International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-12 Issue-2, July 2023

The Effectiveness of Code Reviews on Improving Software Quality: An Empirical Study

D. I. De Silva, W.A.C Pabasara, S.V Sangkavi, Wijerathne L.G.A.T.D, Wijesundara W.M.K.H, Reezan S.A

Abstract: ***This study intends to investigate how well code reviews contribute to higher software quality. A group of developers working on the study examine source code to find flaws, improve readability, and guarantee compliance with coding standards. The research on the effects of code review on defect discovery, defect prevention, and code maintainability is given together with a thorough overview of the literature on code review and software quality. This study has shown that code review is a highly efficient way to raise the caliber of software. According to the study of several studies and trials, code review significantly reduces flaws and improves code maintainability. Code review helps to increase client satisfaction by making sure the product satisfies their needs. The goal of this study is to highlight the value of code review as a quality assurance technique in software development workflows. The study's findings provide useful information for software development teams by emphasizing the advantages of code review for raising software quality and customer satisfaction. The results of this study can assist software development teams in incorporating code review into their workflows as a normal procedure, which would improve software quality and cut down on mistakes. In conclusion, this study shows that code review is a highly efficient way to raise the caliber of software. In terms of fault detection and prevention, code maintainability, and customer satisfaction, the study underlines the benefits of code review. This study can influence software development teams to make code review a common practice by highlighting its advantages, which would increase product quality and client satisfaction.***

Keywords: Code review, Software quality

Manuscript received on 06 May 2023 | Revised Manuscript received on 17 May 2023 | Manuscript Accepted on 15 July 2023 | Manuscript published on 30 July 2023.

*Correspondence Author(s)

Dr. D. I. De Silva, Department of Computer Science and Software Engineering, Sri Lanka Institute of Information Technology, Malabe, Sri Lanka E-mail: dilshan.i@sliit.lk, ORCID ID: https://orcid.org/0000-0001-6821-488X

W. A. C Pabasara, Department of Computer Science and Software Engineering, Sri Lanka Institute of Information Technology, Malabe, Sri Lanka E-mail: chamali.p@sliit.lk, ORCID ID: https://orcid.org/0009-0000-6431-8251

S. V Sangkavi*, Department of Computer Science and Software Engineering, Sri Lanka Institute of Information Technology, Malabe, Sri Lanka E-mail: vithyasangkavi@gmail.com, ORCID ID: https://orcid.org/0009-0002-6350-6714

Wijerathne L.G.A.T.D, Department of Computer Science and Software Engineering, Sri Lanka Institute of Information Technology, Malabe, Sri Lanka E-mail: thilakna@gmail.com, ORCID ID: https://orcid.org/0009-0001-7635-7431

Wijesundara W.M.K.H, Department of Computer Science and Software Engineering, Sri Lanka Institute of Information Technology, Malabe, Sri Lanka E-mail: kaushanihw@gmail.com, ORCID ID: https://orcid.org/0009-0007-1133-9842

Reezan S.A, Department of Computer Science and Software Engineering, Sri Lanka Institute of Information Technology, Malabe, Sri Lanka E-mail: imreezan@gmail.com, ORCID ID: https://orcid.org/0009-0003-5104-2059

I. INTRODUCTION

A. Background

Code review is a widely used method to find errors and raise the quality of software during the development process. A code review is a procedure where a piece of code is examined by someone other than the author(s) [1]. Before it is included into the main codebase, software code must be reviewed for errors, bugs, and other problems. The purpose of code review is to raise the general level of the software's quality, lower the number of errors, and boost its dependability. Code review can be done in two ways either face to face review or online review such as google meets, email and etc.

Every sophisticated software development project must have a code review process that evaluates developer-submitted code. One of the best QA procedures for software projects is code review, which is seen as being quite expensive in terms of time and effort but offers significant value in terms of spotting errors in code modifications before they are committed to the project's code base [2].

According to the survey done by our team 75% of the experts have the habit of conducting code reviews frequently while 25% conduct code reviews rarely. Code reviews can be carried out either manually or automatically using tools. Developers or team members review the code manually and offer feedback based on their knowledge and experience. Software tools are used in automated code review to examine the code and find any potential problems. According to [3] website 60% of Developers are using automated tools; 49% are using them at least weekly.

Numerous studies have been conducted to investigate the effectiveness of code review in improving software quality. These studies have looked at a variety of elements of code review, including how it affects maintainability, developer productivity, and problem identification. As stated in a blog, high quality, on boarding a new employee, knowledge exchange, increased consistency and time saving are some advantages of code review [4].

Creating a robust code review procedure lays the groundwork for ongoing development and stops unstable code from being released to users. In order to increase code quality and make sure that every piece of code has been reviewed by another team member, code reviews should be incorporated into the workflow of software development teams.

Another crucial step in transferring knowledge throughout a company is the code review procedure. In addition to these factors, 76% of developers who participated in the 2022 *Global DevSecOps* Survey indicated code reviews are "very valuable" [5].

Despite the potential advantages of code review, there are some drawbacks to this process as well. For instance, code review can take a lot of time and effort from developers. The size of the codebase, the complexity of the code, and the reviewers' level of experience are other variables that may affect how effective code review is [6].

Given these factors, additional research is required to examine the effectiveness of code review in various situations and to discover how to best utilize this technique, with regard to some quality criteria including maintainability, testability, dependability, security, usability, correctness, efficiency, and scalability. The goal of the current study is to advance this field of study by systematically reviewing the body of literature and analyzing the effectiveness of code review on improving software quality.

