Noname manuscript No.
(will be inserted by the editor)

The Evolution of Code Review Research: A Systematic Mapping Study

Dong Wang · Yuki Ueda · Raula Gaikovina Kula · Takashi Ishio · Kenichi Matsumoto

the date of receipt and acceptance should be inserted later

Abstract Code Review (CR) is a cornerstone for Quality Assurance within software development teams. Also known as *"software inspections"* and *"walk-throughs"*, traditional CR involved time-consuming processes, which is different from more lightweight contemporary forms used today. In this paper, we aim to summarize how CR research has evolved into its current state over the last decade. Our vigorous systematic study revolves around four research questions to uncover changes into the target of *contributions* and *methodologies*, *replicability* of existing studies and the evolution of CR *terminology*. From a collection of 7,266 papers from the top software engineering venues, we generate visual maps for 148 collected papers including 53 conferences, 16 journals, and 79 snowball papers. Our visual maps provide evidence that CR research does cover more than quality assurance, and will continue to evolve with the availability of datasets and emerging technologies within the CR domain.

Keywords Code Review · Modern Code Review · Quality Assurance · Software Inspections · Systematic Mapping study

Dong Wang
Department of Computer Engineering
Nara Institute of Science and Technology, Japan
E-mail: wang.dong.vt8@is.naist.jp

Yuki Ueda
E-mail: ueda.yuki.un7@is.naist.jp

Raula Gaikovina Kula
E-mail: raula-k@is.naist.jp

Takashi Ishio
E-mail: ishio@is.naist.jp

Kenichi Matsumoto
E-mail: matumoto@is.naist.jp

1 Introduction

Code Review (CR) is known traditionally as a key cornerstone behind Quality Assurance for software development teams. However other than finding defects, practitioners also believe that the CR process itself also plays a crucial role in knowledge transfer, team building and coordination within software teams [8]. The importance of CR is apparent with industry giants like Microsoft and Google releasing insights on how *"CRs at Microsoft are an integral part of the development process that thousands of engineers perceive it as a great best practice and most high-performing teams spend a lot of time doing"*.

CR research has been constantly evolving. Older mapping studies [6, 70] survey the early work of "software inspections" and "walk-throughs" in the 1970s [41]. In contrast, the rise of contemporary review tools has brought the availability of data. Known as the Modern Code Review (i.e., MCR), the review process is light-weight and the availability of the data allows researchers to download and analyze the process. Contemporary tool-based reviews (such as Gerrit[1], Codestriker[2], and ReviewBoard[3]) are widely used in both open source and proprietary software projects. Furthermore, platforms like GitHub promote the use of "Pull Requests", where a developer can review changes before merging to the code base. These technologies have led to a plethora of studies conducted in the field. Furthermore, the field has seen a growing number of survey and user studies carried out with developers that use these tools.

[1] `https://www.Gerritcodereview.com/`
[2] `http://codestriker.sourceforge.net/`
[3] `https://www.reviewboard.org/`

In this paper, we aim to summarize CR research to identify mature topics (i.e., contributions, methodologies, datasets, terminology) and their gaps over the last decade. We report on a systematic mapping study of the CR research area. A systematic mapping study provides a structure of which related research papers and results published can be categorized using a vigor process to visually summarize and map the research area. This method is a popular methodology in software engineering and has been common practice in mature fields like medical research.

The scope of our systematic study revolves around four research questions, to uncover state into the target of contributions and methodologies for most research, replicability of existing research and the evolution of CR terminology. Our research makes the following contributions:

- Systematic maps that show an update into the current state of research in CR over the ten years.
- A catalog of collected papers and datasets used in CR research, which is a step towards replicability and reference for researchers and practitioners.
- Deeper insights into mature topics and gaps in CR research.

From a collection of 7,266 papers from the top software engineering venues, we generated visual maps for the final 148 collected papers including 53 conferences, 16 journals, and 79 snowball papers. Our visual maps result in these conclusions: (i) mature evaluation and validation methodologies have targeted socio-technical and the understanding aspects of the CR process. (ii) The Qt code review project is the most popular dataset used by researchers. We find that fewer researches provide replicable datasets. (iii) GitHub's pull request is becoming a trendy topic in CR research, and (iv) more papers have been published in conferences than journals over the last 10 years.

The remainder of this paper is organized as follows. Section 2 presents the systematic mapping process including research questions, search conduction, screening process, classification schemes and data extraction. Section 3 shows the results of the systematic mapping study. Section 4 describes the state of CR research. Section 5 explains the threats to validity of the research. Finally, we summarize this paper in Section 6. The full catalog of the papers and their classifications are available at https://naist-se.github.io/code-review/.

2 The Systematic Mapping Process

Figure 1 outlines the mapping process used in this study. Our process is based on the work of Petersen et al. [101]. Similar to the systematic mapping study performed by Abelein and Paech [2], we introduce each process step as separate sections. Essentially, our process steps of the systematic mapping study are the definition of the research questions, search conduction of papers, screening process, keywording for the mapping and data extraction. In the end, the outcomes of the process are systematic maps of the research area.

2.1 Research Questions

To define the scope of the mapping study, we formulate the following research questions. Since the final goal is an overview of the research area, the research questions quantify the research attributes that the map is based on. In the first two research questions (RQ1 and RQ2), we exclusively focus on only CR research that has been published in the premium venues of Software Engineering.

1. *(RQ1): What contributions and methodologies does CR research target?* The motivation for the first research question is to understand the current focus of research. Based on the work of Bacchelli and Bird [8], we would like to map out the outcomes, expectations, and contributions that the most impactful CR research tackles from the point of view of both a practitioner and researcher.
2. *(RQ2): How much CR research has the potential for replicability?* The motivation for the second research question is to understand how the data source impact CR research. Understanding the sources can provide insight into the current state and gaps in terms of the data collection and availability. Furthermore, there has been growing initiatives to make data open and replicability which is encouraged in the community[4].

RQ3 and RQ4 then expand the study scale including 79 snowball papers. The motivations are listed below:

3. *(RQ3): How has CR terminology changed over time?* The motivation for research question three is to understand the evolution of the research field, especially in terms of the terminology and techniques that have been employed over a decade of research. The output is identified in terminology and technology that researchers use in the field.
4. *(RQ4): Which SE locations include papers on CR research?* The motivation for the last research question is to understand when CR research transitioned

[4] A recent initiative by the Springer EMSE Journal shows how research is working towards open science and replicable studies`https://github.com/emsejournal/openscience`

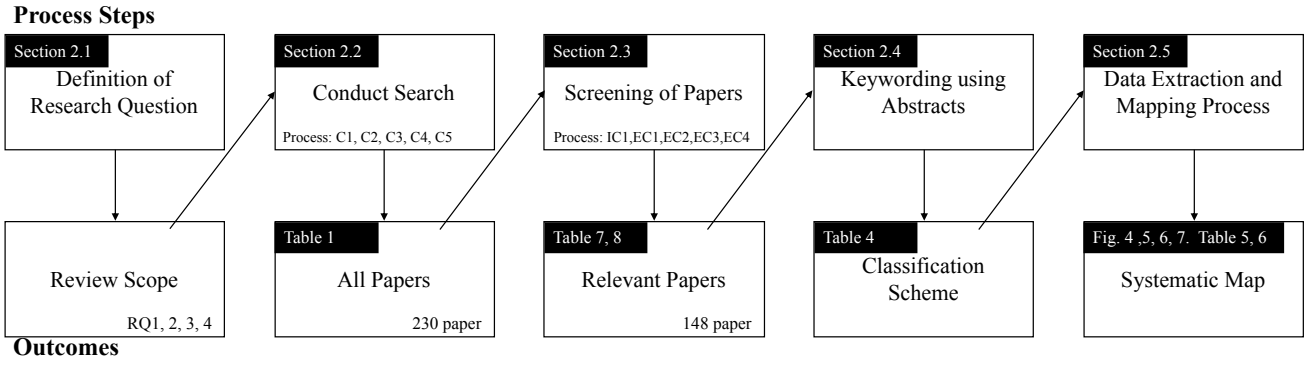


Fig. 1 An Overview of our Mapping Study Process

from the lower-tier workshops and conferences into the premium venues. Such a map can help researchers understand the changes of locations that have impactful CR research.

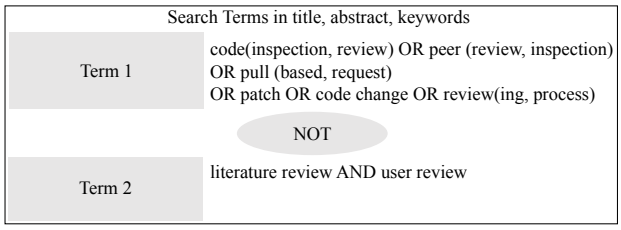


Fig. 2 Defined terms used in the search strings

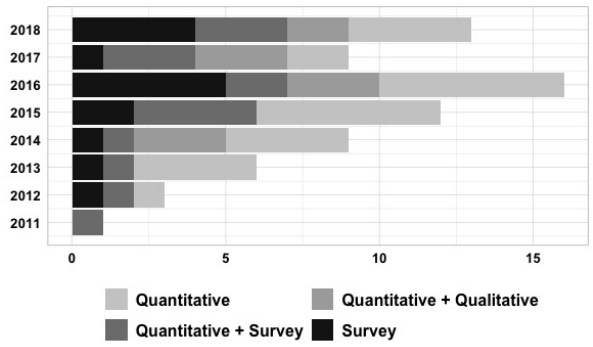


Fig. 3 Majority of collected papers were published in 2016 and tend to increase in recent days.

