Assessing the Effectiveness of MCR-KSM for Waiting Waste Reduction: An Empirical Study

Nargis Fatima1, Sumaira Nazir2, Suriayati Chuprat3
Department of Software Engineering-Faculty of Engineering and Computing, National University of Modern Languages (NUML), Islamabad, Pakistan1, 2
Faculty of Artificial Intelligence, University Technology Malaysia (UTM), Kuala Lumpur, Malaysia3

***Abstract*—Modern Code Review (MCR) is a well-known and widely adopted quality assurance activity to develop quality software. Although it is a core activity for improving code quality, it generates various types of waste, including waiting waste, defect waste, and composite solution waste. Besides all other wastes, the waiting waste is the most critical one, leading to mental distress, delayed code merges, and project delays. Researchers have made efforts to reduce the production of waiting waste by providing various automated code review tools, techniques and models, one of them is the MCR Knowledge Sharing Model (MCR-KSM). The model claims that it supports sustainable software engineering by minimizing waiting waste reduction during MCR activities. This study aims to evaluate the effectiveness of MCR-KSM with respect to the reduction of waiting waste produced during MCR activities. The experiment methodology is employed for this purpose. This paper presents the experimental investigation approach along with the results. The experiment was conducted in dual sessions with 28 graduate students having similar educational and industrial experience. The tools and techniques, such as SPSS paired t-test and value stream mapping, are used for experimental data management and analysis. The study results revealed that the model significantly helps in the reduction of waiting waste production. The conducted study has implications for investigators to extend the research with different parameters and settings.**

Keywords*—*Modern code review; wastes; waiting waste; software quality; automated code review; sustainable software engineering

I. Introduction

Recent software development practices depend on the human aspects of software engineering. For instance, code reviewing [1] that needs verification of code by developers not write it [2]. Code Reviews are the core activity of software development [3]. Besides the quality assurance, the code review is a well-adopted platform for knowledge sharing that can be apparent from pull request discussions on GitHub [4], [5], [6], [7], [8]. Modern Code Review (MCR) is a process where code is submitted, reviewed, discussed and modified before a decision is made on whether to merge it into the main repository or discard the submitted code [1], [9]. Fig. 1 shows the activities involved in the MCR process [10].

As the code review heavily involves the humanoid aspect, specifically for human-to-code and human-to-human interaction for reading, understanding, and providing feedback to modify the code [11], [12]. This interaction and collaboration can result the code review waste production and become inefficient [13] [14]. Waste refers to "Activities that absorb resources and increase cost without adding value". It refers to "Everything that does not make it to the release" The various insights regarding wastes are presented by [15]. The numerous types of waste generated during code review are specifically reported by [13]. The code review wastes can be cognitive load, needless composite solutions, waiting, negative emotions, poor review, poor or delayed feedback, etc. [8], [14], [16], [17].

It is conveyed that the waiting waste is critical and one of the biggest wastes [18], [19], [20]. In code reviews, waiting wastes generated when the author, after submitting the code, waits for reviewers' feedback [21]. In a survey conducted by [22], a survey participant reported, "Usually you write up some code and then you send it out for review, and then about a day later you ping them to remind them ... and then about half a day later you go to their office and knock on their door".

It is also reported that during code review, when the author submits the code, he/she must wait for timely feedback. After waiting for a long time, the request is rejected. Regarding waiting, they conveyed that it is a typical form of disregard and is interactional unfairness. One of the authors in their research discussing the waiting waste reported that "Contributions are ignored unless I beg for attention. I might have time to contribute a minor improvement. I never have time to beg for attention. If you don't want my help, I got the message loud and clear" [23].

Researchers have contributed towards facilitating such review engagement and waiting waste reduction with the aim of improving the feedback process, overall collaboration and effective knowledge sharing [6], [9], [24]. The author of this research study has developed the Modern Code Review-Knowledge Sharing Model (MCR-KSM) to reduce the waiting waste generation. The details regarding the development of MCR-KSM are presented in our previous research work [3], [15]. In our previous work, the MCR-KSM was developed. This study is an extension of our previous work. To develop the MCR-KSM, the included research methodologies were Systematic Literature Review (SLR), along with Expert Review and Delphi Survey. The results of previous work are available at [3], [15]. This study aims to conduct the experimental evaluation of the developed MCR-KSM. The objective of this study is "To evaluate the effectiveness of the Modern Code Review Knowledge Sharing Model to reduce software engineering waiting waste (waiting time)".

The rest of the paper is planned as follows: Section II deliberates the research background and literature review. The experiment design details, including objective, environment, hypothesis, variable, subject selection, instrument and validity evaluation, are given in Section III. The details regarding experiment execution are covered in Section IV. Section V provides the details about the data collection. Section VI highlights the results analysis and discussion. Sections VII-X highlight the conclusion, study limitations, future work suggestions and contribution.

II. Background

Software engineering is a methodical process that aims to create high-quality software within a specified budget and schedule [25]. It involves various underneath systematic activities to achieve its aims. The underneath activities included [26] requirement identification and management, modeling, development, dynamic testing, Static testing (inspection and code reviews). These activities produced numerous wastes [27]. The generated wastes may lead to numerous other issues such as psychological distress, delay in projects, overspending and software malfunctions. When Toyota revolutionized the industry with lean manufacturing, in the 1980's the idea of waste was first presented by [28], [29]. Later in 2000, the lean paradigm was shifted from lean manufacturing to lean software development [20]. Moreover, it is also contended that managing waste generation ensures green and sustainable software development [30]. Various researches were conducted with the aim to identify waste and procedures, tools and methodologies and a model to reduce the waste [18].

In the context of software engineering numerous wastes have been recognized, for instance, "rework", "defects", complexity, cognitive load, and waiting, etc. [18], [28]. As discussed previously, there are multiple underneath activities involved for successful software engineering; each activity produces waste [27]. Code review is an important activity that contributes towards code quality improvement and quality software development [4], [31], [12].

In code review, the reviewer evaluates the source code before uploading it to the version control repository. Code review is supported with the aid of numerous AI-based review tools, such as Quodo merge, Greptle, CodeRabbit, Gerrit, Corbit, Deepsource, PullApprove, for instance, Code flow, such as Gerrit, Review board, Phabricator, etc. [4], [22], [32], [33], [34],[35]. Fig. 1 represents the MCR process workflow.

The code review activity becomes challenging when the team members must confront various types of waste. For instance, Waiting waste, cognitive load, code duplication, poor feedback, negative emotions, poor review comments, Biases, lack of knowledge, poor code understanding, code complexity, ineffective code review tools, poor communication, confusion, lack of motivation to share knowledge, code inconsistency, etc. [13], [36], [37], [38], [39], [40], [41], [42], [43]. It is also reported that waiting waste is the critical waste, and it is conveyed that the waiting waste must be the organization's primary priority if it is to reduce any waste [19], [20].

