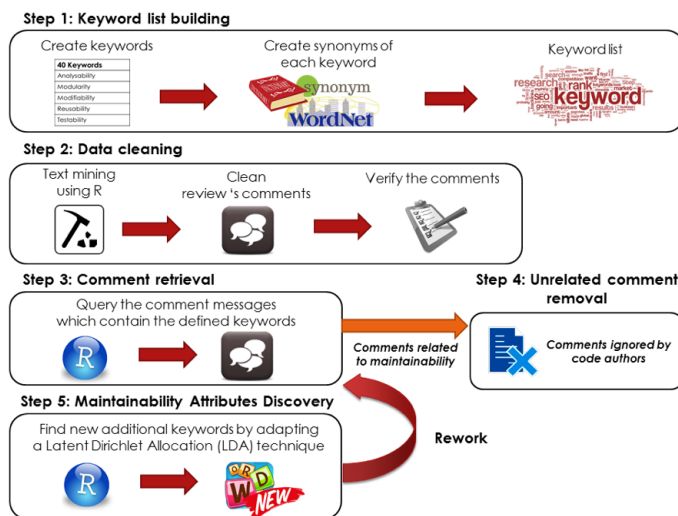


Fig. 1. The data analysis process

Keyword List Building - A set of keywords that are related to software maintainability, such as modularity, analyzability, and testability, was constructed. These keywords were selected based on the sub-characteristics that are related to the definition of “software maintainability”. Previously, Ghosh et al. [31] used these collected keywords to search for comments related to software maintainability. However, they found that the reviewers of OSS projects typically did not use words in comments that matched these keywords. Therefore, the sub-characteristics related to software maintainability are converted to their root forms to reflect the basic meaning of the term; for example, “modifiability” was shortened to the root “modify”. Then, synonyms for each root word, such as “edit”, “improve”, “solve”, and “amend” (for “modify”), were also added to the search set. The words with the same meaning as these keywords were identified using WordNet, a website that provides synonyms for various terms.

Data Cleaning - In this study, R was used for data cleaning. Specifically, the text mining (tm) package was used in this step. In data cleaning, the stop word function was employed to remove terms that were not significant without changing the meaning of the sentence. For example, pronouns, conjunctions, and prepositions usually appear in each comment but could generally be removed. Moreover, whitespace, numbers, and special characters, such as special symbols (, , * , #, and ! were deleted. Then, word stemming was performed by removing prefixes and suffixes. Words with the same root, such as “compatible” and “compatibility,” have similar meanings; therefore, “ible” and “ibility” could be removed to obtain the root “compat”. Another example is “connection”, “unconnected”, “connective” and “connecting”, for which the root is “connect”. Moreover, letters were converted from upper case to lower case to reduce the number of required indices and the processing time.

Comment Retrieval - In this step, a command set was developed to search for comments with keywords or meanings related to software maintainability. At least one word from the constructed keyword list must be matched, and the search results were converted into a file format for easy analysis. In order to search for comments related to software maintainability, we developed a command in R that connects to a MySQL database and searches based on various subcommands. For example, a set of commands is developed to find the word “modify,” and words with the same meaning as “modify” appear in the text of comments in the database.

In order to search for comments containing keywords, we created a command set for each keyword for the convenience of searching and storing. After the comments related to software maintainability were obtained, each comment was reviewed to improve credibility. This check was manually performed to verify that each comment contained a keyword or related term. If the text in a comment did not match a keyword or related word, we deleted the text from the database. After verification, the results were stored in table format, and relations were preserved. The table shows the number of terms, either keywords or related words, that appear in comments and are related to software maintainability.

Unrelated Comment Removal - We analyzed the results from step 3 and determined the number of comments in Eclipse and Qt related to software maintainability. These results could help answer the question of whether the developers are interested in software maintainability in OSS projects based on peer code review. To determine whether a comment was considered in code modifications, a command set was established in R by pairing comments related to software maintainability and comments from Gerrit. Notably, most comments from Gerrit included both the comments from reviewers and the comments of software developers who responded. This code modification assessment considers the type of code modification, such as trivial rebase or rework.

Additionally, the words in the reply related to code modifications, such as “done”, “finished”, or “edited” (as shown in Fig. 2), were considered. After checking the comments based on this procedure, the comments with no replies were removed from the database.

Maintainability Attribute Discovery - Thirty-three sub-



History – Comments in Gerrit		Status
 "Please add a similar test to performTest() where the first two lines are in the header file and the third line is in the source file."		Trivial Rebase
 "Done."		Trivial Rebase

Fig. 2. Example of comment related to maintainability and responses of code authors

characteristics of software maintainability were selected, and other sub-characteristics of software maintainability hidden in comments related to Eclipse and Qt projects were searched using the LDA. This algorithm could be applied to search for topics related to maintainability. The result of the LDA in R presented the frequency of occurrence of words that were related to the keywords, and we were required to determine whether these words should be new keywords. If the answer was yes, the term was added to the keyword set, and steps 3 and 4 were repeated until there were no additional keywords.

IV. RESULTS

Based on our research questions (described in Section I), the results of the analysis are presented.

A. How Many Subs-Characteristics are Related to Software Maintainability?

In Ghosh [31], 40 sub-characteristics related to maintainability were identified and studied. The following criteria were used to select sub-characteristics: 1) the definition of the sub-characteristic must be relevant to the definition of “software maintainability” according to ISO/IEC 25010, and 2) a sub-characteristic keyword cannot be a common or general word. The sub-characteristics of this study that did not meet those criteria were as follows:

- 1) Compliance,
- 2) Conciseness,
- 3) Delivery,
- 4) Documentation,
- 5) Impact Analysis,
- 6) Programming Language; and
- 7) Self-descriptiveness.

