Knowledge Sharing Factors for Modern Code Review to Minimize Software Engineering Waste

Nargis Fatima[1], Sumaira Nazir[2], Suriayati Chuprat[3]
Razak Faculty of Technology and Informatics, University Technology Malaysia (UTM), Kuala Lumpur, Malaysia [1, 2, 3]
Department of Engineering and Computer Science, National University of Modern Languages, (NUML), Islamabad, Pakistan[1, 2]

***Abstract*—Software engineering activities, for instance, Modern Code Review (MCR) produce quality software by identifying the defects from the code. It involves social coding and provides ample opportunities to share knowledge among MCR team members. However, the MCR team is confronted with the issue of waiting waste due to poor knowledge sharing among MCR team members. As a result, it delays the project delays and increases mental distress. To minimize the waiting waste, this study aims to identify knowledge sharing factors that impact knowledge sharing in MCR. The methodology employed for this study is a systematic literature review to identify knowledge sharing factors, data coding with continual comparison and memoing techniques of grounded theory to produce a unique and categorized list of factors influencing knowledge sharing. The identified factors were then assessed through expert panel for its naming, expressions, and categorization. The study finding reported 22 factors grouped into 5 broad categories i.e. Individual, Team, Social, Facility conditions, and Artifact. The study is useful for researchers to extend the research and for the MCR team to consider these factors to enhance knowledge sharing and to minimize waiting waste.**

Keywords*—Knowledge sharing; modern code review; software engineering waiting waste

I. Introduction

Software engineering is a socio-technical activity for the development of software with specified resources [1]. It includes activities such as requirement identification, modeling, construction, testing, and Modern Code Review (MCR) [2]. These activities produce various wastes such as waiting, development of extra or erroneous feature, defect, needless composite solution, rework, and mental distress [2], [3], [4], [5]. In software engineering, waste can be defined as “anything that doesn’t make it to the release, is waste” [4].

Modern Code Review, a lightweight form of traditional Fagan’s code inspection [6], has been expanding in the research. A Fagan examination is a heavyweight code inspection procedure requiring synchronous interactions among the members in multiple stages [7]. On the other hand, MCR is characterized as being trivial, increasingly casual, and strengthened by review tools [6], [8], [9]. Notwithstanding studies that confirm Fagan’s code inspections advances the quality of software [7], [10] their required cost and formality have prohibited widespread acceptance [6], [8], [9]. Contrariwise, MCR has addressed many inadequacies of Fagan’s code inspection and highly adopted in industry and open-source software development contexts [6], [8], [9], [11].

Although MCR has addressed many shortcomings of Fagan’s code inspections and is developed to improve software and code quality through extensive knowledge sharing among MCR team members [6], [8], [9], [11], [12], however, the MCR generates waiting waste due to poor knowledge sharing [5], [8], [13], [14], [15], [16], [17], [18].

Current researchers [8], [9], [17] have shown that MCR team members are hesitant to share knowledge and give a timely response to other members and let them in a waiting condition. It is argued that waiting waste can be minimized by increasing knowledge sharing [2], [4], [5], [19] among the MCR team. It is also argued that knowledge sharing can be increased by identifying the factors influencing knowledge sharing [8], [9], [11], [20], [21] that can increase knowledge sharing among the MCR team that might lead to the reduction in the production of waiting waste in MCR.

Although previous research has given attention to knowledge sharing concerning software engineering activities [22], [23], [24], [25], however, knowledge sharing in the context of MCR has not got much attention from the researchers [8], [9], [11], [20], [21]. No, systematize investigations are available concerning the knowledge sharing aspect in MCR that can help in minimizing waiting waste. Therefore, the purpose of this study is to perform a Systematic Literature Review (SLR) to produce a validated and unique list of factors influencing knowledge sharing in MCR to minimize waiting waste.

The rest of this paper is distributed as Section II describes the background and related work. Section III covers the search method while Section IV introduces the results of SLR and expert review. Section V provides the discussion; Section VI presents the limitation of the study. Section VII presents the conclusion. Section VIII provides future directions. Section IX highlights the contribution of the study.

II. Background

Software engineering is a development of quality software within a stated time and budget [1]. The success factor of software engineering is subject to whether the software can fulfill user demands [1]. Software engineering is a socio-technical activity that incorporates managing other activities [2], [5] such as requirement identification, modeling, construction, testing, and Modern Code Review (MCR). These activities deliver ample prospects of producing wastes [2], [4], [5]. Waste is any act that does not produce any value to the user [2]. Concerning software engineering it can be “anything that doesn't make it to the release, is waste" [4]. It can also refer to any activity which uses resources but does not produce quality software [2], [4].

MCR is a software engineering activity for code improvement [6]. In MCR the code is reviewed by the reviewer, before committing the code to the project codebase. Unlike Fagan formal inspection process, MCR focuses on reviewing the small part of code changes usually named as 'patch' before saving the code into the codebase [26]. MCR regularly occurs in practice [8], [9], [11] with the help of code review tools [6], [9] such as Gerrit, Code flow, Review board, Phabricator, etc. It is a means to identify defects and to improve code quality [2], [6], [8], [9], [11], [12], [27], [28], [29] through knowledge sharing among developers. Fig. 1 shows the MCR process overview.