B. Problem Statement

Software development is a process that involves several stages, including requirements gathering, design, coding, testing, and maintenance. The quality of the software produced is essential to its success, and several quality assurance practices are employed to ensure that the produced software meets the quality standards. Code review is one such practice that is used in the software development industry to detect errors and improve software quality [9].

Code review is a process that check the source code by a group of developers to identify defects and suggest improvements. Code review can be performed manually, where developers review code by reading through it, or it can be automated, where tools are used to analyse the code and identify defects automatically. Code review is also performed at different stages of the software development process, such as during coding, before testing, or during maintenance [9].

Even though the use of code review in software development is very commonly practiced, its effectiveness in improving software quality is still a topic of debate among researchers. There's very little research that were conducted to address this problem. While some studies have reported significant improvements in software quality metrics such as defect density, code maintainability, and overall software quality, others have found little or no improvement [7,10].

Furthermore, the effectiveness of code review might depend on several factors, such as the type of code review, the expertise of the reviewers, the complexity of the code, and the tools used [8]. The lack of general agreement on the effectiveness of code review in improving software quality and the factors that influence its effectiveness makes the need for further research in this area clear. Therefore, the goal of this research paper is to investigate the effectiveness of code review in improving software quality and to identify the factors that influence its effectiveness. To achieve this goal, the following research questions will guide the study:

1. What is the impact of code review on software quality?
2. What are the factors that affect the effectiveness of code review in improving software quality?
3. How can code review best be implemented to improve software quality?

To answer these questions, the study will conduct a literature review of articles published in the last ten years that investigate the effectiveness of code review in improving software quality.

C. Significance of Study

In software development, code review is a critical practice where developers examine the code for potential security and quality issues [15]. The review process can take several forms, including tool-assisted (TA), over-the-shoulder (OTS), peer-to-peer, and pair programming [14]. Despite its widespread use, there is still a lack of empirical evidence on how different code review methods impact software quality [12].

Previous research, including Bosu *et al.'s* work, has examined some effectiveness factors of code review, such as review duration, size, and the number of flaws found [16]. However, additional studies are required to analyze the influence of various techniques and review size on code review effectiveness, as pointed out by a study conducted by Jureczko *et al.* [14].

The present study seeks to contribute to the software development field by analyzing the effectiveness of various code review methods in identifying quality attributes that affect software quality. The authors surveyed software engineers at *hSenid Mobile Solutions* to identify the most critical quality attributes for code reviews, and selected eight attributes that were highly rated for this study. The study aims to compare the effectiveness of different code review methods in addressing these quality attributes.

Code review methods can differ in their impact on the selected quality attributes, as each method is conducted in a unique way. Tool-assisted code review involves specialized tools to facilitate the review process [17], while other methods like over-the-shoulder, peer-to-peer, and lead code reviews involve direct interaction between the author and the reviewer [18]. By comparing the effectiveness of these different methods, the study aims to identify which method is best suited for detecting and addressing quality issues in software development.

The expected outcomes of this study are:

- A better understanding of the strengths and weaknesses of different code review methods in terms of software quality and security.
- A set of guidelines and best practices for choosing and applying code review methods in different contexts and projects.
- A contribution to the improvement of software development processes and practices that can enhance the quality attributes discussed in this journal.

This study will benefit the academic community by addressing a gap in empirical research on code review methods. In addition to benefits for academia and the software industry, this work will have larger impacts on society.

High-quality software is crucial in numerous aspects of modern life, such as healthcare, transportation, and banking [17]. Improving code review procedures can enhance the effectiveness, reliability, and safety of software products, benefiting both users and society as a whole [18].

Software mistakes can have major effects on patients, such as improper diagnosis or therapy. Software faults in the transportation industry may result in mishaps or system breakdowns with lasting effects [17]. This study offers the ability to raise software's safety and quality across a wide range of sectors by enhancing code review procedures [18].

Moreover, this study may also have economic benefits. Software flaws can be expensive, both in terms of the expense of repairing them and the impact on users [19]. This study can decrease the number of errors that reach the production stage by identifying the most efficient code review techniques, saving both software development teams and end users time and money [19]. High-quality software development can also give businesses a competitive edge.

D. Objectives

The primary objective of this research study is to thoroughly investigate the effectiveness of code reviews in improving software quality and to identify the different types of code reviewing methods that are commonly used in the industry itself. Generally, the intention of a code review is to identify and fix errors and other issues in the code, which will help to maintain and improve the overall quality of the software.

In this study, the effectiveness of code review will be thoroughly discussed, while existing studies are comprehensively searched and examined. Additionally, a survey will be conducted among software development professionals to gather information on their opinions regarding code reviews. The survey will include questions about the preferred methods of conducting a code review, the benefits of code review that most affect the improvement of software efficiency, and how effective code review is in improving software quality.

Another important objective of this study is to assess the effectiveness of different types of code reviews on software quality improvement. There are several code review methods that are typically used such as tool assisted reviews, automated reviews, peer reviews, walkthroughs, and formal inspection.

Analyzing the effectiveness of these different methods of code reviews is very important in order to learn how to improve software quality.

This can help in reducing development time, help to improve team collaboration and communication as well. The ultimate goal of analyzing the effectiveness of different types of code reviews is to identify the most efficient method that is suitable for a particular developing environment.

Another goal of this research study is to identify the critical components that affect the success of code reviews. The success of code review may be influenced by other points as well. One of the main key factors is reviewer expertise. There are many studies that show that code reviewer expertise is a major factor that affects the success of code reviewing as well [15]. Other key factors are code review frequency, code review process, code review tools.

The different types of code reviews methods that are commonly used will be examined through this research paper as well. There are several code review methods that are typically used such as tool assisted reviews, over the shoulder, walkthroughs, and formal inspection.