The obtained sub-characteristics that met these criteria were converted into verbs or nouns by removing the suffix, and the final keywords were easy to search for in the comments of reviewers on OSS projects. The results of keyword determination are shown in Table I.

We then reviewed the keyword search results to identify potential words that should not be considered keyword synonyms. The criteria for synonym consideration are as follows:

- 1) Cannot be a common term used in the programming language, such as class, function, feature, method, object, or parameter, or a general computer-related

TABLE I. THE LIST OF KEYWORDS

No.	sub-characteristics	Keyword	No.	sub-characteristics	Keyword
1	Accuracy	Accuracy	18	Implementation	Implement
2	Adaptability	Adapt	19	Instrumentation	Instrument
3	Analyzability	Analyze	20	Integrability	Integrate
4	Augmentability	Augment	21	Localization	Localization
5	Availability	Available	22	Modifiability	Modify
6	Changeability	Change	23	Modularity	Modular
7	Completeness	Complete	24	Perfectiveness	Perfect
8	Complexity	Complex	25	Portability	Portable
9	Comprehensibility	Comprehension	26	Readability	Read
10	Consistency	Consistency	27	Reusability	Reuse
11	Correctability	Correc	28	Simplicity	Simple
12	Durability	Durable	29	Stability	Stable
13	Efficiency	Efficient	30	Standardization	Standard
14	Effort	Effort	31	Testability	Test
15	Expandability	Expand	32	Traceability	Trace
16	Extensibility	Extension	33	Understandability	Understand
17	Flexibility	Flexible			

term, such as system, computer, software, file, or command.

- 2) For duplicate synonyms, only synonyms from three root words are selected.
- 3) For different synonyms, all words from three roots are used as keywords.

The search process for the keywords and synonyms that met these criteria is shown in Table II. The software developers that use Eclipse and Qt can consider all characteristics of software maintainability to improve and edit their source codes. Therefore, in addition to the 33 characteristics identified by the LDA, the frequent terms in comments related to software maintainability were also searched. The LDA results in Table III contain keywords and synonyms such as change, adjust, test, and improve. Therefore, to choose the new maintainability characteristics from the LDA, we considered the most frequently used words in each group.

Considering the maintainability characteristic produced by the LDA, some groups did not contain appropriate words. However, Eclipse and Qt contained two new characteristics related to software maintainability: cohesion and duplicate. To define the new maintainability characteristics, the authors used comments provided by the reviewers to search and identify these two maintainability characteristics. The words that typically appear with “cohesion” and “duplicate” had a relationship value of 0.6. Thus, those keywords were associated with software maintainability at least 60% of the time. The words most highly correlated with the new types of maintainability characteristics in comments are considered to define the maintainability characteristics and group the synonyms (as shown in Table IV).

- 1) The cohesion module identifies the modules that work together by connecting the components of the modules. These components may work at the same time or have the same input but different operations.
- 2) The duplicate function combines functions with similar properties or procedures to reduce the duplication of functionality, which can potentially cause confusion in calling.

TABLE II. THE LIST OF KEYWORD AND SYNONYMS

No.	Keyword	Synonym
1	Accuracy	accuracy, exact, truth, certainty, precision, propriety, rectitude, validity, sure, definite, inevitable, rigorous, evident, categorical, explicit, just, lawful
2	Adapt	adapt, adjust, modulate, alter, fine, shape, regulate
3	Analyze	analyze, analysis, diagnose, assay, delineate, muse, anatomize
4	Augment	augment, amplify, spread, inflate, escalate, dilate, enhance, accumulate, increase, suffuse, raise, mount, aggrandize, splay, accrete
5	Available	available, accessible, handiness, obtainable, satisfactory, convenient, usable, benefit
6	Change	change, vary, remodel, permute, convert, transform, re-vamp, purge, reform
7	Complete	complete, full, absolute, plenary, finish, utter, flawless
8	Complex	complex, complicate, sophisticated, elaborate, manifold, labyrinthine, multiple, confuse, entangle, mix, muddle, discursive, tangle, intricate, bewildered, imbroglia, intricacy, jumble, obscurity
9	Comprehension	comprehension, finality, inference, conclusion, notion, realization, savvy
10	Consistency	consistency, coherence, pertinacity, adhesion, invariability, tenacity
11	Correct	correct, rectify, fit, favorable, appropriate, worthy, deserve, suitable, due, rightful, infallible, redress, regularize
12	Durable	durable, lasting, hardy, imperishable, substantial, permanent, long, immune, indissoluble, enduring, strong
13	Efficient	efficient, able, capable, competent, proficient
14	Effort	effort, endeavor, fighting, might, stamina, energy, strength, activity
15	Expand	expand, add, accretion, accrue, develop, flatten, grow, prosper, thrive
16	Extension	extension, flare, connect, continue, broaden, enlarge, widen, magnify, prolong, elongate, protract, proliferate
17	Flexible	flexible, elastic, dexterous, limber, resilient, springy, pliable, stretch
18	Implement	implement, execute, accomplish, achieve, resource, utilize, apply
19	Instrument	instrument, equipment, accessory, apparatus, machinery, appliance, device, gadget
20	Integrate	integrate, gather, collect, compile, assemble, embody, coordinate, cooperate, harmonize, consolidate, sticking, combination, joining, amassing, hoard, compound
21	Localization	localization, limitation, narrowing, restriction, stint, definition, circumscription
22	Modify	modify, customize, edit, improve, solve, repair, amend, qualify
23	Modular	modular, configuration, component, constituent, ingredient, composition, complement, element, procedure, segment
24	Perfect	perfect, excellent, ideal, immense, keen, superb, wonderful, terrific, fantastic, splendid, magnificent, superior, entirety, consummate, faultless
25	Portable	portable, mild, soft, slight, lightweight, feathery, weak, mushy, flimsy
26	Read	read, peruse, pronounce, extrapolate, imagine, speculate, surmise, see, view, interpret, transliterate, look
27	Reuse	reuse, recycle, reiterate, rehash, reclaim, exploit, revise
28	Simple	simple, ease, expedient, facile, easy
29	Stable	stable, still, steady, invariable, constancy, endurance, firmness, fastness, indissolubility, sturdily
30	Standard	standard, formula, archetypal, representative, typical, characteristic, degree, tier, quality, criterion, measurement, imperative, touchstone, property
31	Test	test, verify, check, prove, evaluate, examine, assess, attempt, experiment, inspect, trial, proof, tryout, investigate
32	Trace	trace, pursue, tag, trail, detect, search, seek, probe, meaning, significance, hint, consequence, point, follow, behave, track, extract, transcribe
33	Understand	understand, explain, fathom, grasp, knowledge, perceive