It is argued that MCR produces wastes such as waiting, development of extra or erroneous feature, defect, needless composite solution, rework, and mental distress [2], [3], [4], [5]. It is contended that waiting is the critical wastes [4], [30], [31]. It is argued that "one of the biggest wastes in software development is usually waiting for the thing to happen"[30]. It is also conveyed that if the organization has to minimize one waste, it should focus on a waiting [4], [30], [31].

Waiting waste refers to a delay between two consecutive activities [2], [3], [4], [5], [30], [31]. For example, in MCR delay between submitting source code review request by the author to the reviewer and getting feedback from the reviewer [8], [9], [17]. It is argued that one of the reasons for waiting waste in MCR is a poor knowledge sharing [5], [8], [13], [14], [15], [16], [17], [18]. The waiting waste decreases the productivity and efficiency of the developers [2], [4], [8], [12], [16], [17], [21], [26], [32]. It also causes project delays due to the blocking of tasks [4], [33].

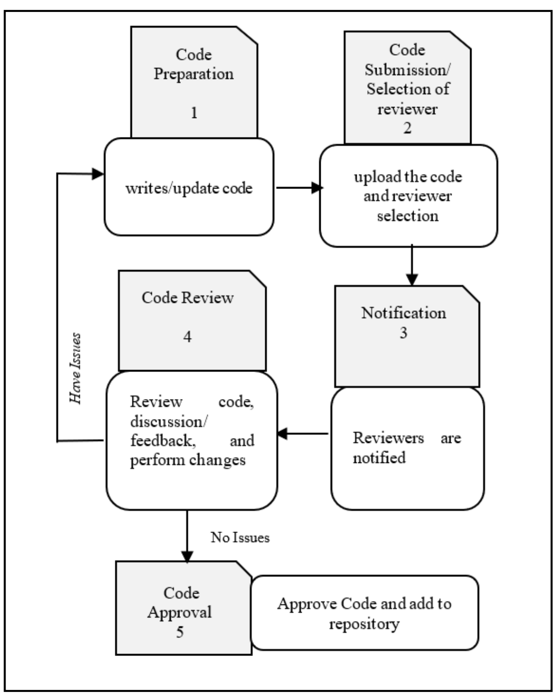

Fig. 1. MCR PROCESS OVERVIEW [9].

To minimize the waiting waste it is necessary to increase knowledge sharing [2], [4], [5], [19] among MCR team members. It is argued that knowledge sharing can be increased by identifying the factors influencing knowledge sharing [8], [9], [11], [20], [21] that can help in effective knowledge sharing among the MCR team.

Though preceding studies [22], [23], [24], [25] focused on knowledge sharing in software engineering activities, however, slight indication is available in MCR [8], [9], [11], [20], [21], resulting in absence of knowledge sharing guidelines in MCR. Therefore, the study aims to identify factors influencing knowledge sharing in MCR to minimize waiting waste.

Systematic Literature Review (SLR), has been directed to identify the factors influencing knowledge sharing in MCR. The expert review has been performed to confirm the identified factors influencing knowledge sharing for their naming, expressions, and categories.

III. RESEARCH METHODOLOGY

Multiple research activities have been performed to generate a distinct and categorized rundown of factors influencing the knowledge sharing in MCR to minimize the generation of waiting waste. The methodologies employed for this study are discussed in subsections.

A. Systemantic Literature Review

The Systematic Literature Review (SLR) methodology given by [34] has been used for this study to identify the relevant data sources for the identification of factors influencing knowledge sharing in MCR to minimize the generation of waiting waste. The SLR methodology is a systematized and well-organized approach to attain less impartial results [34]. It is an authentic methodology to record significant central focuses in the research for assessing and looking at all momentum research related to research questions. The detailed procedure of SLR is explained in subsections.

1) Research question: Constructing the research question is the central action of SLR [34]. Research questions are designed with the support of PICOC criteria specified by Petticrew and Roberts [35]. This investigation has excluded the 'Comparison' segment of the PICOC yet just PIOC has been considered to design the research question. The reason behind excluding the comparison part is that this study is not considering the comparison of techniques or models. Table I represents the PIOC criteria for this study.

TABLE. I. POIC SUMMARY

Population	MCR team
Intervention	MCR Process
Outcome	Factors influencing knowledge sharing in MCR to minimize waiting waste.
Context	The study includes all study types such as interviews, observations, surveys, experiments, questionnaires and case studies relating to MCR.

To gather the indications on the present state of research regarding factors influencing knowledge sharing in MCR to reduce waiting wastes. The designed question is specified below.

RQ1: What factors influence the knowledge sharing in MCR to minimize software engineering waiting waste?