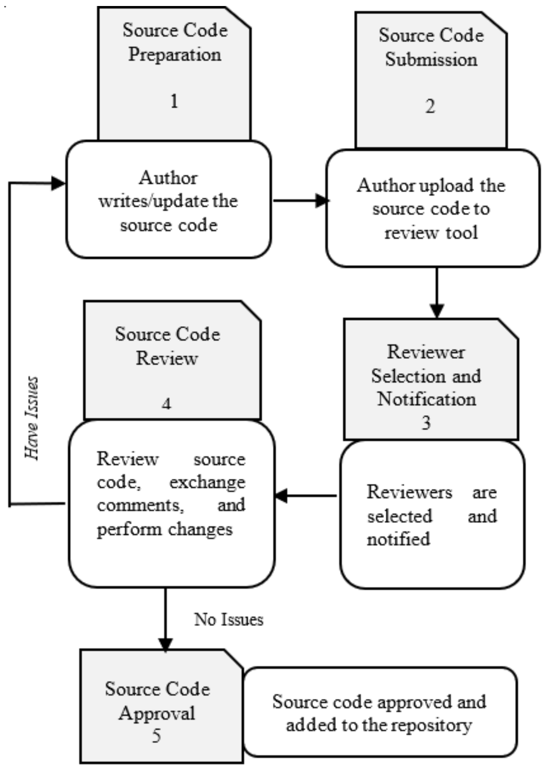
Source Code Preparation 1
Author writes/update the source code
Source Code Submission 2
Author upload the source code to review tool
Reviewer Selection and Notification 3
Reviewers are selected and notified
Source Code Review 4
Review source code, exchange comments, and perform changes
Have Issues
No Issues
Source Code Approval 5
Source code approved and added to the repository

Fig. 1. Workflow of the MCR process [32].

The MCR process allows the freedom for reviewers to accept or reject review requests. This is a critical practice that makes the MCR Process challenging and generates waiting waste. It is reported that as the author submits the code for review, generally the authors wait for the reviewers' response. Numerous research have been performed to highlight the reported issue. A study conducted using dataset of 182 GitHub projects included 55K pull request and 466K code change request comments, and poor clarity of code comments significantly contribute to the delay response [37]. Likewise, poor prioritization of code change requests can contribute to delays and an inefficient review process [14]. A study reported that "Reviewers find only shallow defects and it is hard to give insightful and actionable feedback in timely manner"[44].

Moreover, it is reported that when the author submits the code and request reviewer for review, they have to wait for long time. It became more impertinence when the request is rejected or unnoticed [23]. Likewise, it is conveyed that 16% to 66% of code submitted for review have at least one invited reviewer who let the author in waiting condition [45]. It was argued that regardless of the ease of the MCR process, waiting for feedback is a challenge for authors. It is claimed that multiple human factors such as reviewers' workload, reviewer experience, and familiarity with the requested review can cause poor knowledge sharing and waiting waste [45]. Similarly, it is stated that waiting is the highest reported challenge. In their study a survey participant stated that "Usually you write up some code and then you send it out for review, and then about a day later you ping them to remind them. and then about half a day later you go to their office and knock on their door." [22].

Moreover, It is reported that developers with less experience have to wait for the feedback [32], [46]. Similarly, it is stated that the broadcast or unicast review technology can have a noteworthy effect on delay in response. It is conveyed that broadcast technology has a short response delay as compared to unicast technology [47]. It is stated that there is a need to have guidelines for source code submission to avoid waiting waste [48]. Correspondingly, it is also reported that to get insightful feedback in timely manner is crucial, but the authors often suffer from the non-responsiveness of reviewers and must wait for feedback. The conveyed developer statement is "The value of the feedback is in the proportion to the cost in terms of delay and time spent" [49]. It is argued that developers' reputation, code size are the reason can have impact on timely feedback [50], [51].

Moreover, it is also argued that even in the presence of AI-based code review tools, the wastes are generated during code review [11]. The generated waste can be waiting, rework, defect, needless composite solutions, task switching, code complexity, etc. [28], [43]. It is conveyed that code creation is easy and fast "just prompt LLM to generate desired code", however it is difficult to employ AI-based code review tools to evaluate the code issues. It is reported that expert developers are more valuable than machines and are not replaceable by LLMs [11], [52], [53]. There must be documentation or guidelines to manage the waiting waste and thus project delays [11], [54].

Recent research conveyed that developers' domain knowledge, effective communication and knowledge sharing can reduce waiting waste and project delays [22], [27], [33], [38], [55], [56], [57], [58]. In our previous research work, we have developed the MCR-KSM to reduce waiting waste. In this study, we have presented an experimental study that was performed to validate the designed model with respect to the reduction in waiting waste generation. The upcoming section covers the detailed experiment design, data collection and results analysis.

III. Research Methodology

An experiment in a control environment has been conducted to validate the Modern Code Review Knowledge Sharing Model (MCR-KSM). The aim is to investigate whether the MCR-KSM helps to reduce waiting waste production. The guidelines provided by [59], [60] were utilized for the conduction of the experiment. Grounded on the research work of [61], the experiment was performed with the students of software engineering graduate level having same software development experience and educational background. To conduct the experiment the preparation regarding experiment objectives, selection of environment, hypothesis design, variables, and validity assessment must be completed. This section delivers the particulars regarding planning for the experiment. The experiment execution details are provided in result analysis section. The activities included in the experiment planning are given in sub-sections.

A. *Objective of Experiment Conduction*

The objective to conduct the experiment was to confirm whether the MCR-KSM help to reduce the waiting waste generation in real environment. The waiting waste production evaluated firstly without using the MCR-KSM, then the waiting waste production was assessed while using the MCR-KSM, assisted by electronic reference guideline of MCR-KSM.

B. *Experiment Environment*

The experiment environment discusses the context in which the experiment is executed [59]. The experiment was executed in the laboratory of computer science at Comwave Institute of Science and Information Technology, Islamabad, Pakistan. The systems having Windows operating system and C++ editor were utilized to conduct the experiment. The MCR-KSM was also accessible during the experiment.

C. *Hypothesis Construction*

The experimental statistical analysis is founded on hypothesis testing. A hypothesis is evaluated based on the analysis of dependent variables of the experiment. The guidelines given by [59] were utilized to construct the hypothesis. For experiment pre-test and post-test, the null and alternative hypothesis was constructed. The null hypothesis, H0 refers that there are no real underlying patterns in the experiment situation, the only reasons for differences in observations are coincidental. However, the alternate hypothesis, H1 is in support of which the null hypothesis is rejected.