TABLE III. THE LDA RESULTS

Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
change	use	add	work	method
convert	sure	adjust	example	tool
parameter	suggest	file	make	class
patch	implement	test	code	review
index	cohesion	improve	model	package
name	resource	worth	duplicate	commit

TABLE IV. NEW KEYWORDS

No.	Keyword	Synonyms
1	Cohesion	cohesion, sticky, tough, gummy, leathery, tenacious
2	Duplicate	duplicate, repetitive, double, copy, transcript, counterpart, facsimile, reproduce, mimeograph, repeat, replicate

B. Are the Developers of OSS Projects Interested in Software Maintainability Under the Peer Code Review?

The Qt and Eclipse projects had totals of 309,165 and 108,357 comments, respectively. Thirty-five keywords were identified based on the characteristics that were relevant to maintainability. These keywords included 33 previously used terms and two terms obtained from the LDA. A script was developed in program R for text mining. The results indicated that 39,638 and 93,629 comments were related to maintainability in the Qt and Eclipse projects, respectively, based on the keywords (36.58% and 30.28% in Fig. 3). In both projects, the number of comments related to software maintainability was low compared to the total number of comments.

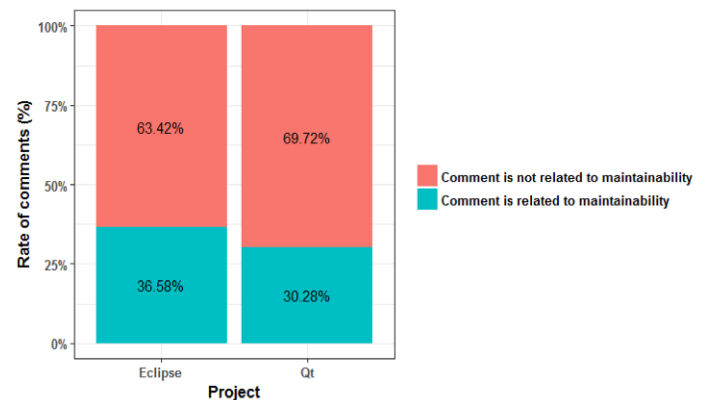


Fig. 3. Rate of comments are related to maintainability and comments are not related to maintainability

The keywords were used 64,616 and 149,610 times in the comments of the Eclipse and Qt projects, respectively. In order to determine the extent to which software developers of Eclipse and Qt prioritized software maintainability, the comments related to software maintainability and code editing were analyzed. Overall, 36,975 and 94,408 comments led to source code modifications for Eclipse and Qt, representing 57.22% and 63.10% of all comments, respectively. Fig. 4 shows that the comments provided to the software developers of Eclipse and Qt resulted in source code edits 50-60% of the time.