The impact of code review on various software quality attributes, such as code maintainability, testability, reliability, security, and usability, will also be discussed in this review.

Also, another objective of this study is to analyze and discuss the relationship between code review and other software engineering practices.

The aim of this study is to provide software developers and managers with alternative advice on optimizing their software development process and maintaining software quality with the collaboration of code reviewing and other software engineering practices.

E. Research Questions

1. What is the impact of code review on software quality?
2. What are the factors that affect the effectiveness of code review in improving software quality?
3. How can code review best be implemented to improve software quality?

The first research question will be answered by looking into how code review affects the caliber of software. To determine the advantages and disadvantages of code review and to calculate its effect on software quality, it will be necessary to analyze the current literature and case studies.

The impact of code review in raising software quality will be examined in the second research question. In order to learn more about software developers' experiences with code reviews, including the types of reviews utilized, the experience of reviewers, and the degree of automation employed in the process, a survey of software developers will be conducted. The factors that are most strongly linked to successful code review will be found through statistical analysis of the data. The third study topic will concentrate on the most effective way to use code review to raise the caliber of software. In order to do this, best practices for code review will be created based on the answers to the first two research questions. A controlled experiment will test these best practices to determine whether they can actually raise software quality.

In general, the research seeks to provide light on the efficiency of code review in raising software quality and to pinpoint tactics for enhancing its application. The study will advance the discipline of software engineering and help with current efforts to enhance software quality by providing answers to these research topics.

II. LITERATURE REVIEW

Previous research has examined the methods of code inspection and code review and how they help to raise the caliber of software. Using the CAIS (Collaborative Asynchronous Inspection of Software) tool, Stein *et al.* did research on distributed, asynchronous code inspections [18].

They discovered that asynchronous inspections utilizing CAIS were successful in locating flaws and raising the caliber of software. As participants could evaluate code at their own pace and on their own schedule, the authors discovered that asynchronous inspections were more flexible and effective than conventional synchronous inspections.

In a study he conducted on code inspection practices, Laitenburger produced a taxonomy of different code inspection methods [19]. Johnson looked into how open-source software project managers' judgments were affected by code review processes [19].

The study in question examines various facets of code review, hence it has been broken up into several subsections, each of which discusses a separate field of research. As was done in the study by Jureczko *et al*. [14], the term "patch" has been utilized in this study to refer to changesets for consistency's sake.

A. Effectiveness of the Review Process

The success of code review is influenced by a number of factors, which have been thoroughly investigated in numerous studies. According to Porter *et al*. [17] and Sauer *et al*. [21], the essential components in enhancing inspection efficacy are the skill of the code reviewer and the adoption of superior flaw detection tools. Sauer *et al*. found that two reviewers are the ideal number for the most efficient code review, and this result is in line with behavioral studies that implies that expert pairs outperform larger groups [20].

According to Jureczko *et al*., over-the-shoulder code reviews produce a higher rate of knowledge transfer than tool-assisted code review approaches, which can ultimately lead to superior software quality. To identify the proper areas to which any of these strategies can contribute, more study is required [14]. This study will be a longer version of that study, and some of its facts and data may originate from it.

In a study of code reviews in two significant open-source projects, Thongtanunam *et al*. [22] found a connection between software quality and both code ownership and reviewers' skill. Code ownership refers to the concept that certain developers have greater expertise and responsibility for specific parts of the codebase. The study discovered that when code ownership algorithms that consider review activity were used, there was a positive impact on software quality. Additionally, the study found a substantial and upward trending link between the likelihood of post-release errors and the percentage of reviewers lacking in knowledge. This suggests that when reviewers lack knowledge or expertise in the area of code they are reviewing, it can increase the likelihood of errors being introduced into the software after its release.

These findings imply that the effectiveness of code reviewers and their involvement in the process are key elements in increasing software quality through code reviews. It is important for reviewers to possess the necessary knowledge and expertise to effectively review the code and provide valuable feedback.

In a separate investigation into the peer review procedures used by the Apache HTTP server project, Rigby *et al*. [23] found that smaller fixes were more likely to result in higher-quality code and were therefore more likely to be accepted.

McIntosh *et al*. looked into the relationship between the quality of software and the involvement in and coverage of the modern code review process [24]. They found that there is a strong correlation between code review participation and program quality. There may be up to two and five more post-release issues for components with low code review coverage and involvement, respectively. This suggests that badly reviewed code in a complex system can have a negative impact on the caliber of software.

Similar to this, Morales *et al*. investigated the relationship between software design quality and code review practices. They found that software components with less review coverage or involvement are typically more prone to anti-patterns than those with more active code review methods when measuring software design quality by the prevalence of seven different types of anti-patterns. The open-source programs Qt, VTK, and ITK were the basis for this case study [22]. In a study of a major commercial project, Dos Santos and Nunes [23] found that when there were a large number of altered lines of code, the review process took longer and received fewer comments. This suggests that when developers made many changes to the code, it required more time for reviewers to go through all the changes and provide feedback.

The study also discovered that having more participating teams and reviewers increased contributions to the review process, but at the cost of time. This indicates that while having more people involved in the review process can result in more feedback and discussion, it can also take longer to complete the review.