2.2 Conduct Search

We use the following strict characteristics as recommended by A. Kitchenham [1] to formulate our search string:

- (C1) a defined search strategy
- (C2) a defined search string, based on a list of synonyms combined by ANDs and ORs
- (C3) a broad collection of search sources
- (C4) strict documentation of the search
- (C5) paper selection should be checked by at least two researchers

Figure 2 shows the (C1) and (C2) a defined search strategy. We apply commonly used terminologies in CR to search, such as code inspection, code review, peer review, peer inspection or tools such as pull requests or pull based. Other terminologies such as patch, code changes, and reviewing are considered as appropriate for CR research. Please note that we decided to remove literature studies and user review papers from the search string.

Table 1 shows the source of paper collection in which the search was conducted. Based on RQ1, to ensure a high quality of papers and to understand the state-of-the-art in the field, we specifically searched for papers in the top journals and conferences from the software engineering domain. Hence, papers were extracted from five premium conferences (i.e., International Conference on Software Engineering) and five premium journals with high impact factors. To reduce its selection bias, we selected from a wide range of digital resources to follow (C3) a broad collection of search sources: ACM Digital Library, IEEE Xplore, Science Direct, and SpringerLink databases. For example, the data from 2012 to 2018 for Mining Software Repositories Conference is collected through IEEE Xplore, but the data from 2011 is available in ACM Digital Library.

We extracted 7,266 papers from above four search sources that were published in the last eight years (i.e., 2011∼2018). To ensure only technical contributions, in the further data processing, we filter out short papers, editorials, tutorials, panels, poster sessions and prefaces

Table 1 Targeted SE journals and conferences with Rankings. (Impact factors (IF) as of 2019)

Type	Location	Rankings	Published (from 2011)
Journal	(TSE) IEEE Transactions on Software Engineering	IF:4.778	∼2018
	(EMSE) Empirical Software Engineering	IF:4.457	∼2018
	(TOSEM) ACM Transactions on Software Engineering and Methodology	IF: 2.96	∼2018
	(IST) Information and Software Technology	IF:2.921	∼2018
	(ASEJ) Automated Software Engineering Journal	IF:2.2	∼2018
Conference	(ICSE) International Conference on Software Engineering	Rank: A*	∼2018
	(ESEC/FSE) ACM Join European Software Engineering Conference and Symposium on Foundation of Software Engineering	Rank: A*	∼2018
	(ASE)Automated Software Engineering	Rank: A	∼2018
	(ICSME) International Conference on Software Maintenance and Evolution	Rank: A	∼2018
	(MSR) Mining Software Repositories	Rank: A	∼2018

Table 2 Snowball Papers in the Relevant Journal Papers

Location	#Papers
(TSE) IEEE Transactions on Software Engineering	11
IEEE Software	11
(TOSEM) ACM Transactions on Software Engineering and Methodology	2
(EMSE) Empirical Software Engineering	1
(JSS) Journal of Systems and Software	1
IBM Journal	1
(IST) Information and Software Technology	1
Journal Human-Computer Studies	1
(JMIS) Journal of Management Information Systems	1
(STVR) Software Testing, Verification and Reliability	1
(SPIP) Software Process Improvement and Practice	1
Quality Software	1
(IEICE) Institute of Electronics, Information and Communication Engineers	1
Journal of Computer Science and Technology	1
The Open Software Engineering Journal	1
Science China Information Sciences	1

and opinions (i.e., we automatically filter out papers which are shorter than 8 pages).

2.3 Screening Process

Our screening process is comprised of (1) inclusion and exclusion criteria and (2) snowballing of references. The defined inclusion and exclusion criteria are relevant to answer RQ1 and RQ2, which focus on the state of premium CR research. For this manual exclusion, the following inclusion and exclusion criteria were applied to the abstract of each paper.

Inclusion criteria: Only a single inclusion criterion is defined, namely,

- *(IC1)*: paper should focus on code inspections/code review/code review tools.

Exclusion criteria: Four exclusion criteria were defined that cover the datasets, purposes and the evaluation of the studies. The following papers were excluded that met these criteria.

- *(EC1)*: the paper does not mention any CR activities.
- *(EC2)*: the paper focuses on other software development process e.g. issue tracking process, continuous integration, testing.
- *(EC3)*: the paper is out of scope with focusing on other sub-fields such as program analysis, code clone, defect prediction, refactoring, social technique.
- *(EC4)*: the paper is from books, tech report, thesis and shorter than 8 pages. (for snowballs)

To reduce bias and follow (C5), this manual paper selection was conducted by the first and the second authors. As a result of Step3, we were able to collect 69 papers out of 230 initial papers which include 53 premium conference papers and 16 high-impact journal papers. Details of all papers are shown in Table 12 and Table 13.

As shown in Table 2 and Table 3, we expand our papers into other influential papers by performing a snowballing of the references as mentioned by A. Kitchenham [1] from 69 collected papers. In detail, three of the co-authors manually extracted all references, then reapplied the exclusion criteria to these papers. To ensure quality, we filtered out short papers, editorials, tutorials, panels, poster sessions for snowballs. After this step, we are able to collect 79 snowball papers. Finally, we have ended up with 148 relevant papers as shown in Table 5.

Table 3 Snowball Papers in the Relevant Conference/Workshop Papers

Location	#Papers
Conference	
(ICSE) International Conference on Software Engineering	6
(SANER) International Conference on Software Analysis, Evolution and Reengineering (including WCRE)	5
(ESEC/FSE) ACM Join European Software Engineering Conference and Symposium on Foundation of Software Engineering	3
(ICSME) International Conference on Software Maintenance and Evolution	2
(Profes) International Conference on Product-Focused Software Process Improvement	2
(ESEM) International Symposium on Empirical Software Engineering and Measuremen	2
(ASE)Automated Software Engineering	2
(CSCW) ACM Conference on Computer-supported Cooperative Work	2
(MSR) Mining Software Repositories	1
(EASE) Evaluation and Assessment in Software Engineering	1
(QRS) International Conference on Software Quality, Reliability and Security	1
(APSEC) Asia-Pacific Software Engineering Conference	2
(CSEET) IEEE Conference on Software Engineering Education and Training	1
(CTS) Collaboration Technologies and Systems	1
(ISSTA) ACM SIGSOFT International Symposium on Software Testing and Analysis	1
(SCAM) International Working Conference on Source Code Analysis and Manipulation	1
(OSS) Open Source Software	1
(ECIS) European Conference on Information Systems	1
(ETRA) Symposium on Eye Tracking Research & Applications	1
(METRICS) International Software Metrics Symposium	1
Symposium on Software validation: Inspection-Testing-Verification-Alternatives	1
(ICMLA) IEEE International Conference on Machine Learning and Applications	1
Workshop	
Workshop on Social Software Engineering	1
Workshop on Evaluation and Usability of Programming Languages and Tools	1
Workshop on Principles of Software Evolution	1

After the collection process, we manually classified the types of research papers (i.e., quantitative and qualitative) [22]. We classify the research types into four categories: i) Quantitative, ii) Quantitative + Qualitative, iii) Quantitative + Survey and iv) Survey. Two authors classified them in the first round and then the third author did the validation. Note that Survey not only includes survey but also includes interview and user studies. Figure 3 shows the distribution of paper types from 2011 to 2018. As we can see from Figure 3, overall quantitative one is the most popular type to conduct the researches.

2.4 Keywording of Relevant Papers

Inspired by Petersen et al. [101], we classified each paper based on the scope outlined in each research question. Instead of keywording from the abstracts, we use related work and existing attributes of the contributions, methodologies to create a classification.

Details of the four types of classification scheme are shown in Table 1 and not only includes detailed reading of the abstract, but sometimes requires a careful reading of the whole paper itself. All papers were classified according to the following classification schemes. The categories were easy to interpret and use for classification. However, in many cases, we had to evaluate the paper in detail to confirm our study.

Contributions (RQ1). To classify research contributions of the papers, we base our work on the work of Bacchelli and Bird [8]. They classify contributions for two objectives (i.e., contributions to benefit practitioner and researcher). For this process, papers from the snowballs were not included in the classification.

For the classification process, three co-authors sat in a round-table and labeled each contribution based on seven category features shown in Table 4. The process was to first read the abstract and decide the classification. If there was a dispute, then the paper was quickly analyzed and a discussion of the paper started between the co-authors before the consensus reached.

Methodologies (RQ1). To classify methodologies that were applied to the studies, we used existing definitions of research facets [101]. For this process, papers from the snowballs were not included in the classification.