We found that software developers of Eclipse and Qt edited

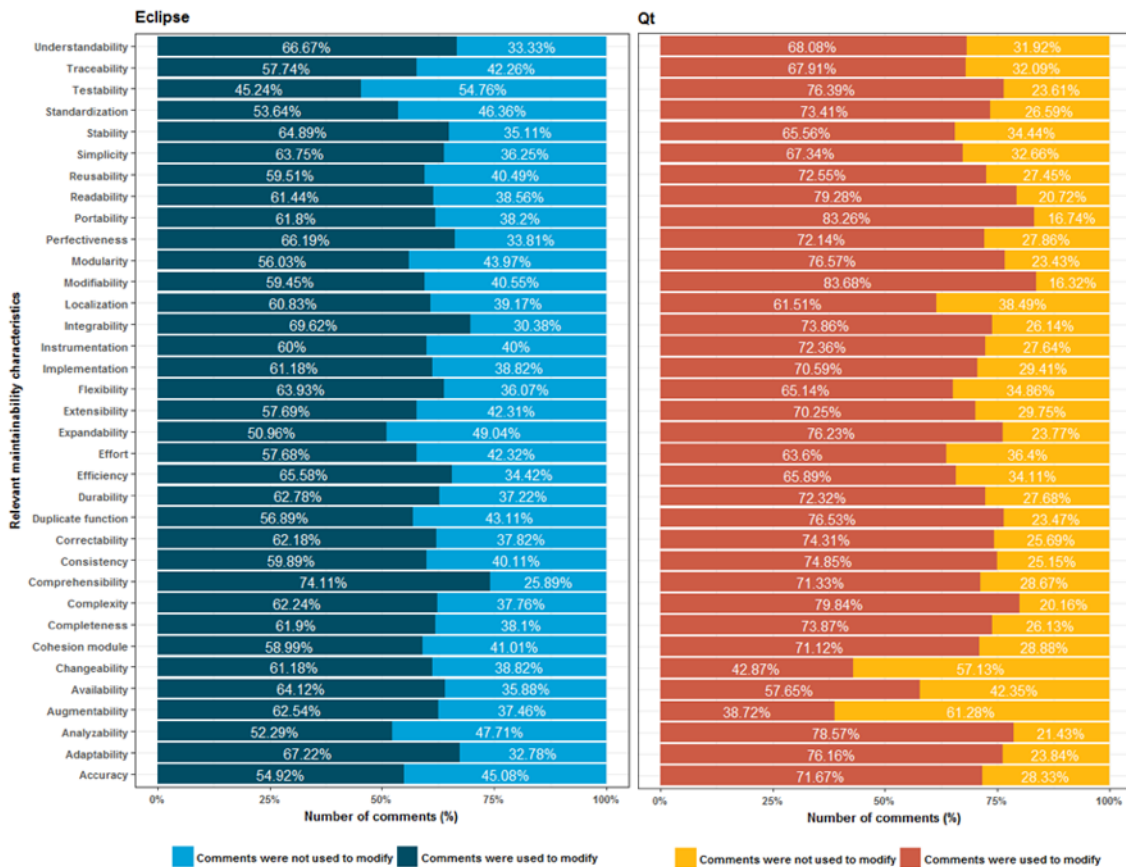


Fig. 4. Ratio of comments to modification code

the source code according to the comments related to software maintainability approximately 50-60% of the time. This result provides empirical evidence suggesting that the software developers of Eclipse and Qt prioritized software maintainability in more than 50% of source code modifications.

The Pearson correlation coefficient was determined in this study at a 95% confidence interval to obtain the relationship between comments related to maintainability and comments related to code changes in these two OSS projects. Based on an analysis in R, the Pearson correlation coefficients of Eclipse and Qt were 0.995 and 0.993, respectively. These coefficients indicate that the number of comments related to software maintainability and the number of comments related to code modifications were highly related. In other words, when the reviewers of Eclipse and Qt provided more software maintainability comments, the software developers made more source code edits.

C. What Characteristics of Software Maintainability have the Developers Considered?

The five most common comments related to software maintainability were investigated, and the number of code modifications related to those comments was determined. Fig. 5 illustrates that the software developers of OSS projects make the most source code edits based on comments that discuss readability, and the rates of related changes are

61.44% for Eclipse and 72.14% for Qt.

D. Does the Size of the Reviewer Pool Influence the Comments Related to Maintainability?

In the two OSS projects, most of the comments that led to code edits were made by reviewers and other software developers. Therefore, the number of comments was compared to the number of comments related to software maintainability each month. Moreover, the sizes of the groups or teams of software developers were considered to determine how many software developers reviewed the code.

R was used to calculate correlation coefficients and obtain the relationship between the number of software developers who provided comments related to software maintainability and the number of comments related to software maintainability. The annual Pearson correlation coefficients in the study period were 0.78, 0.88, 0.72, 0.53, and 0.75 for Eclipse and 0.85, 0.86, 0.57, 0.005, and 0.17 for Qt (as shown in Fig. 6 and 7).

Fig. 6 shows that the group size of reviewers and the number of comments related to software maintainability were highly related in the Eclipse project. Fig. 7 illustrates that the group size of reviewers and the numbers of comments related to the project maintainability in Qt were highly correlated.