Ultimately, the study points out that for larger patches of code, reviewers were less engaged and provided less feedback. This implies that when there are many changes to review, it can be more challenging for reviewers to thoroughly examine all the changes and provide detailed feedback. Communication during code reviews is another important factor to consider. In their study on communication in code review, Ebert et al. [24] found that confusion was a common issue. The study explains that this confusion often arose from a lack of clarity and explanation of non-functional aspects of the solution. Non-functional aspects refer to characteristics of the software that are not directly related to its functionality, such as performance, security, and maintainability. If these aspects are not clearly explained and understood, it can lead to confusion during the review process. Additionally, the study found that issues with tools and communication could also contribute to confusion during code reviews. For example, if there are disagreements or unclear communicative objectives between developers, it can lead to confusion and delays in the review process.

This confusion can have several negative effects on the code review process. It can result in delays, less effective reviews, increased discussions, and lower quality software. To address these issues, it is important for developers to communicate clearly and effectively during code reviews to minimize confusion and ensure that the process is efficient and effective.

In addition, some discussions may result in a greater number of rejected comments. Thus, to some extent, this study provides a calculated advantage code reviews can have on determining software quality.

B. Classfication Of Reviews

The study's research has shown that different methods of code reviews produce different amounts of pointless comments. An extensive examination into the incidence of such remarks in various code review methodologies was done by Jureczko *et al*. According to the study, there were significantly more or less nonsensical comments for various review kinds. In comparison to over-the-shoulder reviews, the researchers found that tool-assisted review procedures produced fewer pointless comments [14]. These findings imply that the effectiveness and efficiency of the code review process can be considerably impacted by the choice of review methodology.

In a research by Mäntylä and Lassenius, which looked at 23 student Java projects and nine commercial C/C++ programs, flaws were categorized. The study found 388 and 371 flaws in the student and industrial projects, respectively, and showed that 75% of the flaws found during code review had no effect on the software's apparent functionality. These flaws were discovered to improve software evolvability by increasing its understandability and modifiability. Code reviews can thereby increase software quality by discovering flaws that enhance program evolvability. Code reviews can help to increase the software's evolvability and, as a result, its overall quality by locating and fixing flaws that make it easier to comprehend and modify.

An exhaustive mixed-methods study by Bosu, Greiler, and Bird entitled "An Empirical Study at Microsoft" sought to identify the elements that contribute to effective code reviews. Three parts made up their study: first, they carried out a qualitative analysis to determine the features of code reviews that engineers found useful. Based on their qualitative findings, they subsequently created and verified a categorization model that could distinguish between relevant and pointless code review feedback. To study the elements that contribute to more effective code review feedback, they utilized their classifier to analyze 1.5 million review comments from five Microsoft projects [13].

The researchers found a number of variables that influence the worth of code review feedback. For instance, they discovered that during a reviewer's first year at Microsoft, the proportion of valuable remarks they make climbs dramatically, but after that, it tends to stagnate. They also found that the proportion of code review comments that will help the author of the change is negatively correlated with the number of files in the modification. In addition, they discovered that for some projects, comparing changes that span more than 40 files to those made up of only one file could result in a 10% drop in the number of meaningful comments [14]. These results emphasize how crucial it is to pinpoint the elements that lead to efficient code reviews in order to increase their use and, ultimately, improve the quality of the software being developed.

C. Transparency of code

The knowledge transparency of the code across various reviewers and programmers is one of the critical elements contributing to well-defined code reviews, according to extensive study in this area. This suggests that code reviews facilitate developer knowledge sharing and encourage a better comprehension of the code base among teams. Developers can improve the overall quality of the product and significantly reduce the amount of coding errors by exchanging expertise and spotting potential problems in the code.

When compared to tool-assisted code reviews, Jureczko *et al*. found that the over-the-shoulder method enhances knowledge transfer in MCR. They did, however, note that this does not always lead to evaluations of a better caliber. In contrast to tool-assisted code reviews, they found that most of the comments gathered through the over-the-shoulder method were minor, indicating that they did not address any significant functional changes that needed to be made to the code [14]. However, the authors stress the importance of taking knowledge transfer and review quality into account when selecting a code review technique because each method may have benefits and drawbacks depending on the demands placed on it by the project and the team. The expectations, results, and difficulties of the existing code review process were examined by Bacchelli and Bird [11]. They emphasized the importance of team members sharing knowledge and working together during the code review process, even if they did not directly address knowledge transfer through code review. They found that code review promotes knowledge sharing, high-quality code, and teamwork in addition to finding problems. Additionally, they identified a number of issues with code reviews, including the need for effective tools and procedures, overcoming reviewer biases, and juggling code reviews with other development tasks.

In their study "Knowledge Transfer in Modern Code Review," Caulo *et al*. looked into whether developers' contributions to open-source projects can get better over time thanks to the code review procedure. They examined numerous peer-reviewed pull requests that developers had posted to GitHub, making the assumption that if a developer's pull request had previously been reviewed, knowledge had been passed on to that developer during the code review process. They then evaluated whether, as more of their pull requests were examined, the developer's contributions to open-source projects got better over time. They failed to uncover proof, nonetheless, that the code review procedure improved the caliber of developers' contributions [25]. However, Jureczko *et al*. noted that Caulo *et al*. studied tool-assisted reviews while Jureczko *et al*.'s tests showed that over-the-shoulder reviews have a considerably bigger influence on knowledge transfer [14] and that tool-assisted reviews do enhance knowledge transmission to some level. As a result, depending on the specific approach taken, the efficiency of the code review process in raising software quality and fostering knowledge transfer may differ.

The present study aims to provide insight into selected quality attributes of software, informed by the findings of the literature studies reviewed during its conduct.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

III. METHODOLOGY

Based on a thorough analysis of the current literature, the goal of this study is to determine how well code reviews contribute to higher software quality. Below is a description of the study's methodology.