2) Search Strategy: The search strategy comprises of identification of key terms and their alternate substitutes.

a) Identification of key term: The study key terms includes knowledge sharing, modern code review and software engineering waiting waste

b) Finding substitutes of identified key terms: The substitutes for the identified key terms are shown in Table II.

c) Use of Boolean OR to design search strings with key terms and their substitutes: The key terms along with their substitutes are joined using Boolean OR and are represented in Table III.

d) Use Boolean AND to concatenate the search key terms and limit the research: The designed search string is given below.

(‘Knowledge sharing’ OR ‘knowledge distribution’ OR ‘knowledge transfer’, ‘knowledge dissemination’ OR ‘knowledge exchange’) AND (review’ OR ‘modern code inspection ’OR ‘code review’ OR ‘code inspection ’OR ‘lightweight code review’) AND (‘Software Engineering Waiting Waste’ OR ‘software engineering delay waste’ OR ‘software engineering linger waste’ OR ‘software engineering blocking waste’ OR ‘software development delay waste’ OR ‘software development linger waste’)

TABLE. II. KEY TERMS AND THEIR SUBSTITUTES

Key term	Substitutes
Knowledge sharing	‘knowledge distribution’, ‘knowledge transfer’, ‘knowledge dissemination’, ‘knowledge exchange’
Modern Code Review	‘contemporary code review’, ‘modern code inspection’, ‘code review’, ‘code inspection’, ‘lightweight code review’
Software Engineering Waiting Waste	‘software engineering delay waste’, ‘software engineering linger waste’, ‘software engineering blocking waste’, ‘software development delay waste’, ‘software development linger waste’

TABLE. III. KEY TERMS WITH THEIR SUBSTITUTES AND BOOLEAN OR OPERATOR

Key terms, Substitutes and Boolean OR
‘Knowledge sharing’ OR ‘knowledge distribution’ OR ‘knowledge transfer’, ‘knowledge dissemination’ OR ‘knowledge exchange’
‘Modern Code Review’ OR ‘contemporary code review’ OR ‘modern code inspection ’OR ‘code review’ OR ‘code inspection ’OR ‘lightweight code review’
‘Software Engineering Waiting Waste’ OR ‘software engineering delay waste’ OR ‘software engineering linger waste’ OR ‘software engineering blocking waste’ OR ‘software development delay waste’ OR ‘software development linger waste’

e) Search process and database sources: The search process involved databases such as IEEE, Science Direct, ACM, Wiley online, Springer link, Web of Science and Scopus. The reason for selecting the above databases is that the selected databases are known to have software engineering literature. To make the search process comprehensive and to avoid the chance of missing out evidence, the search included the literature published from 2013 – 2019. Database sources that were considered are presented in Table IV along with their URLs and distribution.

f) Study Selection Criteria: The studies are included and excluded based on the inclusion and exclusion plan shown in Fig. 2.

Study Quality Assessment: Notwithstanding broad inclusion and exclusion criteria, it is viewed as basic to evaluate the "quality" of essential investigations. For the evaluation of concentrate quality, the checklist specified by [34] has been used. The investigations chosen after the introductory inclusion and exclusion plan are additionally assessed utilizing the checklist articulated in Table V.

The questions specified in the checklists represented in Table V are answered according to the rule specified by [34]. The evaluation scale is presented in Table VI.

TABLE. IV. DATABASE SOURCES

Data Source	URL
IEEE	http://ieeexplore.ieee.org/
ACM	http://dl.acm.org
Science Direct	http://www.sciencedirect.com
Wiley	http://onlinelibrary.wiley.com
Web of Science	https://www.webofknowledge.com
Springer link	https://www.springer.com
Scopus	http://www.scopus.com

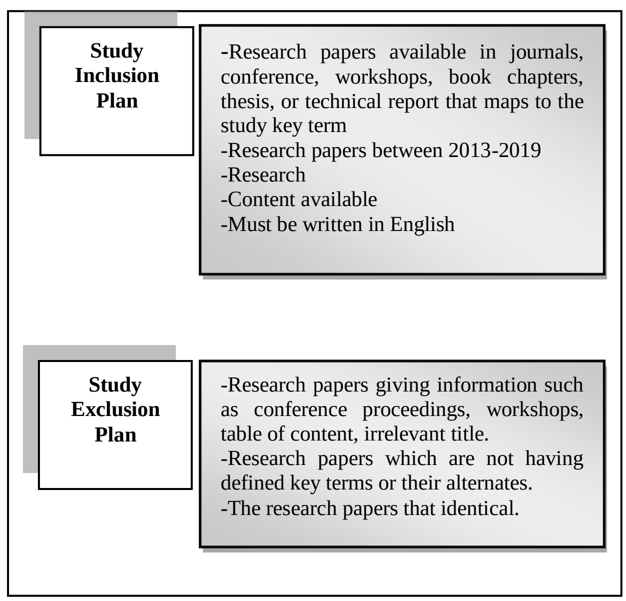

Fig. 2. INCLUSION AND EXCLUSION PLAN.