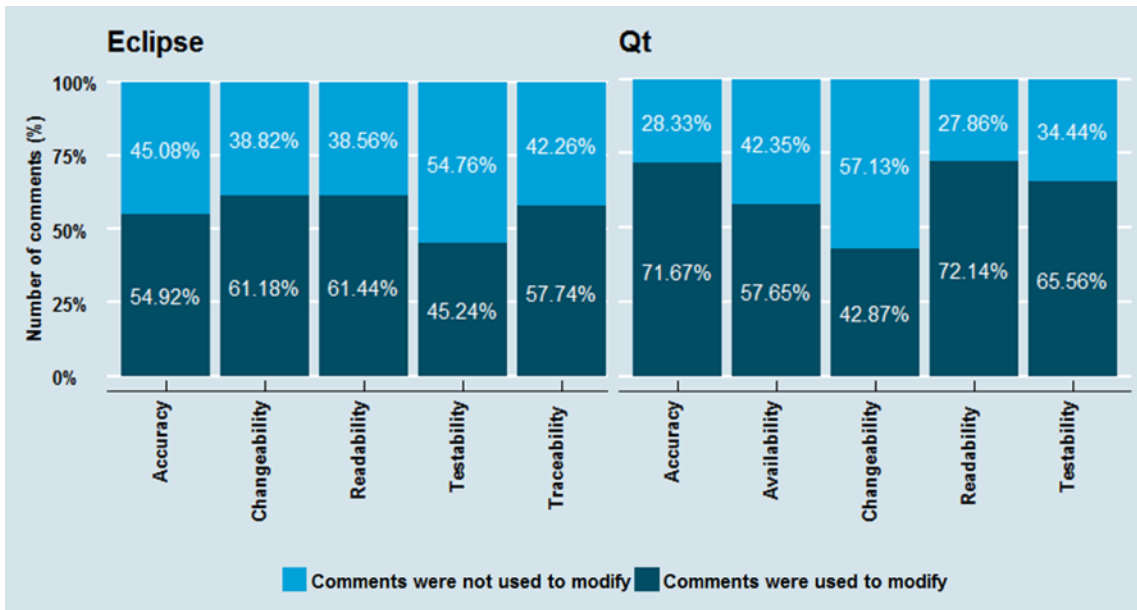


Fig. 5. The percentage of addressed comments related to sub-characteristics

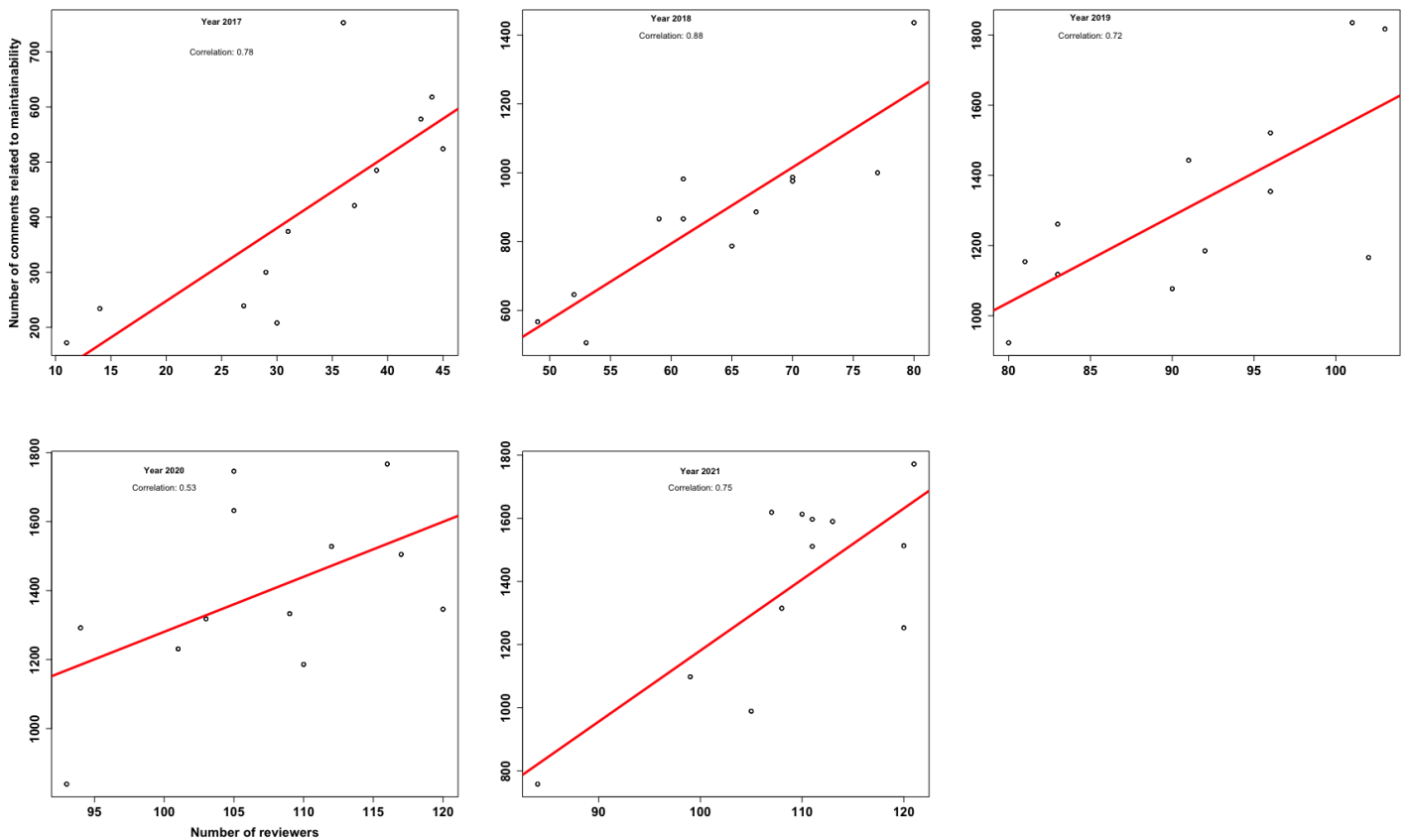


Fig. 6. The relationship between the number of reviewers and comments related to maintainability of the Eclipse project

The data were further analyzed to determine the trend in the number of reviewers. Fig. 8 shows that the number of reviewers who provide comments related to maintainability has increased

annually. As observed in the Eclipse and Qt projects, the new generation of software developers is interested in developing high-quality software and providing comments that improve