1) Hypothesis for pre-test: For the pre-test Null and Alternate hypothesis were constructed and are presented in Table I.

TABLE I HYPOTHESIS FOR PRE-TEST

Hypothesis Types	Hypothesis Representation	Hypothesis
Null Hypothesis	H0	There is no noteworthy difference in waiting waste production for both the groups without using MCR-KSM.
Alternate Hypothesis	H1	There is a noteworthy difference in waiting waste production for both the groups without using MCR-KSM.

2) Hypothesis for post-test: For the post-test, hypotheses Null (H_0), Alternate (H_1, and H_2) were constructed. If the H_0 was rejected, it refers to dual situations. The first one was that the waiting waste production in the modern code review process using the MCR-KSM was less than the waiting waste production without using the MCR-KSM. The second situation was that the waiting waste production in the modern code review process using MCR-KSM was greater than the waiting waste production without using MCR-KSM. Therefore, two alternative hypotheses were constructed supporting each situation. Post-test hypothesis are given in Table II.

TABLE II HYPOTHESIS FOR POST-TEST

Hypothesis Types	Hypothesis Representation	Hypothesis
Null Hypothesis	H0	There is no significant difference in the waiting waste production in the modern code review process with or without using MCR-KSM.
Alternate Hypothesis	H1	The waiting waste production in modern code review process using MCR-KSM is lesser than without using MCR-KSM
	H2	The waiting waste generation in modern code review process using MCR-KSM is greater than without using MCR-KSM.

D. Experiment Variables

According to the guidelines provided by [59], independent and dependent variables can be defined for an experiment. In this study, one independent variable was "modern code review process". The process was manipulated with and without the support of MCR-KSM, and the dependent variables were assessed. The dependent variable was the "Waiting Waste". The waiting waste was measured using Value Stream Mapping (VSM) from three aspects, i.e. author waiting time, reviewer waiting time, and total waiting time during MCR activities.

The VSM is utilized by various researchers to calculate the waiting time in various situations. For instance, [29] utilized VSM to measure waiting waste by calculating the customer waiting time, development team waiting time, and total waiting time in the software customization process. Likewise, [20] utilized VSM to measure waiting waste in traditional and agile processes.

In this study, the waiting waste was analysed based on aspects such as "Author Waiting Time (AWT)", that is, waiting time observed by the author while executing MCR activities. "Reviewer Waiting Time (RWT)", that is, the waiting time observed by the reviewer while executing MCR activities and "Total Waiting Time (TWT)", that is, the waiting time observed by the author and reviewers during MCR.

E. Experiment Subject Selection

In this study, 28 students of software engineering were designated as test subjects according to the guidelines given by [62]. The subjects have equal experience. They were grouped as "Group I" and "Group II". Each group contained 14 subjects. Out of 14 subjects in both groups, 7 subjects were designated as "Author" and "Reviewer". The subjects were chosen depending on their industry experience and programming subjects studied during graduation.

F. Experiment Instrument

Formerly, in the experiment implementation, the instruments must be developed. It can be guidelines, objects, and measurement instruments for the measurement [59]. In this experimental study, the included instruments were problem statements that were used by the subjects to write the code, Modern Code Review Knowledge Sharing Model (MCR-KSM), along with an electronic reference guideline, and a document explaining the MCR activities. The details about experiment objective, problem and activities were specified to all the subjects. The subjects "Author" had to write the code, whereas the subjects "Reviewers" had to evaluate the code written by the authors of their group. The subjects of "Group II" were prearranged with the MCR-KSM, supported with its electronic reference guideline. The experiment data were collected via forms as per the directions of [59]. The waiting time experienced by subjects' "Author" and "Reviewers" was measured through the data collected during the experiment. The experiment moderator was given the task of managing the record of the start and end time of each MCR activity.

G. Validity Assessment

It strengthens the rationality of the experimental results. It involves recognition of validity threats, "factors that can influence the dependent variables that are not included as independent variables". The internal and external validity threats were considered as per the guidelines given by [59]. The internal validity threats influence the dependent variable deprived of the researcher's information [59]. The following internal validity threats were discussed for the experiment.

1) *Selection effect*: It is because of the usual dissimilarity in human recital. Each human can have a different understanding of the English language [59]. It was managed by the selection of subjects with a common educational background. Additionally, they were given similar details about the experiment.

2) *Learning effect*: It is because of the subjects' behaviour during the experiment activities. It was controlled by common training of the subjects about activities, problems, MCR and MCR-KSM.

3) *Instrumentation effect*: It is because of the artifacts, such as data collection forms, documents that need to be reviewed, or problems [59]. This hazard was overcome since every subject was given the same task to code, as [59] explained.

4) *Information exchange*: It is reported that the exchange of information while performing an experiment influences the outcome [59]. It was controlled by strictly observing the subjects during the experiment.

External validity is the external conditions which limit the ability to generalize the experimental results to industrial practice [59]. The two external validity threats, generalizability of subjects and Experiment Scale, were managed. The details about external validity threats are given in sub-sections.

5) *Generalizability of subject*: It occurs when the population is not taken from the industry [59]. In this study, the selected subjects have industry experience. Thus, this threat was managed as per [59] guideline.

6) *Experimental scale*: It occurs when the experimental situation or the materials are not illustrative of industrial practice. It was overcome by delivering the subject with real industry problems.

IV. Experiment Execution

The experiment was conducted in compliance with the experiment planning discussed in the previous section, and the actual data was collected and analysed. The experiment was performed in dual sessions. In "Session 1", "Group I" and "Group II" performed MCR activities given in Fig. 1 without the Modern Code Review Knowledge Sharing Model (MCR-KSM). In "Session 2", "Group II" was provided with MCR-KSM with its electronic reference guideline, while "Group I" was not given the model.