TABLE. V. QUALITY ASSESSMENT CHECKLIST

Question	Answer
Are the goals visibly detailed?	Yes/ No/Partially
Are the outcomes complete and substantial?	-do-
Are the prediction methods used visibly defined and their choice are acceptable?	-do-
Is the information been extended by the study?	-do-
Is the diversity of viewpoint and context been sightseen?	-do-
Are the links between data, understanding, and assumptions are vibrant?	-do-
Does the difficulty of the data is transferred?	-do-

TABLE. VI. SCALE FOR ANSWERING QUESTIONS GIVEN IN CHECKLIST [34]

Answer	Score
Yes	1
No	0
Partially	0.5

g) Data Extraction: After the essential studies have been chosen and their quality assessed, the data is extracted from the selected papers. The data extraction method is discussed in this section. The data extraction method is intended to contain all the data that is important for responding to the research question and tending to the investigation quality criteria [34]. The data extraction form is represented in Table VII.

h) Data Synthesis: After vigilant data extraction the extracted data is synthesized following the data coding, continual data comparisons and memoing from grounded theory [36] are adopted for data unit categorization, and to get the unique list of factors influencing knowledge sharing in MCR.

B. Expert Review

After getting the unique list of factors influencing knowledge sharing in MCR the list is evaluated through experts for naming, expression, categorization, and suggestions of new factors or categories. The considered experience for experts' selection is more than 10 years in software development knowing MCR, software engineering wastes and knowledge sharing. For expert review, the guidelines of Ayyub [30] are followed.

TABLE. VII. DATA EXTRACTION FORM [34]

Data characteristics	A unique identifier in the format: KSFP(1)…KSFP(n)
	Title
	Author (s)
	Year
	Study Set (Conference/Journal)
	Study Commissioner (IEEE, ACM, etc.)
	Selection (Inclusion/exclusion)/Quality assessment
Research Question	What factors influencing knowledge sharing in MCR?

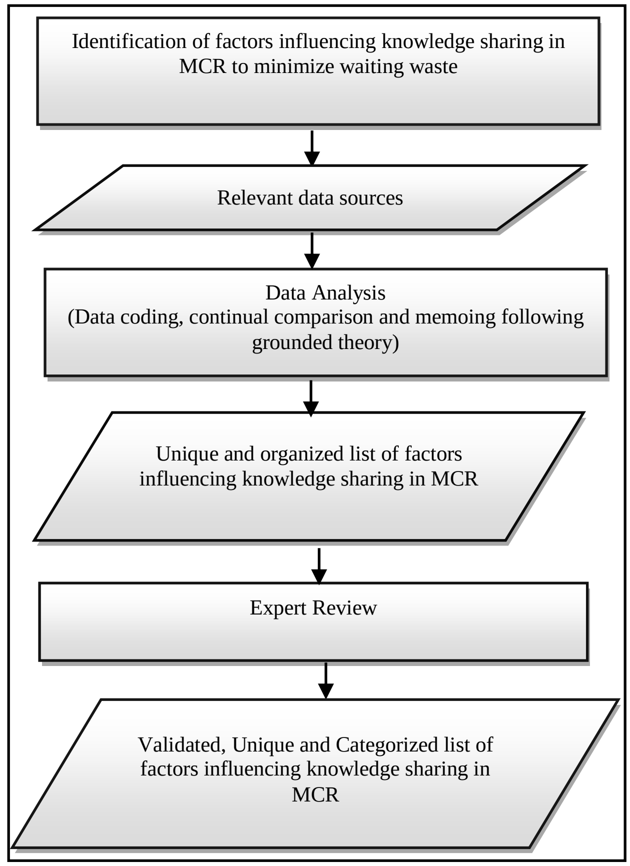

Fig. 3. DATA SYNTHESIS PROCEDURE.

Fig. 3 summarizes the data synthesis procedure employed for this study.

IV. RESULTS

This section discusses the results achieved in the study. It presents the results concerning the study search process to achieve pertinent data sources and the factors influencing knowledge sharing in MCR to minimize waiting waste.

A. Data Source Selection Results

Through initial search based on defined key terms, 9289 papers are obtained. The studies that represent only the table of content, conference or workshop preceding details or having unrelated titles are omitted. After the first exclusion, 1103 studies are obtained. The obtained 1103 studies are evaluated for the relevant key terms (modern code review, knowledge sharing, and software engineering waiting waste). The studies that do not have any of the correlated key term are eliminated and 190 studies are included. After assessment for having duplication among 190 studies, 162 studies are obtained and evaluated for their quality assessment. During the quality assessment, 6 studies are excluded and finally, 156 studies are recognized as most appropriate to this study and are included for detailed review.

B. Knowledge Sharing Factors in MCR

This section stretches the insights about factors influencing knowledge sharing in MCR to minimize waiting waste. The study results reported 22 factors that impact knowledge sharing in MCR, the identified factors are grouped under 5 broad categories namely Individual, Team, Social, Artifact and Facility Conditions. The details are represented in subsections. Table VIII provides a summarized view of the factors influencing knowledge sharing in MCR along with their references.