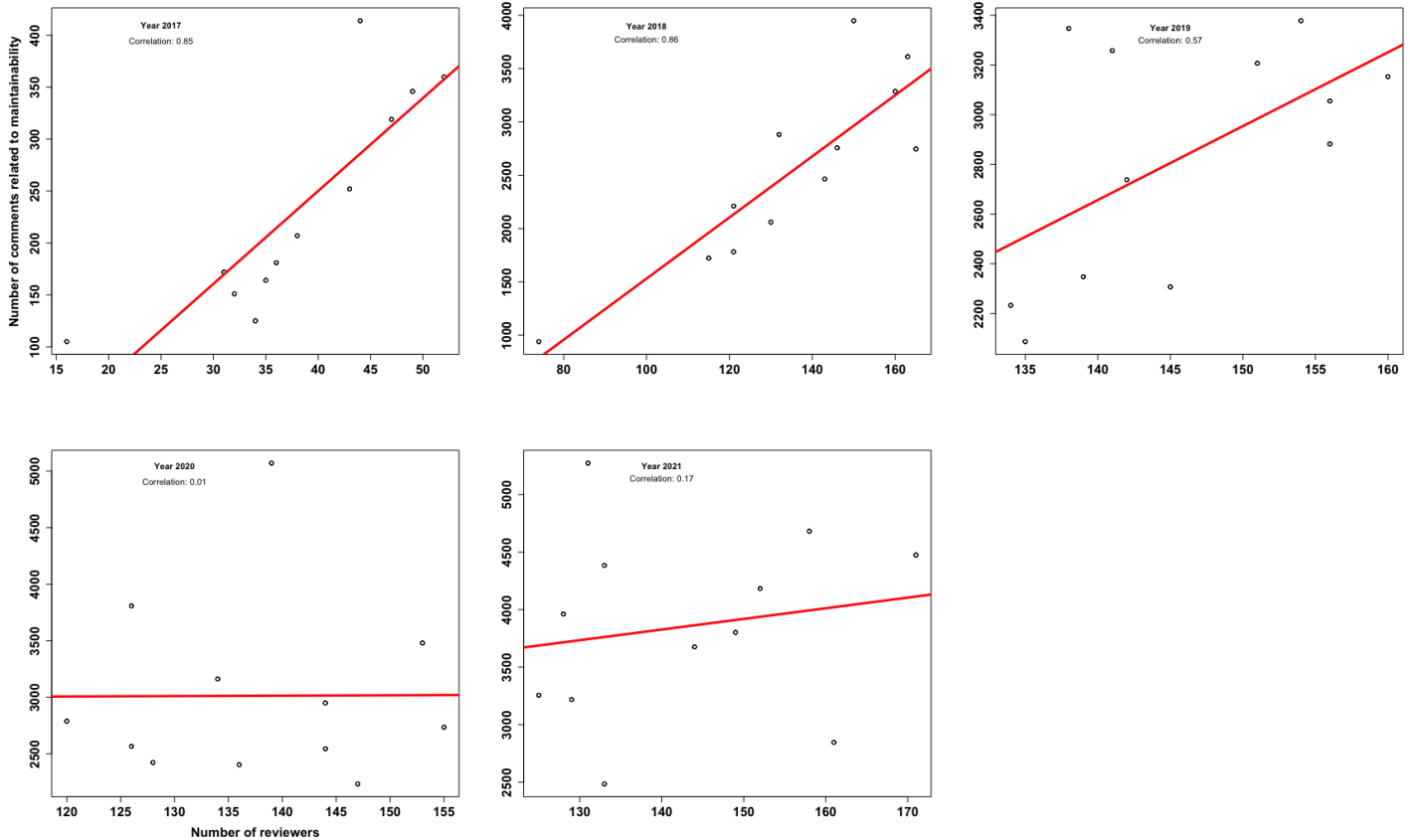


Fig. 7. The relationship between the number of reviewers and comments related to maintainability of the Qt project

the quality of OSS projects.

E. How Many Comments Related to the Maintainability are Given by Code Reviewers each Day?

The working times of reviewers for each project were evaluated based on ANOVA and posthoc comparisons considering Tukey's honest significant difference (HSD), which compares the average of multiple data pairs. Tukey's HSD was used because hypothesis testing with ANOVA considers multiple averages. Therefore, the results do not indicate which data pair is different. Moreover, Tukey's HSD is a popular statistical test in software engineering research. In this study, the results of the posthoc Tukey's HSD indicate how many comments were made by reviewers on each day of the week on average.

R was used to perform ANOVA at a 95% confidence level for the following hypotheses.

H0: The average number of comments related to software maintainability was the same on each day of the week.

H1: The average number of comments related to software maintainability differed on each day of the week.

The results indicate that Eclipse has a p-value of 0.000649 and Qt has a p-value of 0.0105. The p-values of both projects are lower than 0.05, so H0 is rejected. Therefore, it can be concluded that the average number of comments related to

maintainability provided by reviewers each day varies. To assess the differences, Tukey's HSD is calculated.

R was used to perform the variance for Tukey's HSD calculation at a confidence level of 95%, and the hypotheses of the test are as follows.

H0: The average number of comments related to software maintainability is the same each day.

H1: The average number of comments related to software maintainability is different each day.

Based on the variance of Tukey's HSD, the average number of comments from Monday to Saturday does not vary for Eclipse. While there is a significant difference between the comments related to the maintainability on Sunday compared to those made throughout the rest of the week, the difference is not significant. Moreover, the values on Saturday and Sunday are similar. This finding may indicate that the Eclipse code reviewers usually review code and provide comments related to software maintainability from Monday to Saturday rather than on Sunday. Based on the variance of Tukey's HSD for Qt, there are no significant differences between the number of relevant comments on weekdays and weekends. Therefore, it can be concluded that the reviewers of Qt usually review code and provide comments related to software maintainability every day.