Prior to the experiment execution, 28 subjects were distributed into "Group I" and "Group II" as discussed in the design section. The subjects were also provided unique IDs. The "Group I" subjects were provided IDs as "A1 to A7" and "R1 to R7" for the author and the reviewer roles, respectively. Similarly, the "Group II" subjects were given IDs as "A8 to A14" and "R8 to R14". The subjects of both groups with the author role were given a problem statement in both experiment sessions for code writing, whereas the subjects' reviewer role, both groups were given the job to review the code written by the authors of their respective group. The subjects were given 1 hour of training regarding experiment objectives, experiment process, data collection forms, activities, and MCR process. The subjects of "Group II" were described in the electronic guide of the MCR-KSM. In "Session I" of the experiment, no group was given access to MCR-KSM. The "Authors" from "Group I" and "Group II" were requested to write the code in parallel. After finishing the coding activity, they were directed to submit the code to the moderator. After gathering the code from the first subject, "Author", the moderator gave the code to the subject "Reviewer" with the first ID of the same group to review the code. Like this, the "Author-Reviewer" sub-group was established. The "Reviewers" were then asked to review the allocated code and give feedback to the respective "Author". The subject "Author", after receiving the feedback from the subject "Reviewer", was requested to make corrections as suggested by the reviewer. At that moment, the "Author" and "Reviewer" can exchange comments for clarification purposes. The "Author" resubmitted the code to the respective reviewer after addressing the issues. This cycle continues till the chosen reviewer accepts the code. If the reviewer is satisfied with the code, then the code is signed off and formally accepted by the reviewer. During experiment execution, the moderator recorded the waiting time faced by the "Author" or "Reviewer". The data collection details are given in Section V.

V. Data Collection

Based on the experiment design deliberated in Section III, the data were collected. In Session I of the experiment, "Author" from "Group I" and "Group II" were requested to write the code without using MCR-KSM. Similarly, the "Reviewer" from "Group I" and "Group II" were asked to review the code and provide feedback without using MCR-KSM.

During Session II, the subjects "Author" of "Group I" were asked to write the code without using MCR-KSM, and subjects "Authors" of "Group II" were asked to write the code using the MCR-KSM. Likewise, the subjects "Reviewer" of "Group I" were asked to review the code and provide feedback without using MCR-KSM and the subjects "Reviewer" of "Group II" were asked to review the code and provide feedback using MCR-KSM. For both sessions, the waiting time experienced by authors and the reviewers was recorded, and the total waiting time was computed. The collective waiting times for both sessions are given in Tables III and IV. The waiting time is given in Table III, calculated in the experiment session I for the "Groups I" and "Group II" when they executed MCR activities without using MCR-KSM. The waiting time presented in Table IV was computed in the experiment session II when "Group I" performed MCR activities without using the MCR-KSM and 'Group II' performed MCR activities using the MCR-KSM.

Tables III and IV have dual main columns, "Group I" & "Group II". Each main column has four sub-columns. The "Test Subject Sub-group ID" column shows the sub-group ID, the "Group Members" column shows the group members, "Author Waiting Time" column shows the waiting time faced by the author. The "Reviewer Waiting Time" column shows the waiting time faced by the reviewer. "Total Waiting Time" is the collective waiting time faced by the author and reviewer. The result analysis based on the data collected is discussed in Section VI.

TABLE III SESSION I EVALUATION OF WAITING WASTE WITHOUT USING MCR-KSM (PRE-TEST)

Group I					Group II				
Test Subjects Sub-group (sg) ID	Group Members	Author Waiting time	Reviewer Waiting time	Total Waiting Time (min)	Test Subjects Sub-group (sg) ID	Group Members	Author Waiting time	Reviewer Waiting time	Total Waiting Time (min)
TSG-I-sg1	A3, R1	40	25.00	60	TSG-II-sg1	A8, R9	42	20	62
TSG-I-sg2	A4, R2	45	35.00	80	TSG-II-sg2	A10, R12	45	45	89
TSG-I-sg3	A2, R3	40	35.00	75	TSG-II-sg3	A9, R10	45	35	90
TSG-I-sg4	A5, R4	55	23.00	78	TSG-II-sg4	A13, R13	45	20	65
TSG-I-sg5	A1, R5	42	32.00	69	TSG-II-sg5	A12, R14	33	43	66
TSG-I-sg6	A6, R6	40	21.00	60	TSG-II-sg6	A1, R11	37	22	59
TSG-I-sg7	A7, R7	39	40.00	79	TSG-II-sg7	A11, R18	40	35	75

VI. Results and Discussion

This section provides the details about the result analysis performed based on the data collected and presented in Section V. The waiting waste, i.e. waiting time was calculated utilizing value stream mapping technique based on variables, i.e. "Author Waiting Time (AWT)", "Reviewer Waiting Time (RWT)", and "Total Waiting Time (TWT)". The collected data is given in Tables III and IV. The experiment data was recorded in SPSS for data analysis. To measure the waiting waste generation in "Pre-test" paired sampled t-test was used [59]. Waiting waste was calculated with three perspectives, i.e. waiting time faced by subject "Authors", subject "Reviewers", and "Total Waiting Time". The mean differences between the dependent variables for examining the waiting waste generation were analyzed. The result analysis of the dependent variables is provided in sub-sections.

1) Pre-test result analysis: By using "paired t-test" the waiting waste production in Pre-test for "Group I" and "Group II was analysed. The subjects were provided with a pre-test where none of the group were given MCR-KSM. Table V shows the stats of the paired t-test for both the groups. The result analysis of the paired t-test shows that there was no considerable difference amongst Group I and Group II who completed experimental activities in pre-test.

The mean values for "Author Waiting Time (Group I, mean=43.00, standard deviation =5.66 Group II, mean =41.00, standard deviation=4.65), Reviewer Waiting Time (Group I, mean=30.1429, standard deviation=7.17469 Group II, mean =31.4286, standard deviation=10.75263) and Total Waiting Time (Group I, mean=71.5, standard deviation=8.695 Group II, mean =72.2, standard deviation=12.75035)" shows that there is no considerable difference in waiting waste production (Author Waiting Time, Reviewer Waiting Time, and Total Waiting Time) for both the groups. The p-value (sig, (2-tailed)) was (p>0.05) for author waiting time (p=0.386), reviewer waiting time (p=0.630), and total waiting time (p=0.843). If p>0.05 it infers that the null hypothesis was accepted as there was no substantial difference between author waiting time, reviewer waiting time, and total waiting time of "Group I" and "Group II" in Pre-test. It was analyzed from the paired t-test results with variables author waiting time, reviewer waiting time, and total waiting time that there was no significant difference in waiting waste generation when both the group were not provided with Modern Code Review Knowledge Sharing Model. The author waiting time of both groups was almost the same ("Group I", mean=43, "Group II", mean=41). The measure of 2-tailed t-test found this difference not to be significant, t (7) = 0.935, p>0.386. Hence, it was determined that

"The author waiting time was almost same when both the groups were not provided with modern code review knowledge sharing model".