Survey: An online survey was administered, which consisted of questions pertaining to how code review influences various software quality attributes such as maintainability, testability, usability, correctness, reliability, security, and efficiency. Additionally, participants were asked to provide their demographic information, including job role and level of experience. Some of the common questions that this study covers through the journal are,

1. How often does the participant participate in code reviews?
2. What is the most effective way to integrate code review into the development process for maximum efficiency?
3. Which benefit of code review do participants consider the most significant for enhancing software efficiency?
4. How much time does code review typically add to the testing process for scalability?
5. When working on a security-related aspect of a software application, how useful have code reviews been in helping the participant understand their mistakes and adopt safer coding practices.
6. How do you track and measure the correctness of your software over time?
7. Which aspects of software reliability have participants noticed the most significant improvements due to code reviews, based on their personal experiance?

Search Strategy: To conduct a systematic review of the literature, a structured search strategy is used to identify relevant studies: Google Scholar, Scopus, and other journals and articles. Inclusion Criteria: Articles must adhere to the following inclusion requirements in order to be considered for this study:

1. Accentuate how successful code review is at enhancing software quality.
2. Describe empirically how code review affects software quality.
3. Published in conference proceedings or a peer-reviewed journal.
4. Presented in English

Exclusion Criteria: The following exclusion criteria will prevent articles from being included in this study:

1. Consider subjects other than code quality and code review.
2. Don't include empirical information about how code review affects the caliber of software.
3. Not included in conference proceedings or peer-reviewed journals
4. Not in English but written in another language.

Ethics: This study adhered to ethical standards for human subjects' research, which include gaining informed consent from each participant and respecting their anonymity and privacy. In conclusion, the study employed a mixed-methods research design to investigate the contribution of code review to higher software quality. Data was gathered through a survey and analysis of the code review process, which will be evaluated using a combination of qualitative and quantitative techniques. The study's findings will enrich the existing knowledge on the significance of code review and provide suggestions on how to use it most efficiently in software development.

IV. RESULTS

The study of source code by one or more people in order to find and correct errors is known as code review, and it is a crucial step in the software development process. Through error detection and improved maintainability, code reviews can raise the quality of software. The efficiency of code reviews has been the subject of numerous research, which have also revealed several variables that may affect the review process's quality. A survey was conducted to gather insights from software developers and practitioners on the various methods of code review, their benefits and challenges, and their impact on software quality. The poll included a number of inquiries about code review and how it affects the quality of software. ‘What is the most effective way to integrate code review into the development process for maximum efficiency? ‘, was one of the main questions posted. Five alternatives were given to respondents:

Fig 1. Survey results on code review

Retrieval Number: 100.1/ijrte.B76660712223
DOI: 10.35940/ijrte.B7666.0712223
Journal Website: www.ijrte.org

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

56% of those polled said the best way to incorporate code review into the development process was to evaluate the code before merging it into the main branch. This method involves analyzing code modifications before they are merged into the main branch, assisting in the early detection of bugs and issues.

18.8% of respondents thought that performing code reviews on a frequent basis (weekly or monthly, for example) was the best strategy. With the help of this method, which entails periodically reviewing a set of changes, the codebase can be reviewed in greater detail over time.

12.5% of those surveyed said that performing code reviews after each commit was the best strategy. This method requires analyzing each modification to the code as soon as it is committed, giving quick feedback, and spotting mistakes early.

The survey's findings can offer useful information about how to incorporate code review into the development process for maximum effectiveness and enhanced software quality.

Another main question that was asked focusing on the quality attributes was 'Which of the following benefits of code review do you think is most important for improving software efficiency'. Respondents were given four options

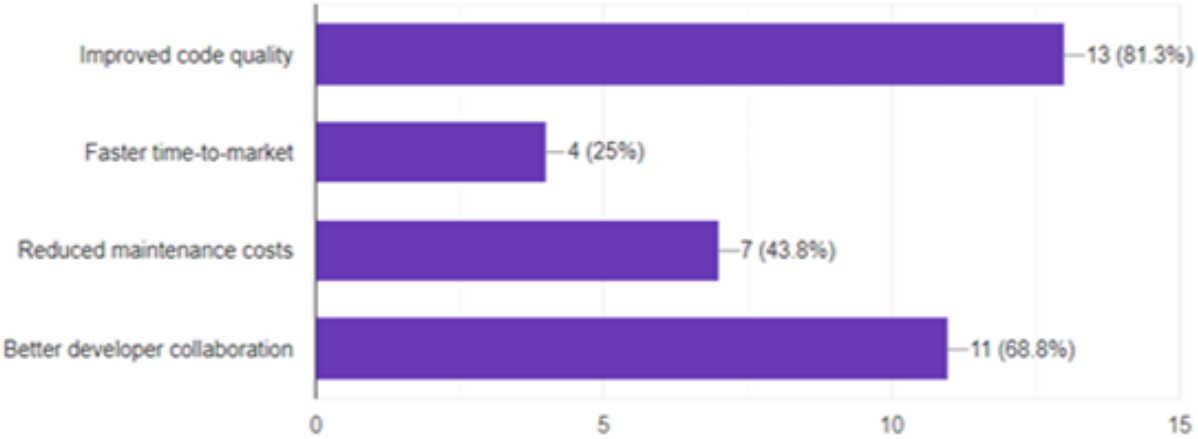

Fig 2. Survey results on quality attribute efficiency

Improved code quality, according to 81.3% of respondents, is the primary advantage of code review for increasing software effectiveness. This advantage includes spotting flaws and problems at an early stage of development, making sure the code is maintainable, and improving the quality of the software product.