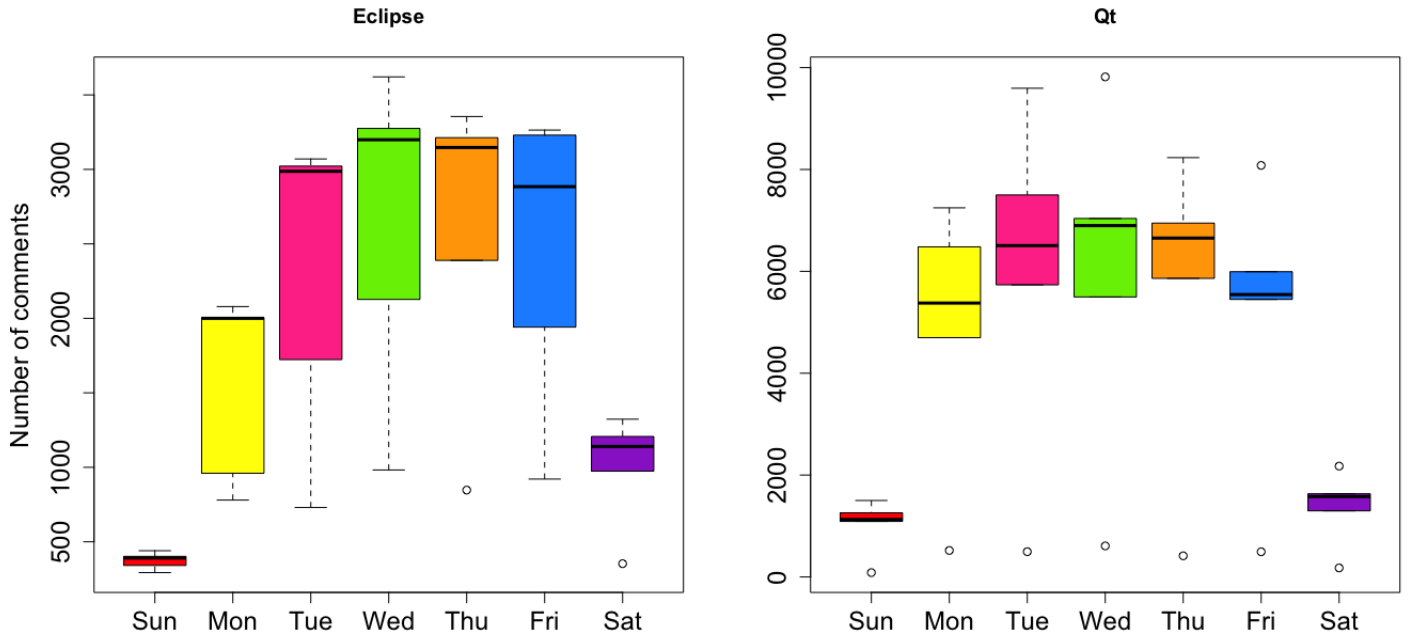


Fig. 8. Boxplot of comments related to the maintainability per day of the week

V. DISCUSSION AND THREATS TO VALIDITY

This section discusses the results and threats to the validity of this study.

A. Discussion

The comments related to software maintainability from the reviewers of OSS projects account for approximately 30% of all comments, and some comments are related to the development of the main functions and subfunctions of the system and to other characteristics, such as stability [32]. In addition to maintainability, stability is a very important characteristic because users around the world want to be sure that OSS can maintain their confidentiality and integrity and prevent accessibility issues or allow software editing without permission. Moreover, published software must be made available. Therefore, software developers who focus on stability mainly try to find possible gaps in code or vulnerable code changes. The results of this study indicate that most comments led to modifications of the code, which is a good sign for software quality. Therefore, it is predicted that the software developer communities of the Eclipse and Qt OSS projects will continue to focus on maintainability in the future.

The results of this study and the applied methodology are of interest in assessing software maintainability. This study differs from other studies in that software engineering researchers have typically investigated Eclipse and Qt based on other maintainability topics and methods, such as using indicators to verify source code and check maintainability. For example, Yamashita et al. [33] checked for files that were vulnerable to size increases and analyzed the complexity of the software. Additionally, Caglayan et al. [34] used indicators to predict

defects in OSS projects, and Counsell et al. [35] calculated the maintainability index based on coupled factors. Therefore, the results of this study related to “software maintainability” may differ from the results of other studies based on various factors, such as the methods, processes, tools, and environments of OSS projects.

B. Threats to Validity

Threats to validity can reduce the accuracy and reliability of studies and lead to inaccurate results. This section presents the threats to the validity of the current study. The threats in this study can be categorized into three classes as follows.

Construct Validity - This research compiled OSS comments from the code review system called “Gerrit”. The storage structure of Gerrit was studied to design and develop searching and storage software. The collected data were sets of text or characters, and some data, such as symbols and figures, may be incomplete. However, the collected data were considered acceptable for analyzing the quality of OSS projects, especially maintainability, which was the objective of the research. In the research process, R was used for text mining, data processing, and data analysis, as well as to search for comments related to software maintainability. Moreover, R was used to conduct various statistical methods, such as variance analysis, correlation analysis, and regression analysis. Therefore, other programs or tools were used for text mining, and the results may be different.

Internal Validity - This study presents the comments related to software maintainability from the code review processes of Eclipse and Qt. In the comment search process, keywords and synonyms related to each sub-characteristic of

software maintainability were selected to provide the support framework for searching. Only comments with one or more of the selected keywords or synonyms were analyzed, and irrelevant comments or those that did not meet specific criteria were removed. Result verification was performed by manually reviewing all comments related to software maintainability before processing. However, this manual may have an inherent bias, and we attempted to minimize this bias by repeating the verification process many times. Moreover, expert code reviewers were asked to verify the results by reading the comments (approximately 30% of all comments). Currently, there is no tool available for the automatic review of comments.

External Validity - The results of this study only provide empirical evidence for the two OSS projects considered. Therefore, the findings are not applicable to every OSS project because each project differs in internal structure, and the code review and software development processes also vary. However, the results of this study can provide preliminary guidelines for other studies related to the quality of similar OSS projects.

VI. CONCLUSION

The objective of this research was to analyze the comments associated with the Eclipse and Qt projects as part of the code review process. The analytical results suggest that the number of changes based on maintainability is moderate. This result can improve the awareness of software developers regarding the importance of code modification related to maintainability. Moreover, the empirical data in this research can benefit source code improvements made by software developer communities in OSS projects. Additionally, the code review trends illustrated in this study can improve decision-making processes among software developers of OSS projects, who should prioritize software maintainability. In addition, these developers should be ready to perform code editing in situations or environments that may change in the future.

Although the results of this research provide only basic conclusions, the existing information is sufficient for guiding the development and improvement of OSS quality. Moreover, the findings can be used by researchers who are interested in software maintainability to study related topics, such as plug-in development, using Gerrit. This comment-related approach could improve code modification and extend beyond maintainability to other topics, such as readability and testability. Additionally, tools or processes could be developed to assess the edits of software developers in OSS projects according to reviewer comments in Gerrit. If such a script or program was developed to automatically check code edits in the Gerrit framework, the reviewers who provide the comments could be notified to facilitate better communication within the software development community and allow for faster subsequent reviews.