TABLE IV SESSION II EVALUATION OF WAITING WASTE WITHOUT AND WITH USING MCR-KSM (POST-TEST)

Group I (Without (WOT) using modern code review knowledge sharing model)					**Group II (With (WT) using modern code review knowledge sharing model)**				
Test Subjects Sub-group (sg) ID	Group Members	Author Waiting time	Reviewer Waiting time	Total Waiting Time (min)	Participants Sub-group (sg) ID	Group Members	Author Waiting time	Reviewer Waiting time	Total Waiting Time (min)
TSG-I-sg1	A3, R1	37	19	56	TSG-II-sg1	A11, R8	25	15	40
TSG-I-sg2	A4, R2	45	35	80	TSG-II-sg2	A10, R9	20	13	33
TSG-I-sg3	A2, R3	40	35	75	TSG-II-sg3	A9, R10	25	15	40
TSG-I-sg4	A5, R4	55	25	80	TSG-II-sg4	A8, R11	32	16	48
TSG-I-sg5	A1, R5	40	26	66	TSG-II-sg5	A14, R12	30	14	44
TSG-I-sg6	A6, R6	39	20	59	TSG-II-sg6	A12, R13	20	17	37
TSG-I-sg7	A7, R7	36	35	71	TSG-II-sg7	A13, R14	20	10	30

TABLE V WAITING WASTE ANALYSIS PRE-TEST – EXPERIMENT SESSION I

Variables	**Mean**	**N**	**t**	**Sig. (2-tailed)**
Author Waiting Time (AWT)	43.00	7	0.935	0.386
	41.00	7		
Reviewer Waiting Time (RWT)	30.14	7	-.508	0.630
	31.4286	7		
Total Waiting Time (TWT)	71.5	7	-.207	0.843
	72.2	7		

Similarly, the reviewer waiting time of both the groups was almost the same ("Group I", mean=30.14, "Group II", mean=31.4). The measure of 2-tailed t-test found this difference not to be noteworthy, t (7) = -0.508, p>0.630. Hence, it was concluded that

"The reviewer waiting time was almost same when both the groups were not provided with modern code review knowledge sharing model"

Likewise, the total waiting time of both the groups was almost the same ("Group I", mean=71.5, "Group II", mean=72.2). The measure of 2-tailed t-test found this difference not to be momentous, t (7) = -0.207, p>0.843. Hence, it was concluded that

"The total waiting time was almost same when both the groups were not provided with modern code review knowledge sharing model".

Based on the independent results of dependent variable "author waiting time", "reviewer waiting time", and total waiting time it was concluded that there was no noteworthy difference in the waiting waste generation when both the groups were not provided with the Modern Code Review Knowledge Sharing Model, therefore the null hypothesis, H0, i.e. "There is no noteworthy difference in waiting waste generation for both the groups without using modern code review" was accepted. The paired sample t-test results rejected the alternate hypothesis, H2, i.e. "There is a noteworthy difference in waiting waste generation for both the groups without using modern code review." It was analysed from the results analysis of Group 1 and Group II pre-test that the variance between "Group I" and Group II" was not noteworthy that signifies both the groups were equal regarding their programming capability.

2) Post-test result analysis: The paired t-test was performed to analyze the waiting waste generation for "Group I" and "Group II" in the post-test. The subjects were provided the post-test in which "Group I" was not offered with MCR-KSM and "Group II" was provided with MCR-KSM. Table VI shows the stats of the paired t-test for "Group I" and "Group II" for variables "Author Waiting Time", "Reviewer Waiting Time", and "Total Waiting time".

The stats of paired t-test exhibited that there was a noteworthy variance between "Group I" and "Group II". The mean values for Author Waiting Time "Group I, mean = 41.7143, standard deviation = 6.52468. Group II, mean = 24.5714, standard deviation = 4.96176". Reviewer Waiting Time "Group I, mean = 27.0571, standard deviation = 7.12808 Group II, mean = 14.2857, standard deviation = 2.28869" and Total Waiting Time "Group I, mean = 69.5714, standard deviation = 9.64118 Group II, mean = 38.8571, standard deviation = 6.17599" exhibited a noteworthy difference in waiting waste generation for both the groups.

TABLE VI WAITING WASTE ANALYSIS POST-TEST – EXPERIMENT SESSION II

Group ID	**Variables**	**Mean**	**N**	**t**	**Sig. (2-tailed)**
Group I	Author Waiting Time (AWT)	41.7143	7	8.216	0.000
Group II		24.5714	7		
Group I	Reviewer Waiting Time (RWT)	27.0571	7	4.058	0.007
Group II		14.2857	7		
Group I	Total Waiting Time (TWT)	69.5714	7	7.223	0.000
Group II		38.85	7		

Therefore, it was analysed that there was a noteworthy difference regarding waiting waste generation "Author Waiting Time, Reviewer Waiting Time, and Total Waiting Time" amongst "Group I" and "Group II". The p-value (sig, (2-tailed)) was (p<0.05) for author waiting time (p=0.00), reviewer waiting time (p=0.007), and total waiting time (p=0.000). The p value i.e. p<0.05 it infers that the null hypothesis "There is no difference in the waiting waste generation in the modern code review process with or without using Modern Code Review Knowledge Sharing Model" was rejected and there was a noteworthy difference between author waiting time, reviewer waiting time, and total waiting time of "Group I" and "Group II". The positives t values for Author Waiting Time (t=8.216), Reviewer Waiting Time (t=4.058), and Total Waiting Time (t=7.223) indicate that the Modern Code Review Knowledge Sharing Model (MCR-KSM) reduces the author waiting time, reviewer waiting time, and total waiting time. From the paired t-test results it was obvious that that there was a noteworthy decrease in waiting waste generation when participants considered the MCR-KSM.

The waiting time of the "Group II" author was less when they performed MCR activities while using MCR-KSM (mean = 24.5714, standard deviation = 4.96176) as compared to "Group I" author who did not use MCR-KSM (mean = 41.7143, standard deviation = 6.52468). Measure of 2-tailed t-test found this difference to be noteworthy, t (7) = 8.216, p<0.05. Hence, it was determined that

"The author waiting time was reduced with the use of modern code review knowledge sharing model".

Fig. 2 shows the comparative view of Author Waiting Time with and without using MCR-KSM during Post-test session. It clearly shows that when code review was performed using MCR-KSM the average Authors Waiting Time (AWT) was significantly less, i.e. 24.5 (min) as compared to code review activities when performed without using MCR-KSM where average AWT was 41.7 (min).

Likewise, waiting time of the "Group II" reviewer was less when they performed MCR activities while using MCR-KSM (mean = 27.0571, standard deviation = 7.12808) as compared to "Group I" who did not MCR-KSM (mean = 14.2857, standard deviation = 2.28869). The measure of 2-tailed t-test found this difference to be significant, t (7) = 4.058, $p<0.05$. Hence, it was determined that

"The reviewer waiting time was reduced with the use of modern code review knowledge sharing model".