Better developer collaboration, according to 68.8% of respondents, is the main advantage of code review. This advantage entails encouraging developer collaboration and communication, promoting knowledge sharing and learning, and developing a sense of shared accountability for code quality.

Most responders (43.8%) thought that lower maintenance costs were the most significant advantage of code review. Early bug detection reduces the need for time-consuming debugging and maintenance labor, which eventually results in time and cost savings.

Faster time-to-market was deemed the most significant benefit of code review by 25% of respondents. This advantage includes early error and problem detection, decreased rework requirements, and eventually faster software delivery.

Another important question that was asked in the survey was 'How can code review help to improve software maintainability. 'Respondents were given with three options.

Fig 3. Survey results on quality attribute maintainability

According to the respondents, 81.3% said that the most crucial method that code review can increase software maintainability is by locating and addressing errors that may have an impact on it. This entails going over code updates to find any potential problems, including complicated code or confusing logic, that can affect the software's capacity to be maintained over time. Code review can help to guarantee that the codebase is manageable and simple to work with in the long run by identifying these problems early.

According to 87.5% of respondents, one significant way that code review can increase software maintainability is by making the code easier to read and understand. This entails checking updated code to make sure it is simple to read and comprehend and that it makes use of clear and breif naming conventions, comments, and documentation. Code review can make the codebase easier to read and understand, which will make it easier to maintain and work with over time. According to 75% of respondents, one significant way that code review can increase software maintainability is by ensuring that the code is compatible with the design and specifications. This entails checking updated code to make that it adheres to project specifications and is consistent with the software's overall design. Code review can make it simpler to maintain and update over time by verifying that the codebase is consistent and in line with the project goals.

The findings of this survey can offer insightful information about how code review can enhance software maintainability, and they can assist firms in streamlining their code review procedures to achieve this crucial objective.

Overall, by considering numerous elements such the reviewer's experience, the use of diverse approaches, and the involvement of reviewers in the process, the effectiveness of code review can be increased. Code reviews are becoming more and more crucial as software systems become more complicated to ensure software quality and maintainability.

V. DISCUSSION

The results imply that code review is a crucial procedure that can raise the quality of software. According to the research analyzed in this paper, code review can reduce post-release errors, help uncover issues, and increase knowledge transfer.

The reviewers' level of experience is one of the main variables affecting the efficiency of code review. Expert reviewers are more adept at spotting errors and enhancing software quality, according to the studies examined in this paper. The number of reviewers also affects how effective a code review is. Two reviewers have been shown to be the ideal quantity for the most fruitful code review.

Asynchronous code review has the potential to be more effective and adaptable than synchronous code review, which is another significant discovery. Code reviews may be of higher quality when conducted asynchronously since participants can evaluate the code at their own leisure and on their own time. Additionally, depending on the circumstance, it may be beneficial to employ various code review approaches such as tool-assisted and over-the-shoulder reviews.

The results of this study indicate that code review is a useful technique for raising software quality, although there are some restrictions and difficulties to consider. One drawback is that the size and complexity of the codebase being examined may have an impact on how effective code review is. Another drawback is that the review's quality can be significantly impacted by the reviewers' experience and engagement in the review process. Implementing code review in practice may also be difficult due to time restrictions and reluctance to change.

The study's findings show conclusively that code review is a useful strategy for raising the quality of software. The findings imply that asynchronous review and expert reviewers can result in higher-quality reviews. But it's crucial to consider the restrictions and difficulties associated with actually putting code review into practice. Future studies could examine the usefulness of code review in various situations and investigate methods for resolving implementation difficulties.

VI. CONCLUSION

This study has produced convincing evidence in favour of the usage of code review as a useful strategy for raising the quality of software development after performing a thorough analysis into the usefulness of code review on improving software quality. The study's findings show that code review can significantly raise software quality, especially when it comes to lowering defect rates, making software more maintainable, and boosting output.

The advantages of code review are numerous. First and foremost, the code review procedure aids in the early detection of errors in the development process, before they can evolve into expensive and time-consuming issues. This is crucial in the software development industry since even the smallest mistakes can quickly add up to serious failures. Code review can help issues get caught early and stop them from getting out of hand, which will ultimately result in a more effective development process.

Additionally, it has been demonstrated that code review is a powerful tool for lowering software flaws. This is due to the fact that code review promotes developer collaboration and information sharing, which aids in finding mistakes and potential defects that could have gone unnoticed during individual coding efforts. Additionally, the comments and recommendations made during a code review can assist raise the calibre of the code and guarantee that best practices are being followed.

Finally, code review can enhance the software's ability to be maintained, making it simpler for engineers to update the code over time. This is due to code review's role in ensuring that code is well-structured, well-documented, and written in a clear and understandable manner. Code review can assist extend the life of software and guarantee that it stays useful and relevant over time by enhancing maintainability.

The implementation of code review still faces difficulties despite its many advantages. Constraints of time and resources are a significant issue, especially in organizations that are already overburdened with competing agendas.

Retrieval Number: 100.1/ijrte.B76660712223
DOI: 10.35940/ijrte.B7666.0712223
Journal Website: www.ijrte.org

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

As a result, it's crucial to create methods for streamlining and increasing the efficiency of code review, such as by employing automation tools or creating explicit rules for the procedure. Future research has a lot of chances to build on the findings of this study, in the future. For instance, more research might look into how the effects of code review change for other softw0are project types, such as large-scale systems or safety-critical applications. Research may also examine the effects of various ways to code review, such as pair programming or tool-assisted code review. Future study can assist improve and modify the practice of code review to better fulfil the demands of software engineers and organizations by examining these and other issues. In conclusion, this study offers compelling evidence in favour of the usage of code review as an effective method for raising the calibre of software. Code review can help to guarantee that software is of good quality and remains relevant over time by finding problems early on, minimizing faults, and enhancing maintainability. While implementing code review still presents some difficulties, continuous research can assist in overcoming these obstacles and improving the technique to better meet the objectives of software engineers and organizations.