For the classification, three co-authors sat in a round table and labeled each methodology based on the category features. First keywords relating to the methodology were searched and discussed. Similar to the keywording of contributions, the full contents of the paper were consulted if a dispute arose among the co-authors.

Replication (RQ2). To classify the replicability of papers, we identified the source of the data, whether the dataset is either available via the link or is referred to a prior dataset.

Since detailed information of the dataset is not likely to be in the abstracts, co-authors were required to scan the papers to extract any online links of a dataset or a

Table 4 Summary of the classification scheme used to identify contributions, methodology and replicability.

Class	Sub-class	Category	Description
Contributions	Practitioner	Communication [8]	The developers are provided with the need of richer communication than comments annotating the changed code when reviewing. Teams should provide mechanisms for in-person or, at least, synchronous communication.
		Potential Benefit [8]	Modern CR provides benefits beyond finding defects. CR can be used to improve code style, find alternative solutions, increase learning, share code ownership, etc. This should guide CR policies.
		Quality Assurance [8]	CR does not result in identifying defects as often as project members would like and even more rarely detects deep, subtle, or macro level issues.
		Understanding [8]	When reviewers have prior knowledge of the context and the code, they complete reviews more quickly and provide more valuable feedback to author.
	Researcher	Automation [8]	Tools for enforcing team code conventions, checking for typos, and identifying dead code already exist. Even more advanced tasks such as checking boundary conditions or catching common mistakes have been shown to work in practice on real code. Automating these tasks frees reviewers to look for deeper, more subtle defects.
		Program comprehension [8]	Context and change understanding are challenges that developers face when reviewing, with a direct relationship to the quality of review comments.
		Socio-technical effect [8]	These are studies that involves the consideration of both human and technical aspects. In terms of CR, Studies can be designed and carried out to determine if and how team collaboration, coordination, awareness and learning occurs.
Methodologies	–	Validation Research [146]	Techniques investigated are novel and have not yet been implemented in practice. Techniques used are for example experiments, i.e, work done in lab
		Evaluation Research [146]	Techniques are implemented in practice and an evolution of the technique is conducted.That means, it is shown how the technique is implemented in practice (solution implementation) and what are the consequences of the implementation in terms of benefits and drawbacks (implementation evaluation). This also includes to identify problems in industry.
		Solution Proposal [146]	A solution for a problem is proposed, the solution can be either novel or a significant extension of an existing technique. The potential benefits and the applicability of the solution is shown by small example or a good line of argumentation.
		Experience Paper [146]	Experience papers explain on what and how something has been done in practice. It has to be the personal experience of the author
		Survey Paper	These papers are qualitative studies that use a questionnaire or interviews to evaluate some phenomena
Replication	-	Closed Data	The source of the dataset is from survey, interview, industry, and replication is not provided.
		Open Data	The source of the dataset is identified within Open Source projects, but replication is not provided.
		Data Available via Links	Replication is provided via hyper links.
		Reference to Dataset	Dataset is adopted from previous published works.
Terminology	-	Terminology in Abstracts [101]	Set of Keywords {(Modern, Peer, Code, Software, Patch) Review (Process), Pull Request, (Two-person, Code, Software, Peer) Inspection, Walkthrough}

reference to an existing dataset. Furthermore, as shown in Table 4, authors classified the papers according to the nature of the studied systems (i.e., open source projects or industry). Note that the classification is non-exclusive as some studies involved projects that were both open and closed data.

Terminology in Abstracts (RQ3). To classify topics of the papers, we used the technique similar to [101], where we extracted key terminology from Table 6. For this process, papers from the snowballing were used in the study. In the first round, we collected all terminology from the papers and then merged common terminology to get a final coding (i.e., shown in Table 1).

Table 5 Statistics of the filtering of the papers during the conduct search and screening process

		# of Papers	
Conduct Search			
	Search String Result	7,266	
All Papers			230
Screening of Papers			
	Conference paper	53	
	Journal paper	16	
	Snowball paper	79	
Relevant Papers			148

Table 6 Terminology extracted from the Title and Abstract from all Relevant Papers

Terminology	Rationale
(Code, Software, Peer) Inspection	The term 'inspection' is associated with more traditional forms of review. It can also refer to more vigorous or traditional forms of reviewing
(Modern, Peer, Code, Software, Patch) Review(s) (Process)	This terminology is common for research studying the contemporary review process and tools
Pull Request(s)	This terminology is common for contemporary git-based projects (i.e., GitHub)
Patch(es)	Patches is more associated with source-code related research
Other	Not includes any clear key words. These could be cross-cutting research.

The purpose of the first round is to remove non-important words. Then in a second round, we labeled each paper according to the coding. For the classification, one author was involved in the first round. Then other authors checked the final coding before we proceeded to the classification of the papers.

2.5 Data Extraction and Mapping of Studies

Using the classification scheme, we then utilize visual mappings of the results to highlight states in the collected papers. To identify which categories have been emphasized in past research and show possible opportunities for future work, we use three types to show maps (i) tables, (ii) line and bar plots, and (iii) bubble maps. Once the scheme is in place, we used excel spreadsheets to store the data and applied R scripts to extract and categorize the papers. Furthermore, we put rationales to decide why we believe each paper is categorized. Below are the visual techniques and rationale for answering each RQ:

Visual Map of RQ1. To answer RQ1, we show a visual mapping of the contributions (with the researcher and practitioners separately) against the methodologies. We intend to find out how the methodologies influence the contributions and what is the popular combination of contributions and methodologies.

A bubble map will be used to show results. The map should show what contributions are saturated and which perceived contributions have the potential for future work. We will also pick up examples of each classified papers for an in-depth discussion of the maps.

Visual Map of RQ2. To answer RQ2, we show a visual mapping of the replicability of the collected papers. We intend to determine how much CR research has the potential to be replicated.

A bar chart will be used to visualize the main results. The map should show the proportion of how many papers can be replicated and show what forms are used to provide replication (i.e., via links or reference to dataset). For a deeper understanding of the data sources, we perform additional sub-classification of the source:

- *CR Process*: researches extract data from pure code review tools (e.g., Gerrit tools in OSS and special review systems or tools in industry such as CodeFlow tool in Microsoft).
- *Software Development Process*: researches extract data that not only contains CR, but expands on other software development tools such as mailing lists, version control system, GitHub and issue tracking system.
- *Interviews, Survey, or a Controlled Study*: researches extract data from observational experiments in the form of interviews, survey and control study.

Finally, we also take a deeper look at the collected datasets to understand which project is the most used especially for the source from *CR Process*.

Visual Map of RQ3. To answer RQ3, we show a visual mapping of the selected terminology over time. We intend to find out how the terminology has changed over time.

A line plot combined with the timeline of terminology will be used to visualize the main results. The map should show the changes in terminology. Furthermore, we will use a combination of examples and important technological advancements (i.e., emergence of GitHub), to show relation to the terminology.

Visual Map of RQ4. To answer RQ4, we show a visual mapping of conference and journal locations where CR research has been published. We intend to find out the trends of published locations and show the most impactful academic software engineering domain for CR works.

We use two visual maps to answer the research question. The first map is a timeline analysis of the number of papers published in either conferences or journals. The second map is a bubble chart to show details of each premium location in SE research.

3 Maps of CR Research

We now present the results of the mapping study. The results will answer the research questions, with the visual maps of the categories of the papers from the mapping study.

3.1 (RQ1): What contributions and methodologies does CR research target?

> *"CR research published in premium SE venues use sound evaluation and validation methodologies, targeting particularly socio-technical and understanding of CR processes. On the other hand, there is a lack of papers that propose solutions to deal with CR problems."*

Figure 4 shows both the saturation of papers as well as the potential research opportunities for the field. The figure clearly shows that evaluation methodology is most popular, benefiting both the practitioner and researchers.

For practitioners, most of the papers have contributions to potential benefits and understanding aspects. Potential benefits mean that modern CR provides benefits beyond the fundamental need to find defects. CR is demonstrated to be useful for other tasks. We introduce three examples in detail below. For the task of improving code style, Zhang et al. [154] presented an interactive approach named *CRITICS* for inspecting systematic changes and the results show that it should improve developer productivity during this process. For the task of increasing the learning, Gousios et al. [50] conducted a large-scale survey to investigate work practices and challenges in pull-based development model and results show that integrator should consider several factors in their decision making. For the task of review comments usefulness, Rahman et al. [106] found that useful comments share more vocabulary with the changed code, contain salient items like relevant code elements, and their reviewers are generally more experienced. For instance, exploring how CR is conducted can be used for practitioners to better implement review activity and improve the review quality. Understanding is when reviewers have prior knowledge of the context and the code, they complete reviews more quickly and provide more valuable feedback to the author. Key examples are researches that look into the quality of review and types of defects. Kononenko et al. [73] provided a deep insight into how developers define review quality, what factors contribute to how they evaluate submitted code, and what challenges they face when performing review tasks. Beller et al. [21] conducted a manual research to increase the understanding of practical benefits that the MCR process produces on reviewed source code. Their results show that types of changes due to the MCR process in OSS are strikingly similar to those in the industry and academic systems.