Fig. 3 shows the comparative view of Reviewer Waiting Time with and without using MCR-KSM during Post-test session. It evidently shows that when code review was performed using MCR-KSM the average Reviewer Waiting Time (RWT) was significantly less, i.e. 14.2 (min) as compared to code review activities when performed without using MCR-KSM where average RWT was 27.0 (min).

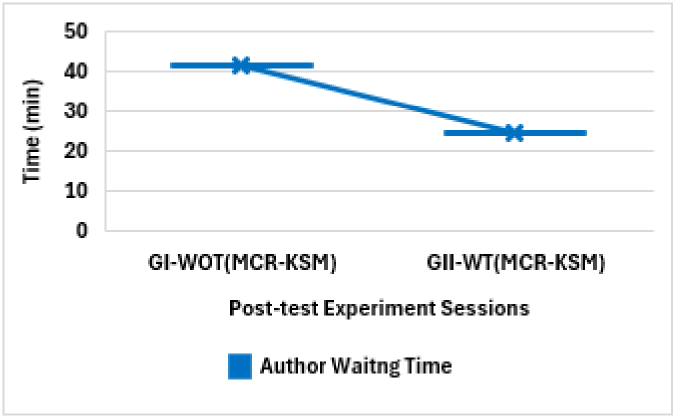

Fig. 2. Comparative view of author waiting time (AWT)-post-test.

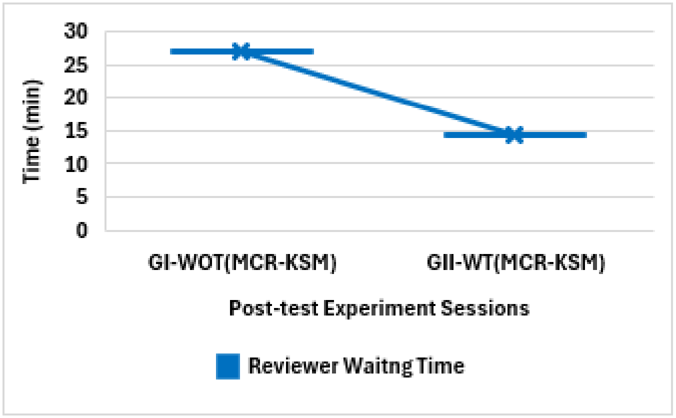

Fig. 3. Comparative view of reviewer waiting time (RWT)-post-test.

Equally, total waiting time was less for "Group II" when the subjects performed MCR activities while using MCR-KSM (mean = 38.8571, standard deviation = 6.17599) as compared "Group I" to when they did not use the MCR-KSM (mean = 69.5714, standard deviation = 9.64118). The measure of 2-tailed t-test found this difference to be noteworthy, t (7) = 7.223, $p<0.05$. Therefore, it was determined that

"the total waiting time was reduced with the use of modern code review knowledge sharing model".

Fig. 4 shows the comparative view of Total Waiting Time with and without using MCR-KSM during Post-test session. It clearly shows that when code review was performed using MCR-KSM the average Total Waiting Time (TWT) was significantly less i.e.38.8 (min) as compared to code review activities when performed without using MCR-KSM where average TWT was 69.5 (min).

Based on the independent results of dependent variable "Author Waiting Time", "Reviewer Waiting Time", and "Total Waiting Time", it was determined that there was a noteworthy variance in the waiting waste generation with and without the support of the MCR-KSM, so the null hypothesis, H0, i.e. "There is no difference in the waiting waste generation in the modern code review process with or without using Modern Code Review Knowledge Sharing Model" was rejected. The paired sample t-test results supported the alternate hypothesis, H1, i.e.

"The waiting waste generation in MCR using MCR-KSM is lower than without using MCR-KSM.".

The alternate hypothesis H_2 given in Table II was not supported by the paired sample t-test.

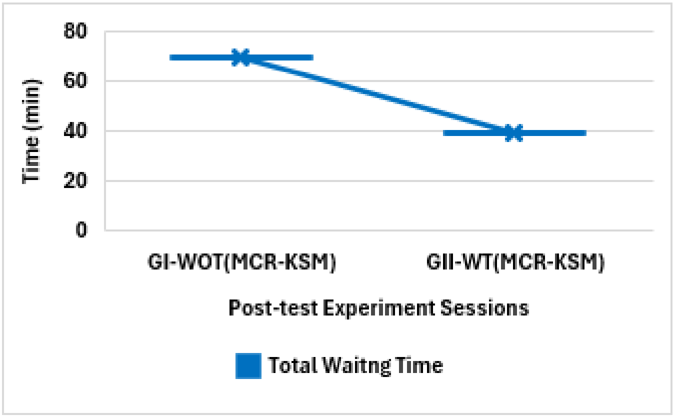

Fig. 4. Comparative view of waiting waste generation-post-test.

VII. Conclusion

MCR is a quality assurance activity, though it is supported with AI and LLM, but it generates various wastes such as rework, defect waste, composite solution waste and waiting waste, etc. Waiting waste is the critical one that causes mental distress. Reducing waiting waste is crucial for sustainable software engineering. The Knowledge exchange has a profound role in the reduction of waiting waste production during code reviews. To reduce waiting waste, our previous study introduced the Modern Code Review Knowledge Sharing Model (MCR-KSM). This study presented empirical results to evaluate the effectiveness of MCR-KSM for the reduction in waste generation during the MCR process. The waiting waste was calculated based on variables, i.e. "Author Waiting Time" and "Reviewer Waiting Time". The study results show that there was a significant reduction in the authors, reviewers and total waiting time when they considered the Modern Code Review Knowledge Sharing Model.

VIII. Limitations

To conduct the study, it was hard to gather dedicated respondents. Twenty-eight graduate students with software engineering experience and knowledge of MCR, software engineering wastes, took part in the experiment. The results might be more confident if more students contributed to the experiment. Nevertheless, despite the limitations, we believe that the results have educational and practical implications.

IX. Future Work Opportunities

The experiment can be augmented in future with varying settings. In future, it is planned to evaluate the significance of MCR-KSM in the actual industry environment. Likewise, more experiments can be conducted to check whether the developed model can reduce other waste, such as defect waste and motion waste, negative emotion waste, etc. and emotional intelligence. Moreover, innovation in AI, machine learning, and LLM-based code review is evolving, and it is understood that developers' domain knowledge is essential and cannot be ignored. Therefore, solutions combined with AI, machine learning and developers' domain knowledge can be beneficial for the reduction of various types of code review wastes. Moreover, other software engineering activities or computing domains can be explored for waste identification and reduction.