DECLARATION

Funding/ Grants/ Financial Support	No, We did not receive.
Conflicts of Interest/ Competing Interests	No conflicts of interest to the best of our knowledge.
Ethical Approval and Consent to Participate	No, the article does not require ethical approval and consent to participate with evidence
Availability of Data and Material/ Data Access Statement	Not relevant
Authors Contributions	All authors having equal contribution for this article.

REFERENCES

1. "Code Review Developer Guide," [Online]. Available: https://google.github.io/eng-practices/review/.
2. O. B. L. G. Y. C. M. W. G. Oleksii Kononenko, "Investigating Code Review Quality," p. 10, 2015.
3. "Best practices for reviewing a code," [Online]. Available: https://www.codegrip.tech/productivity/best-practices-for-reviewing-code/.
4. "Benefits of Code Review: Every Team Must Know [2022 Guides]," [Online]. Available: https://gaper.io/benefits-of-code-review/.
5. [Online].Available:https://about.gitlab.com/topics/version-control/what-is-code-review/#:~:text=Code%20reviews%2C%20also%20known%20as,developers%20learn%20the%20source%20code.
6. [Online]. Available: https://www.freecodecamp.org/news/how-to-avoid-code-review-pitfalls-that-slow-your-productivity-down-b7a8536c4d3b/.
7. Bacchelli, A., & Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. IEEE Software, 30(4), 84-91. doi: 10.1109/MS.2012.175 [CrossRef]
8. Baltes, S., & Diehl, S. (2014). Improving software quality through code review: A case study. IEEE Software, 31(4), 60-67. doi: 10.1109/MS.2013.104 [CrossRef]
9. Kemerer, C. F., & Porter, L. F. (1992). Improving software quality through a formal technical review process. Journal of Systems and Software, 19(2), 119-131. doi: 10.1016/0164-1212(92)90074-f
10. P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE), 2013, pp. 202-212. [CrossRef]
11. A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in Proceedings of the 2013 International Conference on Software Engineering (ICSE), 2013, pp. 712-721. [CrossRef]
12. Beller, M., Bacchelli, A., Zaidman, A., & Juergens, E. (2014). Modern code reviews in open-source projects: which problems do they fix? In Proceedings of the 11th Working Conference on Mining Software Repositories (pp. 202-211). [CrossRef]
13. Bosu, A., Greiler, M., Bird, C.: 'Characteristics of useful code reviews: an empirical study at microsoft'. Proc. of the 20
14. M. Jureczko, Ł. Kajda, and P. Górecki, "Code review effectiveness: An empirical study on selected factors influence," IET Software, vol. 14, no. 7, pp. 794–805, 2020. [CrossRef]
15. J. &. K. Y. Shimagaki, "The effect of reviewer's expertise on code review outcomes," 2018.
16. Devart Software, "Code Review Benefits," [Online]. Available: https://www.devart.com/review-assistant/learnmore/benefits.html.
17. M. Stein, J. Riedl, S.J. Harner and V. Mashayekhi, "A Case Study of Distributed Asynchronous Software Inspections," in Proceedings of the 19th International Conference on Software Engineering, New York, NY, USA: ACM Press, 1997, pp. 107-117. [CrossRef]
18. A. Sutherland and G. Venolia, "Can peer code reviews be exploited for later information needs?," in Proceedings of ICSE, may 2009 [CrossRef]
19. O. Laitenberger, "A Survey of Software Inspection Technologies," in Handbook on Software Engineering and Knowledge Engineering, vol. 1, World Scientific Publishing Co., 2002, pp. 517-555. [CrossRef]
20. A. Porter, H. Siy, A. Mockus and L. Votta, "Understanding the sources of variation in software inspections," in ACM Transactions on Software Engineering and Methodology, vol. 7, no. 1, pp. 41-79, Jan. 1998. [CrossRef]
21. Sauer, C., Jeffery, D.R., Land, L., *et al.*: 'The effectiveness of software development technical reviews: A behaviorally motivated program of research', IEEE Trans. Softw. Eng., 2000, 26, (1), pp. 1–14 [CrossRef]
22. R. Morales, S. McIntosh and F. Khomh, "Do code review practices impact design quality? A case study of the Qt, VTK, and ITK projects," 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Montreal, QC, 2015, pp. 171-180. [CrossRef]
23. Dos Santos, E.W., Nunes, I.: 'Investigating the effectiveness of peer code review in distributed software development'. Proc. of the 2017 Brazilian Symp. on Software Engineering, 2017, pp. 84–93 [CrossRef]
24. Ebert, F., Castor, F., Novielli, N., *et al.*: 'Communicative intention in code review questions'. Proc. of the 2018 IEEE Int. Conf. on Software Maintenance and Evolution, 2018, pp. 519–523 [CrossRef]
25. Caulo, M., Lin, B., Bavota, G., Scanniello, G., & Lanza, M. (2020). Knowledge Transfer in Modern Code Review. In Proceedings of the 28th International Conference on Program Comprehension, pp. 230-240. [CrossRef]

AUTHORS PROFILE

Dr. Dilshan De Silva is a seasoned professional in the field of Information Technology (IT) with a strong background in Software Engineering. They obtained their BSc. Special (Hons.) in Information Technology (Software Engineering) between 2005 and 2009, followed by an MSc. in Information Technology (Software Engineering) from 2009 to 2012. With extensive expertise and experience, Dr. Dilshan currently serves as a Senior Lecturer (HG) in the Department of Computer Science and Software Engineering at the Faculty of Computing, Sri Lanka Institute of Information Technology. In this role, they impart their knowledge and mentor students in the field of Software Engineering. Dr. Dilshan's research interests primarily revolve around Software Complexity, Software Metrics, Machine Translations, and Augmented Reality. They are deeply committed to exploring these captivating areas and advancing the understanding of software engineering practices. with a strong academic foundation, combined with their practical experience and passion for innovative technologies, Dr. Dilshan is well-equipped to contribute significantly to the field of IT. Their expertise in software engineering, research acumen, and dedication to teaching make them an invaluable asset in academia and research projects.