On the other hand, researcher-oriented CR studies mostly focus on Socio-technical contributions. Socio-technical related papers are studies that involve the consideration of both human and technical aspects. One popular topic is reviewers recommendation and many studies have been done on this topic. Xia et al. [148] put textual information and file location analyses together to recommend reviewers more accurately. Hannebauer et al. [55] recommended code reviewers based on their expertise. Apart from reviewer recommendation topic, many other human related researches have been conducted such as review participation [133], evaluation of contributions [135] and broadcast during CR process [110]. In terms of CR, studies can be designed and carried out to determine if and how team collaboration, coordination, awareness and learning occurs. In terms of opportunities, Figure 4 highlights the lack of experience papers. This is crucial and shows a lack of reporting and feedback from developers. Instead, we see that there are a stable number of survey papers. Other notable potential methodologies are the experience and solution, which indicates that more practical tools need to be developed to help practitioners in reality.

Table 7 shows the top five paper contributions with the methodologies used, illustrating how evaluation studies are dominant. According to our results, two most popular combinations are the evaluation study that has a socio-technical contribution (e.g., 35 papers), then followed by the study with understanding target following the evaluation methodology. One example of the evaluation study with a social-technical effect is Thongtanunam et al. [130], which investigates CR practices in defective files combined with human factors (e.g., the participation of the reviewer in the process). In detail, authors evaluate the results using a detailed empirical study of the Gerrit review system within the Qt project. Similarly, Kononenko et al. [72] reported on a case study investigating CR quality for Mozilla and explore the relationships between the reviewers code inspections and a set of factors, both personal and social. Another popular combination is the understanding con-

Table 7 Top 5 combination of contribution and methodology

		Contribution		
		Socio-technical effects	Understanding	Potential Benefits
Methodology	Evaluation	[20, 21, 32, 43, 49, 58, 63–65, 68, 72, 74, 75, 83–85, 89, 106, 109, 110, 113, 118, 122, 123, 130, 132–135, 142, 148, 151, 155, 157]	[17, 20, 21, 23, 32, 49, 52, 58, 63, 65, 72, 74, 80, 83–85, 89, 96, 109, 110, 113, 118, 122, 123, 128, 130, 132–134, 139, 142, 151, 155, 157]	[20, 31, 43, 52, 64, 68, 80, 85, 106, 109, 113, 126, 148, 151]
	Validation	[43, 64, 68, 83–85, 89, 95, 105, 106, 130, 132, 133, 148, 151]		[5, 10, 11, 43, 55, 64, 68, 85, 87, 95, 105, 106, 148, 151, 153, 154]

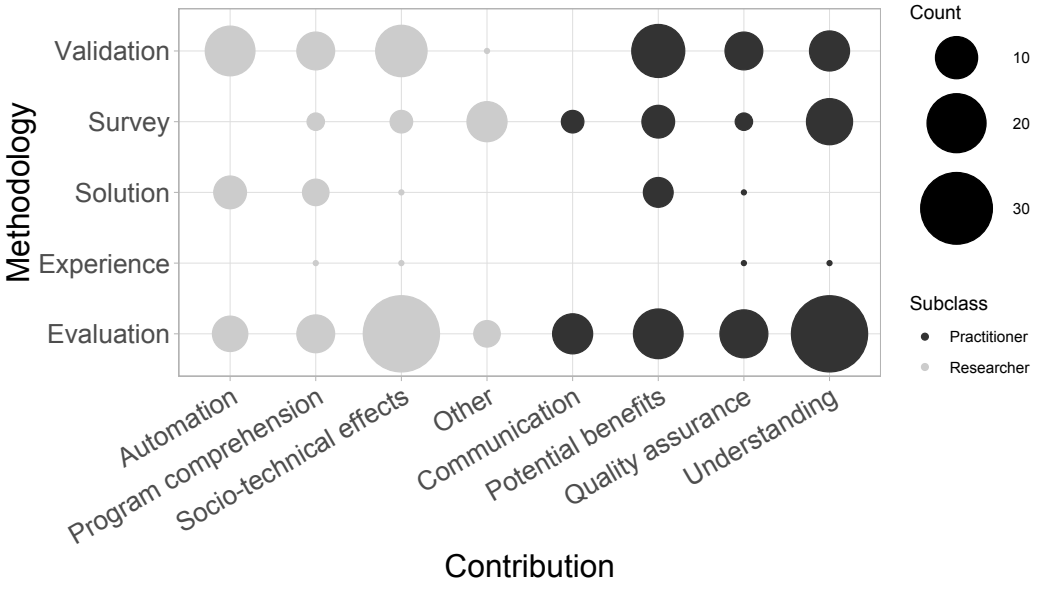


Fig. 4 Visual Map for RQ_1, showing the contribution and methodology of CR research

tribution, using the evaluation methodology. It is interesting to note that 27 out of 35 papers that contribute to better understanding also have socio-technical contributions. For example, Thongtanunam et al. [133] studied what factors influence review participation in the CR process which in turn helps practitioners understand the situation when they tend to join.

As shown in Table 7, the third popular combination of contributions and methodologies for CR research is papers that target contributions of potential benefits and have the validation methodology. The majority of the validation papers are based on recommendation or prediction models. An example of this type of paper is Rahman et al. [105], where authors suggest an approach of reviewer recommendation based on cross-project and technology experience.

3.2 (RQ2): How much CR research has the potential for replicability?

> *"The Gerrit code review tool has revolutionized CR research, with over 49% of CR papers have the possibility to be replicable. Yet, we find that only 17% of papers refer to available sources of datasets."*

Table 8 Data Source Classifications

Data source	Relevant Papers
CR Process	[5, 10, 11, 17, 21, 31, 32, 43, 55, 83–85, 95, 96, 106, 109, 115, 117, 118, 123, 130, 132, 133, 145, 148, 153]
Software Dev. Process	[5, 17, 20, 23, 49, 52, 55, 58, 63, 64, 68, 72, 74, 80, 83, 87, 89, 105, 109, 110, 113, 118, 122, 126, 128, 134, 139, 142, 151, 155, 157]
Inter./Sur./Control Study	[8, 11, 12, 16, 26, 33, 45, 47, 50, 51, 73, 75, 76, 107, 115, 117, 127, 128, 135, 145, 147, 154]

Table 9 Top 3 Studied Projects in Gerrit

Gerrit project	Relevant Papers
Qt	[55, 83–85, 95, 123, 130, 132, 133, 148]
OpenStack	[43, 55, 83, 95, 123, 132, 133, 148]
Android	[17, 31, 95, 118, 133, 148, 153]

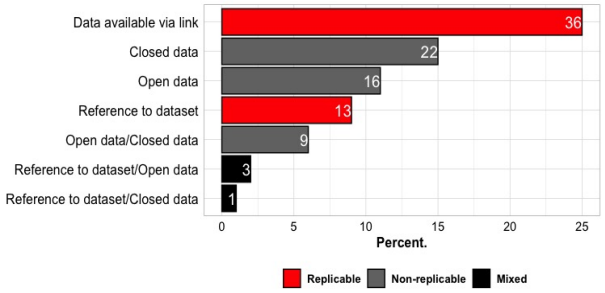


Fig. 5 Visual Map for RQ_2, showing replicability of the collected papers

Table 8 and Figure 5 show the data source classifications and the visual map for replicability of the papers. The results show how many papers in CR have the potential to be replicated and summarize the sources of datasets.

In Table 8, we divide all premium papers into different classification according to the definition of data sources shown in Section 2.5. We describe each data source in detail below. In *CR Process*, for example, Mcintosh et al. [85] conducted the research to see the impact of code reviews on software quality. They only focus on review process and extract the data from QT, VTK, ITK projects using review tools (e.g., Gerrit). For *Software Development Process*, for example, Kononenko et al. [72] investigated whether people and participation matter the quality of review. In their research, they collected data from issue tracking system (e.g., Bugzilla) which belongs to the development process. In *Interview/Survey/Control Study*, for instance, Bosu et al. [33] analyzed the process aspects and social dynamics of CR from the diverse surveys of Mircosoft and other open source projects. In another example, Floyd et al. [45] researched the representation of code in the brain with fMRI study. They involved 29 participants to carry out the controlled experiment and got result feedback. We find that *code review process* related dataset is the most extracted from the well-studied Gerrit tool. One advancement has been the release of the rest API, in which anyone is able to download and collect data on projects. As shown in Table 9, we summarize and draw the top 3 popular projects using Gerrit tools. We observe for these CR papers, Qt project is the most studied project, with ten, eight and seven papers researching Qt, OpenStack and Android respectively.