1) Individual: Individual perspective is the most noticeable lens in MCR [32]. The factors involved in this category are individual impartiality, individual historical factors, individual emotions, individual pressure, individual awareness, individual turnover, and individual intentions [9], [11], [17], [19], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47].

2) Social: MCR is a multifaceted process that involves social interactions among team members [32]. This category includes factors i.e. relational and structural factors [8], [9], [11], [17], [19], [48], [49], [50], [51].

3) Team: The team signifies a group of individuals who worked together to achieve a common goal. Their work involved multiple projects, from new to legacy systems [8]. This category involves factors i.e. team organization, team strategies, team culture, team, and team drive [8], [9], [11], [16], [32], [52], [48], [40], [53].

4) Artifact: An artifact, is an object made or given form by humans [12], [32]. This category includes factors such as source code, testing, feedback [8], [9], [11], [16], [19],[32], [52], [54], [55], [56], [57], [58], [59], [60].

5) Facility Conditions: Facility conditions support the successful conduction of the MCR [32]. This category involves factors i.e. project, process, tool, communication channel, and organization [9], [11], [19], [12], [15], [32].

Table VIII summarizes the validated list of factors influencing knowledge sharing in MCR prompting knowledge sharing in MCR along with the references.

TABLE. VIII. LIST OF KNOWLEDGE SHARING FACTORS IN MCR TO MINIMIZE SOFTWARE ENGINEERING WASTES

Categories	**Knowledge Sharing Factors**	**References**
INDIVIDUAL	Individual Impartiality	[9], [11], [17]
	Individual historical factors	[6], [8], [9], [16], [11], [17], [19], [32], [37], [52], [45], [48], [61], [62], [49], [38], [39], [40] , [41], [42] , [43], [44], [46]
	Individual Emotions	[8], [9], [15], [17], [32], [52] [63], [64]
	Individual Pressure	[6], [8], [9], [11], [15],[19], [32], [52], [48], [49], [40], [65], [54],
	Individual Awareness	[8], [9], [11], [14], [19], [32], [37], [52], [48], [49], [44], [65], [54], [66], [55], [56], [67],
	Individual Turnover	[64]
	Individual Intentions	[9], [11], [12], [17], [19], [37], [52], [61], [49] [64], [54], [56], [68], [69], [70],
SOCIAL	Relational	[8], [9], [11], [16], [17], [19], [32] , [48], [61], [49], [39], [40], [41], [42], [43], [44], [54], [70], [37], [57], [71], [72], [73], [74], [75]
	Structural	[15], [44], [50], [51]
ARTIFACT	Source Code	[8], [9], [11], [16], [19],[32], [6], [12] [15], [52], [45], [48], [38], [40], [41], [46], [63], [65], [54], [55], [56], [76], [77], [58], [78], [79]
	Feedback	[8], [9], [11], [15], [19], [32] [48], [40], [63], [54], [55], [56], [57], [72], [76], [58], [59]
	Testing	[8], [9], [11], [15], [19], [32], [52],[48], [75], [58], [59], [60],
FACILITY CONDITIONS	Process	[8], [9], [11], [19], [52], [48], [39], [78]
	Tool	[6], [8], [11], [12], [15], [32], [38] , [55], [71] [77], [78]
	Organization	[8], [12], [17], [32], [52], [38]
	Communication	[8], [9], [15], [52], [48], [38], [55]
	Project	[9], [11], [15], [32], [48]
TEAM	Team Organization	[8], [9], [11], [16], [32]
	Team Strategies	[8], [12], [15], [52]
	Team Culture	[8], [11], [52]
	Team Intensions	[6], [8], [9], [12], [32], [48], [40], [56]
	Team Drive	[8], [9], [11], [19], [32], [52], [48], [40], [53]

V. Discussion

This work stretches the direction to a comprehensive list of factors influencing knowledge sharing in MCR to minimize waiting waste. The identified factors are significant for software engineers involved in the MCR process. The preliminary list can act as a guide for the researchers and practitioners working in MCR to consider and these factors in order to increase knowledge sharing and to minimize waiting waste. This study contributed to the software engineering body of knowledge (SWEBOK) particularly to knowledge sharing in the context of MCR. The study helps the MCR team to achieve its objective while minimizing waiting waste.

VI. Limitations

This study lacks the identification of factors from the industry as the study comprises of factors that are recognized from the literature. A large effort has been made to cover all the correlated papers, but still, there is a possibility that some research may be missed.

VII. Conclusion

The research study provides a categorized list of factors influencing knowledge sharing in MCR to minimize waiting waste. The reported factors that influence knowledge sharing in MCR are distributed into five main categories that are Individual, Social, Team, Artifact and Facility Conditions. These factors ought to be considered while performing MCR to minimize waiting waste by increasing knowledge sharing.