X. Contribution

The research conducted has created the foundation for green and sustainable computing by developing and evaluating the MCR knowledge sharing model to reduce waste. The study contributed to three aspects. Firstly, it confirms that the MCR-KSM supports in reduction of waiting waste production. Secondly, the research article provided a detailed experiment design and conduct procedure that can be beneficial for young researchers to experiment with their respective research field. Thirdly, the research highlighted future research venues that can be beneficial for practitioners and researchers.

References

[1] S. Nazir, N. Fatima, and S. Chuprat, "Modern code review benefits-primary findings of a systematic literature review," in ACM International Conference Proceeding Series, 2020, pp. 210–215.

[2] T. Maikantis, I. Natsiou, A. Ampatzoglou, A. Chatzigeorgiou, S. Xinogalos, and N. Mittas, "What you See is What you Get: Exploring the Relation between Code Aesthetics and Code Quality," in Proceedings - 2024 ACM/IEEE International Conference on Technical Debt, TechDebt 2024, 2024, pp. 1–10.

[3] N. Fatima, S. Nazir, and S. Chuprat, "Knowledge sharing factors for modern code review to minimize software engineering waste," Int. J. Adv. Comput. Sci. Appl., vol. 11, no. 1, pp. 490–497, 2020.

[4] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in Proc. International Conference on Software Engineering, 2013, pp. 712–721.

[5] M. Caulo, B. Lin, G. Bavota, G. Scanniello, and M. Lanza, "Knowledge transfer in modern code review," in IEEE International Conference on Program Comprehension, 2020, pp. 230–240.

[6] A. Bouraffa, Y. D. Pham, and W. Maalej, "How do Developers Use Code Suggestions in Pull Request Reviews?," 2025, pp. 227–238.

[7] M. T. Rahman, R. Singh, and M. Y. Sultan, "Automating Patch Set Generation from Code Reviews Using Large Language Models," in Proceedings - 2024 IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI, CAIN 2024, 2024, pp. 273–274.

[8] S. Ahmed and N. U. Eisty, "Exploring the Advances in Identifying Useful Code Review Comments," in International Symposium on Empirical Software Engineering and Measurement, 2023.

[9] T. Jetzen, X. Devroey, N. Matton, and B. Vanderose, "Towards Debiasing Code Review Support," 2025, pp. 143–148.

[10] N. Davila, J. Melegati, and I. Wiese, "Tales From the Trenches: Expectations and Challenges From Practice for Code Review in the Generative AI Era," IEEE Softw., vol. 41, no. 6, pp. 38–45, 2024.

[11] F. Kazemi, M. Lamothe, and S. Mcintosh, "Interrogative Comments Posed by Review Comment Generators: An Empirical Study of Gerrit," in International Symposium on Empirical Software Engineering and Measurement (ESEM 2025), 2025.

[12] S. Nazir, N. Fatima, and S. Chuprat, "Situational factors for modern code review to support software engineers' sustainability," Int. J. Adv. Comput. Sci. Appl., vol. 11, no. 1, pp. 498–504, 2020.

[13] C. S. Fatima, Nargis, Nazir, Suimaira, "Software Engineering Wastes – A Perspective of Modern Code Review," in The 3rd International Conference on Software Engineering and Information Management (ICSIM), 2020.

[14] L. Yang et al., "Prioritizing code review requests to improve review efficiency: a simulation study," Empir. Softw. Eng., vol. 30, no. 1, 2025.

[15] N. Fatima, S. Nazir, and S. Chuprat, "Knowledge sharing framework for modern code review to diminish software engineering waste," Int. J. Adv. Comput. Sci. Appl., vol. 11, no. 6, pp. 442–450, 2020.

[16] M. S. Rahman, Z. Codabux, and C. K. Roy, "Investigating the Understandability of Review Comments on Code Change Requests," 2025 IEEE/ACM 22nd Int. Conf. Min. Softw. Repos., pp. 539–551, 2025.

[17] Z. Yang et al., "A Survey on Modern Code Review: Progresses, Challenges and Opportunities," vol. 1, no. 1, pp. 1–62, 2024.

[18] H. Alahyari, T. Gorschek, and R. Berntsson, "An exploratory study of waste in software development organizations using agile or lean approaches: A multiple case study at 14 organizations," Inf. Softw. Technol., vol. 105, no. 7, pp. 78–94, 2019.

[19] J. Urrego, R. Munoz, M. Mercado, and D. Correal, "Archinotes: A global agile architecture design approach," Lect. Notes Bus. Inf. Process., vol. 179 LNBIP, pp. 302–311, 2014.

[20] M. Poppendieck and T. Poppendieck, Lean software development: An agile toolkit. 2003.

[21] N. Davila, J. Melegati, and I. Wiese, "Tales from the Trenches: Expectations and Challenges from Practice for Code Review in the Generative AI Era," IEEE Softw., vol. PP, pp. 1–8, 2024.

[22] L. MacLeod, M. Greiler, M. A. Storey, C. Bird, and J. Czerwonka, "Code reviewing in the trenches: Challenges and best practices," IEEE Softw., vol. 35, no. 4, pp. 34–42, 2018.

[23] D. M. German, U. Rey, and J. Carlos, "' Was my contribution fairly reviewed ?' A Framework to Study the Perception of Fairness in Modern Code Reviews," in Proc. ACM/IEEE 40th International Conference on Software Engineering Synthesizing, 2018, no. 2, pp. 523–534.

[24] C. Adapa, S. S. Avulamanda, A. R. K. Anjana, and A. Victor, "AI-Powered Code Review Assistant for Streamlining Pull Request Merging," in Proceedings of ICWITE 2024: IEEE International Conference for Women in Innovation, Technology and Entrepreneurship, 2024, pp. 323–327.

[25] P. Bourque and R. E. Fairley, Guide to the software engineering - Body of knowledge. 2014.

[26] S. Nazir, N. Fatima, and S. Chuprat, "Individual Sustainability Barriers and Mitigation Strategies: Systematic Literature Review Protocol," in 2019 IEEE Conference on Open System, ICOS 2019, 2019, pp. 1–5.

[27] T. Sedano and P. Ralph, "Software Development Waste," in Proc. IEEE/ACM 39th International Conference on Software Engineering, 2017.

[28] N. Fatima, S. Nazir, and S. Chuprat, "Software engineering wastes-A perspective of modern code review," ACM Int. Conf. Proceeding Ser., pp. 93–99, 2020.

[29] S. Mujtaba, R. Feldt, and K. Petersen, "Waste and lead time reduction in a software product customization process with value stream maps," Proc. Aust. Softw. Eng. Conf. ASWEC, pp. 139–148, 2010.

[30] S. R. Ahmad Ibrahim, J. Yahaya, and H. Sallehudin, "Green Software Process Factors: A Qualitative Study," Sustain., vol. 14, no. 18, 2022.