Figure 5 shows two important findings with the proportion of replicability. The first is that there is up to 47% (i.e., 22% of closed data, 16% of open data and 9% of open data/closed data) of papers that do not provide any access to their datasets. Taking a closer look at the closed data, studies are usually conducted within industries, surveys and control studies. An example of this paper is Balachandran [10], where the authors conducted research on how to reduce human efforts and improve review quality using the data from industry project named VMware. For papers that labeled as open data, the researchers collected data from open source projects but did not share a replication package. For instance, Mirhosseini and Parnin [89] investigated whether or not pull requests encourage developers to upgrade out-of-date dependencies with the data from OSS (i.e., 7,470 projects in GitHub). It could be argued that since the data is open source and available for anyone to download themselves. On the other hand, as shown in Figure 5, we find that 49% of the studies that released a replication package, either referred to a published dataset or released their own dataset via an online link. For the work of Thongtanunam et al. [132], authors referred to a dataset that was previously published [54] to revisit code ownership and its relationship with software quality. Usually, papers release a link to the dataset. For instance, Baysal et al. [20] shared the dataset link (e.g., WebKit and Blinkin projects).

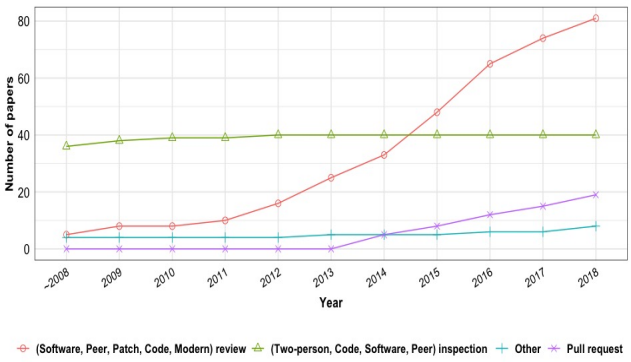


Fig. 6 Visual Map for RQ_3, showing changes of terminology used in CR research

3.3 (RQ3): How has CR terminology changed over time?

> *"CR terminology has changed to correspond with the technology. Research has moved from the older terminology named 'inspection' to more modern*

Table 10 Classification of Terminology for all Relevant Papers

Terminology	Relevant Papers
(Two-person, Code, Software, Peer) Inspection / Walkthrough	[3, 4, 7, 24, 25, 27, 34, 35, 37–42, 44, 46, 48, 57, 61, 66, 71, 77–79, 82, 86, 98, 100, 102–104, 114, 119, 120, 125, 129, 137, 138, 147]
(Modern, Peer, Code, Software, Patch) Review(s)	[5, 8, 10–21, 26, 28–33, 36, 43, 45, 47, 52, 53, 55, 56, 58, 59, 65, 68, 69, 72, 73, 76, 84, 85, 87, 88, 91–93, 95–97, 105–113, 115, 117, 123, 124, 126–128, 130–133, 136, 139, 140, 142, 145, 148, 149, 153, 154]
Pull Request(s)	[23, 49–51, 62–64, 74, 89, 121, 122, 134, 135, 150–152, 155–157]
Other	[80, 81, 83, 90, 99, 118, 143, 144]

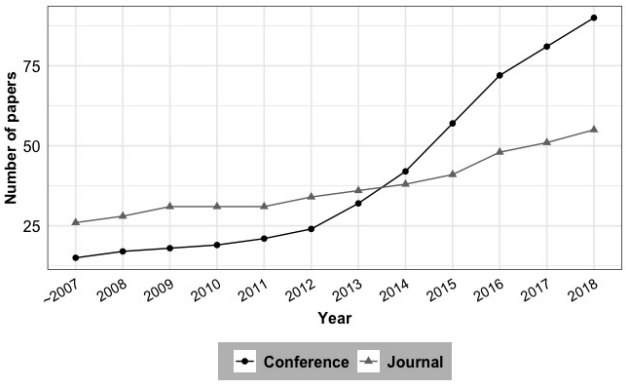
terminology like 'code review' and 'pull request' in GitHub projects."

Figure 6 shows three key findings with the evolution of terminology related to CR. First, from 2012 onwards, the papers whose terminology including 'inspection' reached a saturation. The results suggest that researchers and practitioners moved away from the traditional terminology of 'inspection', which has been widely used for more than 30 years [42]. As stated in Table 10, 'inspection' terminology is more easily associated with the more traditional forms of inspection. For example, in 1976, Fagan [42] proposed the first well known inspection model. In 1989, Bisant and Lyle [27] introduced another important method called 'Two-Person inspection'.

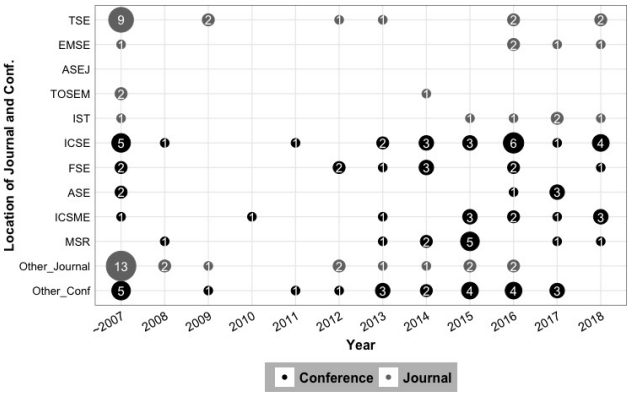
Second, we observe that the terminology called 'code review' first appeared around 2006 year. Moreover, the concept of modern code review which is a light variant of traditional review process is proposed since the 2013 year. This terminology is speculated to be adapted from the contemporary review tools such as Gerrit. These tools originally started from Guido Van Rossum to serve as a practical tool for the Python developer community, and hopefully for other open source communities. During his time at Google, he developed the industrial version named Mondrian. He found that proper code review habits can really improve the quality of a code base, and at the same time good tools for code review will improve developers' life. Review started the revolution into using online review tools. Furthermore, with the available API, researchers could download and use these data.

Third, as shown in the figure, it's clear to see that the rate of the terminology called 'pull request' is increasing fast with the year from 2015. As stated by GitHub[5], pull requests are proposed changes to a repository submitted by a user and they will be accepted or rejected by a repository's collaborators. Like issues, pull requests each have their own discussion forum and serve as a review of any changes to GitHub projects.

3.4 (RQ4): Which SE locations include papers on CR research?



(a) The Changing trends of CR research being published in journals to conferences



(b) Location of papers for conference and journal

Fig. 7 Visual Map for RQ_4, showing the conference and journal papers in CR research

"CR Research has moved from being published in journals to top SE conferences. Furthermore, the research area has moved into the premium locations compared to research before 2008."

[5] `https://help.github.com/en/articles/about-pull-requests`

Table 11 Description of three related studies aiming to summarize CR research.

Related Study	Published	Publication Series	Period Covered	Summary
Badampudi et al. [9]	2019	Evaluation and Assessment in Software Engineering (EASE)	2005-2018	Preliminary results that explore existing literature on modern code review.
Kollanus and Koskinen [70]	2009	The Open Software Engineering Journal	1980-2008	Present the survey of the software inspection research published during 1980-2008.
Aurum et al. [6]	2002	Software Testing, Verification and Reliability	1976-2001	Provide an overview of the software inspection processes that have emerged in the last 25 years.

Figure 7 shows the visual maps that are used to answer RQ4. From the figure, we have two important findings that are related to (a) the changing from journals to conferences and (b) the movement of CR research into the premium locations in SE.

As shown in Figure 7a, CR research has seen an increase in conference publications. The number of papers is increasing consistently from 2010 to 2016 and it reaches the peak (i.e., around 22 papers) in 2016. Figure 7 shows the location of conference papers over the time period. In a more deeper analysis, we see that in publications where top conferences like the top-tier International Conference on Software Engineering (ICSE), the Foundations of Software Engineering (FSE) and the Automatic Software Engineering (ASE) have become popular in recent times. This can be accredited to technical vigor of recent CR research, especially in terms of evaluation and contribution. Corresponding with the results of RQ1 and RQ2, the answer could be that the evaluation and datasets have been improved so much that the paper qualities have been worthy of such locations.

4 The State of CR Research

The key distinction between our study to the other mapping studies [6, 9, 70] is venue selection in the mapping process. There does exist a very similar preliminary study on MCR by Badampudi et al. [9], but their scope covers a wider-range of venues. Also different to that work is the focus on the changes of the methodologies, contributions, and replicability. To increase our scope of papers, we included a snowball approach.

There are two older systematic mapping studies carried out on CR research. Kollanus and Koskinen [70] conducted an older study 10 years ago. This study thus does not include more recent CR work. Aurum et al. [6] is also outdated, yet very related mapping study. It details the early works up to 2001 on CR. This study highlights the benefits of using the Fagan inspection process, which is a more traditional method compared to the Modern CR. A summary of all these three studies is presented in Table 11.

From this study, we outline three implications learned from the study. Furthermore, we identify mature aspects and gaps for potential future avenues of research.

1. CR research contributions are broader than quality assurance. As shown in RQ1 and in a survey conducted by Bacchelli and Bird [8], CR research covers a wider range of contributions, which is beyond quality assurance. With the recent wide-adoption of MCR and pull request model in both Open Source and industry giants like Google and Microsoft, more developers are involved in the review process and discuss the code change online. Compared to traditional CR, apart from fixing defects, developers are also interested in learning and sharing knowledge. CR research nowadays tends to be more connected with social factors and communications.