VIII. Future Directions

A comprehensive list will be produced in the future by quantitative analysis, the ongoing research objectives. In addition to this, a comprehensive model can be produced for MCR that can be used as a guideline for software engineers to minimize software engineering waiting waste. This work recognizes factors influencing knowledge sharing in MCR that provides the foundation for the investigators to outspread this research by discovering other factors for other software development activities to reduce wastes.

IX. Contribution

The examination contributed towards software engineering body of knowledge (SWEBOK), knowledge base software engineering (KBSE) and green software engineering (GREEN SE) by perceiving the significance of knowledge sharing and by giving the arranged rundown of factors influencing knowledge sharing in MCR. The work can help software developers to successfully transfer knowledge by overcoming the negative aspects of identified factors.

References

[1] Alvertis et al., "User Involvement in Software Development Processes," Procedia Comput. Sci., vol. 97, pp. 73–83, 2016.

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

[3] M. Poppendieck and T. Poppendieck, "Implementing Lean Software Development: From Concept to Cash," Addison-Wesley Signat. Ser., p. 304, 2006.

[4] H. Alahyari, T. Gorschek, and R. Berntsson Svensson, "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. August 2018, pp. 78–94, 2019.

[5] "Software Engineering Wastes – A Perspective of Modern Code Review," 2020.

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

[7] M. E. Fagan, "Design and code inspections to reduce errors in program development," IBM Syst. J., vol. 38, no. 2.3, pp. 258–287, 1999.

[8] 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.

[9] 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.

[10] A. Aurum, H. Petersson, and C. Wohlin, "State-of-the-art: Software inspections after 25 years," Softw. Test. Verif. Reliab., vol. 12, no. 3, pp. 133–154, 2002.

[11] 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.

[12] P. C. Rigby and C. Bird, "Convergent Contemporary Software Peer Review Practices Categories and Subject Descriptors," in Proc. ESEC/FSE, 2013, pp. 202–212.

[13] J. C. Carver, B. Caglayan, M. Habayeb, B. Penzenstadler, and A. Yamashita, "Collaborations and code reviews," IEEE Softw., vol. 32, no. 5, pp. 27–29, 2015.

[14] J. Czerwonka, M. Greiler, and J. Tilford, "Code Reviews Do Not Find Bugs. How the Current Code Review Best Practice Slows Us Down," in Proc. IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015, vol. 2, pp. 27–28.

[15] G. Gousios, M.-A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development," in Proc. 38th International Conference on Software Engineering, 2016, pp. 285–296.

[16] E. W. dos Santos and I. Nunes, "Investigating the Effectiveness of Peer Code Review in Distributed Software Development," in Proc. 31st Brazilian Symposium on Software Engineering, 2017, pp. 84–93.

[17] 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.

[18] L. Novikova, "Poor knowledge sharing is the second biggest challenge for software development teams," 2019. [Online]. Available: https://blog.onebar.io/poor-knowledge-sharing-is-the-second-biggest-challenge-for-software-development-teams-a4843f9b9aa. [Accessed: 10-Aug-2019].

[19] A. Ram, Achyudh ; Sawant, Anand; Castelluccio, Marco; Bacchelli, "What Makes a Code Change Easier to Review? An Empirical Investigation on Code Change Reviewability," in Proc. ESEC/FSE, 2018.

[20] N. Fatima, S. Nazir, and S. Chuprat, "Knowledge sharing, a key sustainable practice is on risk: An insight from Modern Code Review," in 2019 6th IEEE International Conference on Engineering Technologies and Applied Sciences (ICETAS), 2019.

[21] N. Fatima, S. Nazir, and S. Chuprat, "Understanding the Impact of Feedback on Knowledge Sharing in Modern Code Review," in 6th IEEE International Conference on Engineering Technologies and Applied Sciences (ICETAS) , In Press, 2019.

[22] R. Anwar et al., "Conceptual Framework for Implementation of Knowledge Sharing in Global Software Development Organizations," pp. 174–178, 2017.

[23] X. Chen, Y. Zhou, D. Probert, and J. Su, "Technology Analysis & Strategic Management Managing knowledge sharing in distributed innovation from the perspective of developers : empirical study of open source software projects in China," Technol. Anal. Strateg. Manag., vol. 7325, no. July, 2016.

[24] N. S. Safa and R. Von Solms, "An information security knowledge sharing model in organizations," Comput. Human Behav., vol. 57, pp. 442–451, 2016.

[25] Q. Zhang and R. Du, "Impacts of cultural difference on knowledge sharing, relationship quality and performance in IT-based service outsourcing," 2011 2nd Int. Conf. Artif. Intell. Manag. Sci. Electron. Commer. AIMSEC 2011 - Proc., pp. 6271–6274, 2011.

[26] C. Thompson and D. Wagner, "A Large-Scale Study of Modern Code Review and Security in Open Source Projects," Proc. 13th Int. Conf. Predict. Model. Data Anal. Softw. Eng., pp. 83–92, 2017.

[27] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," Empir. Softw. Eng., vol. 21, no. 5, pp. 2146–2189, 2016.

[28] 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.