[31] N. Fatima, S. Nazir, and S. Chuprat, "Individual, Social and Personnel Factors Influencing Modern Code Review Process," 2019 IEEE Conf. Open Syst. ICOS 2019, pp. 40–45, 2019.

[32] A. Bosu, J. C. Carver, C. Bird, J. Orbeck, and C. Chockley, "Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at Microsoft," IEEE Trans. Softw. Eng., vol. 43, no. 1, pp. 56–75, 2017.

[33] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern code review: : A case study at google," in Proc. ACM/IEEE 40th International Conference on Software Engineering: Software Engineering in Practice, 2018, pp. 181–190.

[34] N. Fatima, S. Chuprat, and S. Nazir, "Challenges and Benefits of Modern Code Review-Systematic Literature Review Protocol," in Proc. International Conference on Smart Computing and Electronic Enterprise, 2018, pp. 1–5.

[35] P. J. S. Xuan and Y. Lee, "Automated Code Review and Bug Detection," in 19th International Joint Symposium on Artificial Intelligence and Natural Language Processing, iSAI-NLP 2024, 2024.

[36] Z. Zhang and T. Saber, "Machine Learning Approaches to Code Similarity Measurement: A Systematic Review," IEEE Access, vol. 13. Institute of Electrical and Electronics Engineers Inc., pp. 51729–51764, 2025.

[37] M. S. Rahman, Z. Codabux, and C. K. Roy, "Investigating the Understandability of Review Comments on Code Change Requests," 2025, pp. 539–551.

[38] S. Zamir, A. Rehman, H. Mohsin, E. Zamir, A. Abbas, and F. A. M. Al-Yarimi, "Integrating Pull Request Comment Analysis and Developer Profiles for Expertise-Based Recommendations in Global Software Development," IEEE Access, vol. 13, pp. 16637–16648, 2025.

[39] M. S. S. Chowdhury, M. N. U. R. Chowdhury, F. F. Neha, and A. Haque, "AI-Powered Code Reviews: Leveraging Large Language Models," in IEEE International Conference on Signal Processing and Advance Research in Computing, SPARC 2024, 2024.

[40] G. Rong et al., "Code Comment Inconsistency Detection and Rectification Using a Large Language Model," 2025, pp. 1832–1843.

[41] O. Ben Sghaier, M. Weyssow, and H. Sahraoui, "Harnessing Large Language Models for Curated Code Reviews," 2025, pp. 187–198.

[42] C. Adapa, S. S. Avulamanda, A. R. K. Anjana, and A. Victor, "AI-Powered Code Review Assistant for Streamlining Pull Request Merging," Proc. ICWITE 2024 IEEE Int. Conf. Women Innov. Technol. Entrep., no. Icwite, pp. 323–327, 2024.

[43] S. Kansab, M. Sayagh, F. Bordeleau, and A. Tizghadam, "An Empirical Study on the Amount of Changes Required for Merge Request Acceptance," 2025.

[44] L. Dong et al., "Survey on Pains and Best Practices of Code Review," in Proceedings - Asia-Pacific Software Engineering Conference, APSEC, 2021, vol. 2021-December, pp. 482–491.

[45] S. Ruangwan, P. Thongtanunam, A. Ihara, and K. Matsumoto, "The impact of human factors on the participation decision of reviewers in modern code review," Empir. Softw. Eng. Manuscr., pp. 1–43, 2018.

[46] A. Lee and J. C. Carver, "Are One-Time Contributors Different? A Comparison to Core and Periphery Developers in FLOSS Repositories," Int. Symp. Empir. Softw. Eng. Meas., vol. 2017-Novem, pp. 1–10, 2017.

[47] F. Armstrong, F. Khomh, and B. Adams, "Broadcast vs. unicast review technology: Does it matter?," in in Proc. 10th IEEE International Conference on Software Testing, Verification and Validation, 2017, pp. 219–229.

[48] A. Lee, J. C. Carver, and A. Bosu, "Understanding the impressions, motivations, and barriers of one time code contributors to FLOSS projects: A survey," in in Proce. IEEE/ACM 39th International Conference on Software Engineering, 2017, pp. 187–197.

[49] O. Kononenko, O. Baysal, and M. W. Godfrey, "Code review quality: How developers see it," in Proc. International Conference on Software Engineering, 2016, pp. 1028–1038.

[50] A. Bosu and J. C. Carver, "Impact of developer reputation on code review outcomes in OSS projects," Proc. 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 1–10, 2014.

[51] A. Bosu, "Modeling modern code review practices in open source software development organizations," in Proc. IDoESE '13 Baltimore, 2013.

[52] G. Foster, "Why AI will never replace human code review," Graphite, 2025. .

[53] M. Vijayvergiya et al., "AI-Assisted Assessment of Coding Practices in Modern Code Review," AIware 2024 - Proc. 1st ACM Int. Conf. AI-Powered Software, Co-located with ESEC/FSE 2024, pp. 85–93, 2024.

[54] Y. B. Alebachew, M. Ko, and C. Brown, "Are We on the Same Page? Examining Developer Perception Alignment in Open Source Code Reviews," Ease'25, 2025.

[55] T. Sedano, "Removing Software Development Waste to Improve Productivity," in Rethinking Productivity in Software Engineering, Apress, 2019, pp. 221–240.

[56] I. Vlachos, E. Siachou, and E. Langwallner, "A perspective on knowledge sharing and lean management: an empirical investigation," Knowl. Manag. Res. Pract., vol. 00, no. 00, pp. 1–16, 2019.

[57] M. Caulo, B. Lin, G. Bavota, G. Scanniello, and M. Lanza, "Knowledge transfer in modern code review," IEEE Int. Conf. Progr. Compr., pp. 230–240, 2020.

[58] A. Serebrenik and A. Bacchelli, "Competencies for Code Review," vol. 7, no. April, pp. 1–33, 2023.

[59] C. Wohlin, P. Runeson, M. Ohlsson C., B. Regnell, and A. Wesslen, Experimentation in software engineering. Springer, 2000.

[60] Kitchenham, S. L. Pfleeger, D. C. Jones, P.W.Hoaglin, K. El Emam, and J. B.A.Rosenberg, "Preliminary guidelines for empirical research in software engineering," 2002.

[61] M. Host, B. Regnell, and C. Wohlin, "Using students as subjects — A comparative study of students and professionals in lead-time impact assessment," Empir. Softw. Eng., vol. 5, pp. 201–214, 2000.

[62] J. C. F. de Winter, "Using the student's t-test with extremely small sample sizes," Pract. Assessment, Res. Eval., vol. 18, no. 10, pp. 1–12, 2013.