Mature contributions, methodologies, and gaps. Due to the human-centric nature of code review, it is no surprise that there has been much work conducted in understanding and social-technical aspects of CR. Furthermore, the evaluation and validation methodology are two most widely used methodologies to conduct CR research, especially work published in the top venues. Though according to Badampudi et al. [9], 39 solution papers are published from 2005 to 2018, in our study we don't find a number of such papers appearing in premium venues. This evidence may indicate that the tools have not reached maturity by the research community.

2. CR research is highly dependent on data-driven studies. With around 75% of our collected papers being quantitative, datasets now play a vital role in CR research. The introduction of the Gerrit tools provides developers to have an offline review discussion. Furthermore, the data has been made available for use by both researchers and practitioners.

Mature datasets and gaps. Although a lot of researches have been conducted on the similar datasets of

Android, Qt and Openstack projects, we observe that the replicability still has room for improvement, with around 50% of studies not providing replicable datasets. In fact, we observe that fewer studies referred to an existing dataset. The datasets had some unique aspect to the prior works. The implication for researchers and practitioners is that CR research will continue to rely on the data collected from review tools. In the future, we encourage researchers to acknowledge related datasets, while making their data available for future research.

3. CR research terminology evolves with technology and needs of developers. With the explosion of mining software repositories techniques, researchers are now able to carry out empirical studies on the CR process. As shown in RQ3, the terminology and technology influence the research. For example, the current CR researches focus on GitHub's 'pull request' function. We envision that the CR will continue to evolve with the technology.

Mature and emerging technologies . Since 2010, CR research has evolved in its terminology, matching the popularity of the MCR tools. To further validate this point, we notice due to the recent use of GitHub's 'pull request', as shown in RQ3, research into pull request has gained much attention since 2014. Furthermore, an interesting gap in the research is to understand whether or not older Fagan-style CR is still in practice. Overall, the results suggest that CR research will continue as long as there are emerging technologies.

5 Threats To Validity

Key threats to validity in this systematic review are three-fold: our selection of the studies to be included, correct classification, and potential author bias.

During the screening of papers, to cover the large corpus of collected papers, our initial step includes the first author scanning through, and discarding papers based on titles, potentially raising a bias in the paper selection. We are confident of this threat, as the first author is an existing code review researcher and is familiar with the domain. Furthermore, doubtful papers were flagged and discussed with the other two authors for clarification. For the clarification stage, abstracts and sometimes the whole paper was read by three authors to validate a good degree of consensus. In cases where there was disagreement, the issue was discussed until consensus was reached.

The second threat is the potential threat of incorrect classification of the papers. Similar to the inclusion paper method threat, we use the same clarification process to validate a good degree of consensus.

The final threat is a potential bias in that authors have written papers that were included in the review. This threat does not exist in this study because no collected paper is co-authored together.

6 Conclusion

CR research will continue to evolve with the availability of datasets and emerging technologies within the CR domain. Presenting different visual maps of the 148 relevant papers, we show that CR research has indeed gained popularity and covers more than just the quality assurance of source code. This study based on the current established state of CR research is useful for both researchers and practitioners. There are many open avenues for future work: further studies to fill in the gaps on existing work, find out whether older technologies are still used in practice, and understanding what is the next emerging trend in CR research.

Acknowledgment

This work has been supported by Japan Society for the Promotion of Science (JSPS) KAKENHI Grant Numbers JP16H05857, JP17H00731, and JP18KT0013.

7 Appendix: Relevant Papers

See Table 12, 13 and Table 14 for the full listing of the relevant papers.

References

1. B. A. Kitchenham. Kitchenham, b.: Guidelines for performing systematic literature reviews in software engineering. ebse technical report ebse-2007-01, 01 2007.
2. U. Abelein and B. Paech. Understanding the influence of user participation and involvement on system success - a systematic mapping study. In Software Engineering 2016, pages 85–86, 2016.
3. A. F. Ackerman, P. J. Fowler, and R. G. Ebenau. Software inspections and the industrial production of software. In Proc. Of a Symposium on Software Validation: Inspection-testing-verification-alternatives, pages 13–40, 1984.
4. A. F. Ackerman, L. S. Buchwald, and F. H. Lewski. Software inspections: An effective verification process. IEEE Softw., 6(3):31–36, May 1989. ISSN 0740-7459.
5. M. Allamanis, E. T. Barr, C. Bird, and C. Sutton. Learning natural coding conventions. In Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014, pages 281–293, 2014.
6. A. Aurum, H. Petersson, and C. Wohlin. State-of-the-art: software inspections after 25 years. Softw. Test., Verif. Reliab., 12:133–154, 2002.
7. A. Aurum, H. Petersson, and C. Wohlin. State-of-the-art: software inspections after 25 years. Softw. Test., Verif. Reliab., 12:133–154, 2002.
8. A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, pages 712–721, 2013.
9. D. Badampudi, R. Britto, and M. Unterkalmsteiner. Modern code reviews - preliminary results of a systematic mapping study. In Proceedings of the Evaluation and Assessment on Software Engineering, EASE '19, pages 340–345. ACM, 2019. ISBN 978-1-4503-7145-2.
10. V. Balachandran. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, pages 931–940, 2013.
11. M. Barnett, C. Bird, J. a. Brunet, and S. K. Lahiri. Helping developers help themselves: Automatic decomposition of code review changesets. In Proceedings of the 37th International Conference on Software Engineering - Volume 1, ICSE '15, pages 134–144, 2015.
12. T. Baum, O. Liskin, K. Niklas, and K. Schneider. Factors influencing code review processes in industry. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, pages 85–96, 2016.
13. T. Baum, O. Liskin, K. Niklas, and K. Schneider. A faceted classification scheme for change-based industrial code review processes. In 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS), pages 74–85, 2016.
14. T. Baum, e. P. Schneider, Kurt", A. Jedlitschka, A. Nguyen Duc, M. Felderer, S. Amasaki, and T. Mikkonen. On the need for a new generation of code review tools. In Product-Focused Software Process Improvement, pages 301–308, 2016.
15. T. Baum, H. Leßmann, e. M. Schneider, Kurt", D. Méndez Fernández, B. Turhan, M. Kalinowski, F. Sarro, and D. Winkler. The choice of code review process: A survey on the state of the practice. In Product-Focused Software Process Improvement, pages 111–127, 2017.
16. T. Baum, K. Schneider, and A. Bacchelli. On the optimal order of reading source code changes for review. In 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 329–340, 2017.
17. G. Bavota and B. Russo. Four eyes are better than two: On the impact of code reviews on software quality. In 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 81–90, 2015.
18. O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey. The secret life of patches: A firefox case study. In Proceedings of the 2012 19th Working Conference on Reverse Engineering, WCRE '12, pages 447–455, 2012.
19. O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey. The influence of non-technical factors on code review. In 2013 20th Working Conference on Reverse Engineering (WCRE), pages 122–131, 2013.
20. O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey. Investigating technical and non-technical factors influencing modern code review. Empirical Software Engineering, 21, 2015.
21. M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens. Modern code reviews in open-source projects: Which problems do they fix? In Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, pages

202–211, 2014.
22. H. Bernard. Research Methods in Anthropology: Qualitative and Quantitative Approaches. Rowman & Littlefield, 01 2011.
23. J. a. H. Bernardo, D. A. da Costa, and U. Kulesza. Studying the impact of adopting continuous integration on the delivery time of pull requests. In Proceedings of the 15th International Conference on Mining Software Repositories, MSR '18, pages 131–141, 2018.
24. A. Bianchi, F. Lanubile, and G. Visaggio. A controlled experiment to assess the effectiveness of inspection meetings. In Proceedings Seventh International Software Metrics Symposium, pages 42–50, 2001.
25. S. Biffl, P. Grünbacher, and M. Halling. A family of experiments to investigate the effects of groupware for software inspection. Automated Software Engineering, 13(3):373–394, Jul 2006.
26. C. Bird, T. Carnahan, and M. Greiler. Lessons learned from building and deploying a code review analytics platform. In Proceedings of the 12th Working Conference on Mining Software Repositories, MSR '15, pages 191–201, 2015.
27. D. B. Bisant and J. R. Lyle. A two-person inspection method to improve programming productivity. IEEE Trans. Softw. Eng., pages 1294–1304, Oct. 1989.
28. A. Bosu and J. C. Carver. Peer code review in open source communitiesusing reviewboard. In Proceedings of the ACM 4th Annual Workshop on Evaluation and Usability of Programming Languages and Tools, PLATEAU '12, pages 17–24, 2012.
29. A. Bosu and J. C. Carver. Impact of peer code review on peer impression formation: A survey. In 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, pages 133–142, 2013.
30. A. Bosu and J. C. Carver. Impact of developer reputation on code review outcomes in oss projects: An empirical investigation. In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '14, pages 33:1–33:10, 2014.
31. A. Bosu, J. C. Carver, M. Hafiz, P. Hilley, and D. Janni. Identifying the characteristics of vulnerable code changes: An empirical study. In Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014, pages 257–268, 2014.
32. A. Bosu, M. Greiler, and C. Bird. Characteristics of useful code reviews: An empirical study at microsoft. In Proceedings of the 12th Working Conference on Mining Software Repositories, MSR '15, pages 146–156, 2015.
33. 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 Transactions on Software Engineering, pages 56–75, 2017.
34. L. Brothers, V. Sembugamoorthy, and M. Muller. Icicle: Groupware for code inspection. In Proceedings of the 1990 ACM Conference on Computer-supported Cooperative Work, CSCW '90, pages 169–181, 1990.
35. C. Denger and F. Shull. A practical approach for quality-driven inspections. IEEE Softw., 24(2):79–86, Mar. 2007. ISSN 0740-7459.
36. M. di Biase, M. Bruntink, and A. Bacchelli. A security perspective on code review: The case of chromium. In 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), pages 21–30, 2016.
37. A. Dunsmore, M. Roper, and M. Wood. Object-oriented inspection in the face of delocalisation. In Proceedings of the 22Nd International Conference on Software Engineering, ICSE '00, pages 467–476, 2000.
38. A. Dunsmore, M. Roper, and M. Wood. Systematic object-oriented inspection - an empirical study. In Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001, pages 135–144, 2001.
39. A. Dunsmore, M. Roper, and M. Wood. Practical code inspection techniques for object-oriented systems: An experimental comparison. IEEE Softw., pages 21–29, July 2003. ISSN 0740-7459.
40. A. Dunsmore, M. Roper, and M. Wood. The development and evaluation of three diverse techniques for object-oriented code inspection. IEEE Transactions on Software Engineering, pages 677–686, 2003.
41. M. E. Fagan. Design and code inspections to reduce errors in program development. IBM Systems Journal, 15(3):182–211, 1976.
42. M. E. Fagan. Advances in software inspections. IEEE Transactions on Software Engineering, pages 744–751, 1986.
43. Y. Fan, X. Xia, D. Lo, and S. Li. Early prediction of merged code changes to prioritize reviewing tasks. Empirical Softw. Engg., pages 3346–3393, 2018.