[29] S. Nazir, N. Fatima, and S. Chuprat, "Modern Code Review Benefits–Primary findings of a systematic literature review," in The 3rd International Conference on Software Engineering and Information Management.

[30] M. Poppendieck and T. Poppendieck, Lean Software Development: An Agile Toolkit. 2003.

[31] 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.

[32] 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.

[33] M. Ikonen, P. Kettunen, N. Oza, and P. Abrahamsson, "Exploring the sources of waste in Kanban software development projects," in Proc. - 36th EUROMICRO Conference on Software Engineering and Advanced Applications, 2010, pp. 376–381.

[34] B. Kitchenham and S. Charters, "Source: " Guidelines for performing Systematic Literature Reviews in SE " , Kitchenham et al Guidelines for performing Systematic Literature Reviews in Software Engineering Source: " Guidelines for performing Systematic Literature Reviews i," pp. 1–44, 2007.

[35] F. Terms, " M. Petticrew and H. Roberts. Systematic Reviews in the Social Sciences: A Practical Guide . Oxford: Blackwell 2006. 352 pp. ISBN 1 4051 2110 6. £29.99 ," Couns. Psychother. Res., vol. 6, no. 4, pp. 304–305, 2006.

[36] K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research," no. October 2017, pp. 120–131, 2016.

[37] A. Bosu, J. Carver, R. Guadagno, B. Bassett, D. McCallum, and L. Hochstein, "Peer impressions in open source organizations: A survey," J. Syst. Softw., vol. 94, pp. 4–15, 2014.

[38] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Confusion detection in code reviews," in Proc. IEEE International Conference on Software Maintenance and Evolution, 2017, pp. 549–553.

[39] 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.

[40] A. Ouni, R. G. Kula, and K. Inoue, "Search-based peer reviewers recommendation in modern code review," in Proc. - IEEE International Conference on Software Maintenance and Evolution, 2017, pp. 367–377.

[41] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Review participation in modern code review," Empir. Softw. Eng., vol. 22, no. 2, pp. 768–817, 2017.

[42] T. Baum, O. Liskin, K. Niklas, and K. Schneider, "A Faceted Classification Scheme for Change-Based Industrial Code Review Processes," Proc. - 2016 IEEE Int. Conf. Softw. Qual. Reliab. Secur. QRS 2016, pp. 74–85, 2016.

[43] N. Kitagawa, H. Hata, A. Ihara, K. Kogiso, and K. Matsumoto, "Code Review Participation: Game Theoretical Modeling of Reviewers in Gerrit Datasets," in Proc. 9th International Workshop on Cooperative and Human Aspects of Software Engineering, 2016, pp. 64–67.

[44] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?," Inf. Softw. Technol., vol. 000, pp. 1–15, 2015.

[45] S. McIntosh and Y. Kamei, "Are Fix-Inducing Changes a Moving Target? A Longitudinal Case Study of Just-In-Time Defect Prediction," IEEE Trans. Softw. Eng., vol. 44, no. 5, pp. 412–428, 2018.

[46] R. Morales, S. McIntosh, and F. Khomh, "Do code review practices impact design quality? A case study of the Qt, VTK, and ITK projects," in Proc. IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2015, pp. 171–180.

[47] N. Fatima and K. Lumpur, "Individual , Social and Personnel Factors Influencing Modern Code Review Process."

[48] O. Kononenko, T. Rose, O. Baysal, M. Godfrey, D. Theisen, and B. De Water, "Studying Pull Request Merges : A Case Study of Shopify ' s Active Merchant," in Proc. 40th International Conference on Software Engineering: Software Engineering in Practice, 2018, pp. 124–133.

[49] H. Lal and G. Pahwa, "Code review analysis of software system using machine learning techniques," in in Proc. 11th International Conference on Intelligent Systems and Control, 2017, pp. 8–13.

[50] X. Yang, N. Yoshida, R. Gaikovina Kula, and H. Iida, "Peer Review Social Network (PeRSoN) in open source projects," IEICE Trans. Inf. Syst., vol. E99D, no. 3, pp. 661–670, 2016.

[51] A. J. A. M. Van Deursen, C. Verlage, and E. Van Laar, "Social Network Site Skills for Communication Professionals: Conceptualization, Operationalization, and an Empirical Investigation," IEEE Trans. Prof. Commun., vol. 62, no. 1, pp. 43–54, 2019.

[52] T. Baum, O. Liskin, K. Niklas, and K. Schneider, "Factors influencing code review processes in industry," Proc. 2016 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng. - FSE 2016, pp. 85–96, 2016.

[53] T. Zhang, M. Song, J. Pinedo, and M. Kim, "Interactive code review for systematic changes," Proc. - Int. Conf. Softw. Eng., vol. 1, pp. 111–122, 2015.

[54] Z. Xia, H. Sun, J. Jiang, X. Wang, and X. Liu, "A Hybrid Approach to Code Reviewer Recommendation with Collaborative Filtering," in SoftwareMining 2017, Urbana-Champaign, IL, USA, 2017, pp. 24–31.

[55] 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.