REFERENCES

- [1] 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 30th international conference on Software engineering*. ACM, 2008, pp. 541–550.
- [2] M. Aberdour, "Achieving quality in open-source software," *IEEE Software*, vol. 24, no. 1, pp. 58–64, 2007.
- [3] V. Tiwari and R. Pandey, "Open source software and reliability metrics," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 1, no. 10, pp. 808–815, 2012.
- [4] B. Norick, J. Krohn, E. Howard, B. Welna, and C. Izurieta, "Effects of the number of developers on code quality in open source software: a case study," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2010, pp. 1–1.
- [5] Y. Zhou and J. Davis, "Open source software reliability model: an empirical approach," in *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4. ACM, 2005, pp. 1–6.
- [6] G. S. Walia and J. C. Carver, "Using error information to improve software quality," in *Proceedings of the 2013 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2013, pp. 107–107.
- [7] A. Bosu and J. C. Carver, "Peer code review in open source communities using reviewboard," in *Proceedings of the ACM 4th annual workshop on Evaluation and usability of programming languages and tools*. ACM, 2012, pp. 17–24.
- [8] 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, 2016.
- [9] R. A. Baker Jr, "Code reviews enhance software quality," in *Proceedings of the 19th international conference on Software engineering*. ACM, 1997, pp. 570–571.
- [10] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," in *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 146–156.
- [11] M. Bernhart, A. Mauczka, and T. Grechenig, "Adopting code reviews for agile software development," in *Proceedings of the 2010 Agile Conference (AGILE 2010)*. IEEE, 2010, pp. 44–47.
- [12] M. Fagan, "Design and code inspections to reduce errors in program development," in *Software pioneers*. Springer, 2002, pp. 575–607.
- [13] 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 *Proceedings of the 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 141–150.
- [14] International Organization For Standardization, "Software engineering - Software product Quality Requirements and Evaluation (SQuARE) - System and software quality models," *ISO/IEC 25010:2011*, 2011.
- [15] S. Ghosh and A. K. Rana, "Comparative study of the factors that affect maintainability," *International Journal on Computer Science and Engineering*, vol. 3, no. 12, pp. 3763–3769, 2011.
- [16] N. S. A. A. Bakar and N. Arsat, "Investigating the factors that influence the quality of open source systems," in *Proceedings of the 5th International Conference on Information and Communication Technology for The Muslim World (ICT4M)*. IEEE, 2014, pp. 1–6.
- [17] J. Walden, M. Doyle, G. A. Welch, and M. Whelan, "Security of open source web applications," in *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2009, pp. 545–553.
- [18] B. Norick, J. Krohn, E. Howard, B. Welna, and C. Izurieta, "Effects of the number of developers on code quality in open source software: a case study," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2010, pp. 62–62.
- [19] D. C. Schmidt and A. Porter, "Leveraging open-source communities to improve the quality & performance of open-source software," in *Proceedings of the 1st Workshop on Open Source Software Engineering at ICSE 2001*, 2001, pp. 1–5.
- [20] J. Asundi and R. Jayant, "Patch review processes in open source software development communities: A comparative case study," in *Proceedings of the 40th Annual Hawaii International Conference on System Sciences, 2007*. IEEE, 2007, pp. 166–171.
- [21] P. C. Rigby, "Understanding open source software peer review: Review processes, parameters and statistical models, and underlying behaviours and mechanisms," Ph.D. dissertation, 2011.

- [22] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 2013 international conference on software engineering*. IEEE, 2013, pp. 712–721.
- [23] Y. Tao, D. Han, and S. Kim, "Writing acceptable patches: An empirical study of open source project patches," in *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2014, pp. 271–280.
- [24] 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 37th International Conference on Software Engineering*. IEEE, 2015, pp. 27–28.
- [25] R. Glass, "Frequently forgotten fundamental facts about software engineering," *IEEE Software*, vol. 18, no. 3, pp. 112–111, 2001.
- [26] B. C. Wagey, B. Hendradjaya, and M. S. Mardiyanto, "A proposal of software maintainability model using code smell measurement," in *Proceedings of the 2015 International Conference on Data and Software Engineering (ICoDSE)*. IEEE, 2015, pp. 25–30.
- [27] I. Kádár, P. Hegedűs, R. Ferenc, and T. Gyimóthy, "A manually validated code refactoring dataset and its assessment regarding software maintainability," in *Proceedings of the 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2016, pp. 1–4.
- [28] K. Stroggylos and D. Spinellis, "Refactoring—does it improve software quality?" in *Proceedings of the 5th International Workshop on Software Quality (WoSQ'07: ICSE Workshops 2007)*. IEEE, 2007, pp. 10–10.
- [29] S. Rizvi and R. A. Khan, "Maintainability estimation model for object-oriented software in design phase (MEMOOD)," *Journal of Computing*, vol. 2, no. 4, pp. 26–32, 2010.
- [30] S. A. Khan and R. A. Khan, "Analyzability quantification model of object oriented design," *Procedia Technology*, vol. 4, pp. 536–542, 2012.
- [31] S. Ghosh and A. K. Rana, "Comparative study of the factors that affect maintainability," *International Journal on Computer Science and Engineering*, vol. 3, no. 12, p. 3763, 2011.
- [32] A. Bosu, J. C. Carver, M. Hafiz, P. Hillel, and D. Janni, "Identifying the characteristics of vulnerable code changes: An empirical study," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 257–268.
- [33] K. Yamashita, C. Huang, M. Nagappan, Y. Kamei, A. Mockus, A. E. Hassan, and N. Ubayashi, "Thresholds for size and complexity metrics: A case study from the perspective of defect density," in *Proceedings of the 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2016, pp. 191–201.
- [34] B. Caglayan, A. Bener, and S. Koch, "Merits of using repository metrics in defect prediction for open source projects," in *Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*. IEEE, 2009, pp. 31–36.
- [35] S. Counsell, X. Liu, S. Eldh, R. Tonelli, M. Marchesi, G. Concas, and A. Murgia, "Re-visiting the 'maintainability index' metric from an object-oriented perspective," in *Proceedings of the 41st Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2015, pp. 84–87.