44. A. L. Ferreira, R. J. Machado, J. G. Silva, R. F. Batista, L. Costa, and M. C. Paulk. An approach to improving software inspections performance. In 2010 IEEE International Conference on Software Maintenance, pages 1–8, 2010.
45. B. Floyd, T. Santander, and W. Weimer. Decoding the representation of code in the brain: An fmri study of code review and expertise. In Proceedings of the 39th International Conference on Software Engineering, ICSE '17, pages 175–186. IEEE Press, 2017.
46. M. V. Genuchten, W. Cornelissen, and C. V. Dijk. Supporting inspections with an electronic meeting system. Journal of Management Information Systems, 14(3):165–178, 1997.
47. D. M. German, G. Robles, G. Poo-Caamaño, X. Yang, H. Iida, and K. Inoue. "was my contribution fairly reviewed?": A framework to study the perception of fairness in modern code reviews. In Proceedings of the 40th International Conference on Software Engineering, ICSE '18, pages 523–534. ACM, 2018.
48. J. Gintell, J. Arnold, M. Houde, J. Kruszelnicki, R. McKenney, and G. Memmi. Scrutiny: A collaborative inspection and review system. In Proceedings of the 4th European Software Engineering Conference on Software Engineering, ESEC '93, pages 344–360, 1993.
49. G. Gousios, M. Pinzger, and A. v. Deursen. An exploratory study of the pull-based software development model. In Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, pages 345–355, 2014.
50. G. Gousios, A. Zaidman, M.-A. Storey, and A. van Deursen. Work practices and challenges in pull-based development: The integrator's perspective. In Proceedings of the 37th International Conference on Software Engineering - Volume 1, ICSE '15, pages 358–368. IEEE Press, 2015.
51. G. Gousios, M.-A. Storey, and A. Bacchelli. Work practices and challenges in pull-based development: The contributor's perspective. In Proceedings of the 38th International Conference on Software Engineering, ICSE '16, pages 285–296, 2016.
52. M. Gupta, A. Sureka, and S. Padmanabhuni. Process mining multiple repositories for software defect resolution from control and organizational perspective. In Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, pages 122–131, 2014.
53. A. Gyori, S. K. Lahiri, and N. Partush. Refining interprocedural change-impact analysis using equivalence relations. In Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2017, pages 318–328. ACM, 2017. ISBN 978-1-4503-5076-1.
54. K. Hamasaki, R. G. Kula, N. Yoshida, A. E. C. Cruz, K. Fujiwara, and H. Iida. Who does what during a code review? datasets of oss peer review repositories. In Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, pages 49–52, 2013. ISBN 978-1-4673-2936-1.
55. C. Hannebauer, M. Patalas, S. Stünkel, and V. Gruhn. Automatically recommending code reviewers based on their expertise: An empirical comparison. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, pages 99–110, 2016.
56. L. Harjumaa, I. Tervonen, and A. Huttunen. Peer reviews in real life - motivators and demotivators. In Fifth International Conference on Quality Software (QSIC'05), pages 29–36, 2005.
57. L. Hatton. Testing the value of checklists in code inspections. IEEE Software, 25(4):82–88, 2008.
58. V. J. Hellendoorn, P. T. Devanbu, and A. Bacchelli. Will they like this?: Evaluating code contributions with language models. In Proceedings of the 12th Working Conference on Mining Software Repositories, MSR '15, pages 157–167, 2015.
59. T. Hirao, A. Ihara, Y. Ueda, P. Phannachitta, e. K. Matsumoto, Ken-ichi", I. Hammouda, B. Lundell, G. Robles, J. Gamalielsson, and J. Lindman. The impact of a low level of agreement among reviewers in a code review process. In Open Source Systems: Integrating Communities, pages 97–110, 2016.
60. D. Izquierdo-Cortazar, N. Sekitoleko, J. M. Gonzalez-Barahona, and L. Kurth. Using metrics to track code review performance. In Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, EASE'17, pages 214–223, 2017.
61. P. Jalote and M. Haragopal. Overcoming the nah syndrome for inspection deployment. In Proceedings of the 20th International Conference on Software Engineering, pages 371–378, 1998.
62. J. Jiang, J.-H. He, and X.-Y. Chen. Coredevrec: Automatic core member recommendation for contribution evaluation. Journal of Computer Science and Technology, 30(5):998–1016, Sep 2015.
63. J. Jiang, D. Lo, X. Ma, F. Feng, and L. Zhang. Understanding inactive yet available assignees in github. Inf. Softw. Technol., pages 44–55, 2017.

64. 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., pages 48–62, 2017.
65. Y. Jiang, B. Adams, and D. M. German. Will my patch make it? and how fast? case study on the linux kernel. In 2013 10th Working Conference on Mining Software Repositories (MSR), pages 101–110, 2013.
66. P. M. Johnson and D. Tjahjono. Does every inspection really need a meeting? Empirical Software Engineering, pages 9–35, Mar 1998.
67. P. M. Johnson and D. Tjahjono. Does every inspection really need a meeting? Empirical Software Engineering, pages 9–35, Mar 1998.
68. Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi. A large-scale empirical study of just-in-time quality assurance. IEEE Transactions on Software Engineering, pages 757–773, 2013.
69. C. F. Kemerer and M. C. Paulk. The impact of design and code reviews on software quality: An empirical study based on psp data. IEEE Trans. Softw. Eng., pages 534–550, July 2009.
70. S. Kollanus and J. Koskinen. Survey of software inspection research. The Open Software Engineering Journal, 3, 05 2009. doi: 10.2174/1874107X00903010015.
71. S. Kollanus and J. Koskinen. Survey of software inspection research. The Open Software Engineering Journal, 3, 05 2009. doi: 10.2174/1874107X00903010015.
72. O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey. Investigating code review quality: Do people and participation matter? In 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 111–120, 2015.
73. O. Kononenko, O. Baysal, and M. W. Godfrey. Code review quality: How developers see it. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pages 1028–1038, 2016.
74. 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 Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP '18, pages 124–133. ACM, 2018.
75. V. Kovalenko, N. Tintarev, E. Pasynkov, C. Bird, and A. Bacchelli. Does reviewer recommendation help developers? IEEE Transactions on Software Engineering, Aug. 2018.
76. S. Krusche, M. Berisha, and B. Bruegge. Teaching code review management using branch based workflows. In Proceedings of the 38th International Conference on Software Engineering Companion, ICSE '16, pages 384–393. ACM, 2016.
77. O. Laitenberger and J.-M. DeBaud. An encompassing life cycle centric survey of software inspection. J. Syst. Softw., pages 5–31, Jan. 2000.
78. F. Lanubile, T. Mallardo, and F. Calefato. Tool support for geographically dispersed inspection teams. Software Process: Improvement and Practice, 8(4):217–231, 2003.
79. F. Macdonald and J. Miller. A comparison of computer support systems for software inspection. Automated Software Engg., pages 291–313, July 1999.
80. I. Malavolta, R. Verdecchia, B. Filipovic, M. Bruntink, and P. Lago. How maintainability issues of android apps evolve. In 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 334–344, Sep. 2018.
81. J. Marlow, L. Dabbish, and J. Herbsleb. Impression formation in online peer production: Activity traces and personal profiles in github. In Proceedings of the 2013 Conference on Computer Supported Cooperative Work, CSCW '13, pages 117–128, 2013.
82. V. Mashayekhi, C. Feulner, and J. Riedl. Cais: Collaborative asynchronous inspection of software. In Proceedings of the 2Nd ACM SIGSOFT Symposium on Foundations of Software Engineering, SIGSOFT '94, pages 21–34, 1994.
83. S. McIntosh and Y. Kamei. Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction. IEEE Transactions on Software Engineering, pages 412–428, 2018.
84. S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, pages 192–201, 2014.
85. S. Mcintosh, Y. Kamei, B. Adams, and A. E. Hassan. An empirical study of the impact of modern code review practices on software quality. Empirical Softw. Engg., 21:2146–2189, 2016. ISSN 1382-3256.