[56] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern code reviews in open-source projects: which problems do they fix?," in Proc. 11th Working Conference on Mining Software Repositories, 2014, pp. 202–211.

[57] J. Jiang, Y. Yang, J. He, X. Blanc, and L. Zhang, "Who should comment on this pull request? Analyzing attributes for more accurate commenter recommendation in pull-based development," Inf. Softw. Technol., vol. 84, pp. 48–62, 2017.

[58] D. Spadini, A. Bacchelli, M. Bruntink, F. Palomba, and L. Pascarella, "Information Needs in Contemporary Code Review," in Proc. ACM on Human-Computer Interaction, 2018, vol. 2, no. CSCW, pp. 1–27.

[59] R. Wen, D. Gilbert, M. G. Roche, and S. McIntosh, "BLIMP tracer: Integrating build impact analysis with code review," in Proc. IEEE International Conference on Software Maintenance and Evolution, 2018, pp. 685–694.

[60] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, "Wait for It: Determinants of pull request evaluation latency on GitHub," in Proc. IEEE International Working Conference on Mining Software Repositories, 2015, vol. 2015-Augus, pp. 367–371.

[61] A. Bosu, "Modeling Modern Code Review Practices in Open Source Software Development Organizations," in Proc. IDoESE '13 Baltimore, Maryland USA, 2013.

[62] L. Claytor and F. Servant, "Understanding and Leveraging Developer Inexpertise," Proc. 40th Int. Conf. Softw. Eng. Companion Proceeedings, pp. 404–405, 2018.

[63] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Confusion in Code Reviews: Reasons, Impacts, and Coping Strategies," SANER 2019 -

Proc. 2019 IEEE 26th Int. Conf. Softw. Anal. Evol. Reengineering, pp. 49–60, 2019.

[64] A. M. Peter C. Rigby, Yue Cai Zhu, Samuel M. Donadelli, "Quantifying and Mitigating Turnover-Induced Knowledge Loss: Case Studies of Chrome and a Project at Avaya," in Proc. IEEE/ACM 38th International Conference on Software Engineering, 2016, pp. 1006–1016.

[65] S. Fakhoury, "The Effect of Poor Source Code Lexicon and Readability on Developers´s Cognitive Load," in Proc. ICPC, 2018, pp. 286–296.

[66] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Revisiting code ownership and its relationship with software quality in the scope of modern code review," Proc. 38th Int. Conf. Softw. Eng. - ICSE '16, no. 1, pp. 1039–1050, 2016.

[67] C. Parnin et al., "The Top 10 Adages in Continuous Deployment," IEEE Softw., vol. 34, no. 3, pp. 86–95, 2017.

[68] B. Floyd, T. Santander, and W. Weimer, "Decoding the Representation of Code in the Brain: An fMRI Study of Code Review and Expertise," in Proc.-IEEE/ACM 39th International Conference on Software Engineering, 2017, pp. 175–186.

[69] A. Murgia et al., "The emotional side of software developers in JIRA," pp. 480–483, 2016.

[70] J. Shimagaki, Y. Kamei, S. McIntosh, A. E. Hassan, and N. Ubayashi, "A study of the quality-impacting practices of modern code review at Sony mobile," in Proc. 38th International Conference on Software Engineering Companion, 2016, pp. 212–221.

[71] Z. X. Li, Y. Yu, G. Yin, T. Wang, and H. M. Wang, "What Are They Talking About? Analyzing Code Reviews in Pull-Based Development Model," J. Comput. Sci. Technol., vol. 32, no. 6, pp. 1060–1075, 2017.

[72] 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.

[73] A. Meneely et al., "An empirical investigation of socio-technical code review metrics and security vulnerabilities," 6th Int. Work. Soc. Softw. Eng. SSE 2014 - Proc., pp. 37–44, 2014.

[74] Y. Jiang, B. Adams, and D. M. German, "Will my patch make it? And how fast?: Case study on the linux kernel," in Proc. IEEE International Working Conference on Mining Software Repositories, 2013, no. Section II, pp. 101–110.

[75] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of Social and Technical Factors for Evaluating Contribution in GitHub," in Proc. International Conference on Software Engineering, 2014, pp. 356–366.

[76] A. Luxton-reilly, A. Lewis, and B. Plimmer, "Comparing Sequential and Parallel Code Review Techniques for Formative Feedback," in Proc. 20th Australasian Computing Education Conference, 2018, pp. 45–52.

[77] J. Kim and E. Lee, "Understanding review expertise of developers: A reviewer recommendation approach based on latent Dirichlet allocation," Symmetry (Basel)., vol. 10, no. 4, pp. 5–7, 2018.

[78] M. M. Rahman and C. K. Roy, "Impact of continuous integration on code reviews," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 499–502, 2017.

[79] T. Baum, F. Kortum, K. Schneider, A. Brack, and J. Schauder, "Comparing pre-commit reviews and post-commit reviews using process simulation," J. Softw. Evol. Process, vol. 29, no. 11, pp. 1–15, 2017.