W.A. Chamali Pabasara is a talented individual with a passion for the field of Information and Communication Technology (ICT). Currently pursuing a Master of Science in Information Technology at the University of Moratuwa, Chamali holds a BSc (Special) degree in ICT from the University of Sri Jayewardenepura. With a diverse background in academia, Chamali has gained valuable experience and expertise in the field. They are currently working as an Assistant Lecturer in the Department of Computer Science and Software Engineering at the Sri Lanka Institute of Information Technology. Previously, they served as an Academic Instructor at the same institute and as a Computer Instructor at the University of Sri Jayewardenepura. Chamali's research interests lie in several exciting areas, including Internet of Things (IoT), Data Mining, Machine Learning, Cloud Computing, and Computer Vision. They are dedicated to exploring these domains and making contributions to advance the understanding and application of these technologies. with a strong academic foundation, practical experience, and a genuine enthusiasm for emerging technologies, Chamali is poised to become a valuable contributor to the field of ICT. Their commitment to learning and their drive for innovation make them an asset to any research project or academic endeavor.

Srirajan Vithya Sangkavi is a highly motivated and ambitious 3rd year undergraduate student specializing in Software Engineering at the Sri Lankan Institute of Information Technology. With a commendable CGPA of 3.4, Vithya demonstrates a strong aptitude for academic excellence. Passionate about technology and its impact on society, Vithya actively engages in research activities to broaden their knowledge and skills. Vithya's research interests encompass various aspects of software engineering, including software development methodologies, web app developments, and mobile application development. They are keen on exploring innovative approaches to improve software development processes and enhance user experiences. Vithya have a proactive approach to learning and seek opportunities to collaborate with peers and mentors to further expand their knowledge and contribute to the field. Vithya's academic achievements and dedication to research reflect their commitment to making a meaningful impact in the software engineering domain. With their strong work ethic, analytical thinking, and passion for innovation, Vithya aims to make significant contributions to the field of software engineering in the future.

L.G.A.T.D. Wijerathne is a student at the Sri Lanka Institute of Information Technology (SLIIT), pursuing a BSc (Hons) in Information Technology with a specialization in Software Engineering. With a passion for software development, L.G.A.T.D. Wijerathne skillfully combines their technical expertise with a love for writing. Through their works, they provide unique insights into the world of coding, programming, and software engineering, aiming to inspire fellow students and technology enthusiasts alike. Join L.G.A.T.D. Wijerathne on their journey as they explore the intersection of technology and literature, illuminating the wonders of software engineering through their words. L.G.A.T.D. Wijerathne's commitment to learning and drive for innovation make them an asset to any research project or academic endeavor, constantly pushing the boundaries of software engineering with their insatiable curiosity and creative problem-solving skills.

Wijesundara W.M.K.H. is a third-year software engineering student at the Sri Lanka Institute of Information Technology (SLIIT). She is highly interested in software development and all things technological. Wijesundara has developed a strong foundation in programming languages, algorithms, data structures, and software design principles during the course of her education. To obtain expertise in realistic situations, she has worked on a variety of projects both alone and in teams. Wijesundara actively looks for chances to improve her expertise in software engineering. she has taken part in hackathons and coding contests, displaying her inventiveness and capacity for problem-solving. She has gained teamwork, communication, and time management skills as result of these experiences. Her primary areas of interest in study include the interactions between software engineering and modern fields including data science, machine learning, and artificial intelligence. She wants to help with the creation of innovative software programs that use these technologies to solve problems in the real world. As an author, Wijesundara approaches their work with precision. She carries out in-depth study, exercise critical thinking, and maintain academic integrity. With strong attention to detail and clear writing skills, she effectively communicates her ideas to various audiences. Overall, Wijesundara is a motivated and committed student who aspires to become a successful software engineer. She is willing to share her talents and advance the technological sector.

Reezan S.A. is an undergraduate student at the Sri Lanka Institute of Information Technology (SLIIT), pursuing a Bachelor of Science degree in Information Technology with a specialization in Software Engineering. Currently in his third year of studies, Reezan demonstrates a keen interest in programming and possesses experience in various programming languages, including Java, JavaScript, and C. With a passion for software development, Reezan has successfully completed multiple programming projects and assignments, consistently achieving commendable grades. His dedication to academic excellence is evident as he has consistently maintained a top-ranked position throughout his academic career, currently holding an impressive GPA of 3.84/4. In his free time, Reezan enjoys enriching his knowledge by reading books on technology and programming. His commitment to staying updated with the latest advancements in the field further fuels his passion for software engineering. For any inquiries, collaboration opportunities, or further information, Reezan can be reached at imreezan@gmail.com. He looks forward to contributing to the ever-evolving world of technology and software engineering.

Retrieval Number: 100.1/ijrte.B76660712223
DOI: 10.35940/ijrte.B7666.0712223
Journal Website: www.ijrte.org

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)