86. D. A. McMeekin, B. R. von Konsky, E. Chang, and D. J. A. Cooper. Evaluating software inspection cognition levels using bloom's taxonomy. In 2009 22nd Conference on Software Engineering Education and Training, pages 232–239, 2009.
87. M. Menarini, Y. Yan, and W. G. Griswold. Semantics-assisted code review: An efficient toolchain and a user study. In Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, pages 554–565, 2017.
88. A. Meneely, A. C. R. Tejeda, B. Spates, S. Trudeau, D. Neuberger, K. Whitlock, C. Ketant, and K. Davis. An empirical investigation of socio-technical code review metrics and security vulnerabilities. In Proceedings of the 6th International Workshop on Social Software Engineering, SSE 2014, pages 37–44, 2014.
89. S. Mirhosseini and C. Parnin. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, pages 84–94, 2017.
90. A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and mozilla. ACM Trans. Softw. Eng. Methodol., pages 309–346, July 2002.
91. 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 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pages 171–180, 2015.
92. M. V. Mntyl and C. Lassenius. What types of defects are really discovered in code reviews? IEEE Transactions on Software Engineering, pages 430–448, 2009.
93. M. Nurolahzade, S. M. Nasehi, S. H. Khandkar, and S. Rawal. The role of patch review in software evolution: An analysis of the mozilla firefox. In Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops, IWPSE-Evol '09, pages 9–18, 2009.
94. R. S. Oshana. Tailoring cleanroom for industrial use. IEEE Software, pages 46–55, 1998.
95. A. Ouni, R. G. Kula, and K. Inoue. Search-based peer reviewers recommendation in modern code review. In 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 367–377, 2016.
96. M. Paixao, J. Krinke, D. Han, C. Ragkhitwetsagul, and M. Harman. Are developers aware of the architectural impact of their changes? In Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, pages 95–105, 2017.
97. S. Panichella, V. Arnaoudova, M. Di Penta, and G. Antoniol. Would static analysis tools help developers with code reviews? In 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pages 161–170, 2015.
98. D. L. Parnas and M. Lawford. The role of inspection in software quality assurance. IEEE Transactions on Software Engineering, 29(8):674–676, 2003.
99. D. L. Parnas and D. M. Weiss. Active design reviews: Principles and practices. J. Syst. Softw., 7 (4):259–265, Dec. 1987. ISSN 0164-1212.
100. D. E. Perry, A. Porter, M. W. Wade, L. G. Votta, and J. Perpich. Reducing inspection interval in large-scale software development. IEEE Trans. Softw. Eng., pages 695–705, July 2002.
101. K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson. Systematic mapping studies in software engineering. In Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE'08, pages 68–77, 2008.
102. A. Porter, H. Siy, A. Mockus, and L. Votta. Understanding the sources of variation in software inspections. ACM Trans. Softw. Eng. Methodol., pages 41–79, Jan. 1998.
103. A. Porter, H. Siy, and L. Votta. Understanding the effects of developer activities on inspection interval. In Proceedings of the 19th International Conference on Software Engineering, 11 1998.
104. A. A. Porter, H. P. Siy, C. A. Toman, and L. G. Votta. An experiment to assess the cost-benefits of code inspections in large scale software development. IEEE Trans. Softw. Eng., pages 329–346, 1997.
105. M. M. Rahman, C. K. Roy, and J. A. Collins. Correct: Code reviewer recommendation in github based on cross-project and technology experience. In Proceedings of the 38th International Conference on Software Engineering Companion, ICSE '16, pages 222–231. ACM, 2016.
106. M. M. Rahman, C. K. Roy, and R. G. Kula. Predicting usefulness of code review comments using textual features and developer experience. In Proceedings of the 14th International Conference on Mining Software Repositories, MSR '17, pages

215–226, 2017.

107. A. Ram, A. A. Sawant, M. Castelluccio, and A. Bacchelli. What makes a code change easier to review: An empirical investigation on code change reviewability. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2018, pages 201–212. ACM, 2018.
108. P. Rigby, B. Cleary, F. Painchaud, M. Storey, and D. German. Contemporary peer review in action: Lessons from open source development. IEEE Software, pages 56–61, 2012.
109. P. C. Rigby and C. Bird. Convergent contemporary software peer review practices. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013, pages 202–212, 2013.
110. P. C. Rigby and M.-A. Storey. Understanding broadcast based peer review on open source software projects. In Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, pages 541–550, 2011.
111. P. C. Rigby, D. M. German, and M.-A. Storey. Open source software peer review practices: A case study of the apache server. In Proceedings of the 30th International Conference on Software Engineering, ICSE '08, pages 541–550, 2008.
112. P. C. Rigby, D. M. German, and M.-A. Storey. Open source software peer review practices: A case study of the apache server. In Proceedings of the 30th International Conference on Software Engineering, ICSE '08, pages 541–550, 2008. ISBN 978-1-60558-079-1.
113. P. C. Rigby, D. M. German, L. Cowen, and M.-A. Storey. Peer review on open-source software projects: Parameters, statistical models, and theory. ACM Trans. Softw. Eng. Methodol., pages 35:1–35:33, 2014.
114. G. W. Russell. Experience with inspection in ultralarge-scale development. IEEE Softw., pages 25–31, 1991.
115. C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli. Modern code review: A case study at google. In Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP '18, pages 181–190. ACM, 2018.
116. C. Sauer, D. R. Jeffery, L. Land, and P. Yetton. The effectiveness of software development technical reviews: A behaviorally motivated program of research. IEEE Trans. Softw. Eng., pages 1–14, Jan. 2000.
117. 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 Proceedings of the 38th International Conference on Software Engineering Companion, ICSE '16, pages 212–221. ACM, 2016.
118. J. Shimagaki, Y. Kamei, S. McIntosh, D. Pursehouse, and N. Ubayashi. Why are commits being reverted?: A comparative study of industrial and open source projects. In 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 301–311, 2016.
119. F. Shull and C. Seaman. Inspecting the history of inspections: An example of evidence-based technology diffusion. IEEE Software, 25(1):88–90, 2008.
120. H. Siy and L. Votta. Does the modern code inspection have value? In Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01), ICSM '01, pages 281–289, 2001. ISBN 0-7695-1189-9.
121. D. M. Soares, M. L. d. L. Jnior, L. Murta, and A. Plastino. Rejection factors of pull requests filed by core team developers in software projects with high acceptance rates. In 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), pages 960–965, Dec 2015.
122. D. M. Soares, M. L. de Lima Jnior, A. Plastino, and L. Murta. What factors influence the reviewer assignment to pull requests? Information and Software Technology, 98:32 – 43, 2018.
123. D. Spadini, M. Aniche, M.-A. Storey, M. Bruntink, and A. Bacchelli. When testing meets code review: Why and how developers review tests. In Proceedings of the 40th International Conference on Software Engineering, ICSE '18, pages 677–687. ACM, 2018.
124. K. Spohrer, T. Kude, C. T. Schmidt, and A. Heinzl. Knowledge creation in information systems development teams: The role of pair programming and peer code review. In ECIS, 2013.
125. M. Stein, J. Riedl, S. J. Harner, and V. Mashayekhi. A case study of distributed, asynchronous software inspection. In Proceedings of the 19th International Conference on Software Engineering, ICSE '97, pages 107–117, 1997.
126. Y. Tao and S. Kim. Partitioning composite code changes to facilitate code review. In Proceedings of the 12th Working Conference on Mining Software Repositories, MSR '15, pages 180–190, 2015.

127. Y. Tao, Y. Dang, T. Xie, D. Zhang, and S. Kim. How do software engineers understand code changes?: An exploratory study in industry. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE '12, pages 51:1–51:11, 2012.
128. Y. Tao, D. Han, and S. Kim. Writing acceptable patches: An empirical study of open source project patches. In Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution, ICSME '14, pages 271–280, 2014.
129. T. Thelin, P. Runeson, and B. Regnell. Usage-based readingan experiment to guide reviewers with use cases. Information and Software Technology, pages 925 – 938, 2001.
130. P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida. Investigating code review practices in defective files: An empirical study of the qt system. In Proceedings of the 12th Working Conference on Mining Software Repositories, MSR '15, pages 168–179, 2015.
131. P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K. Matsumoto. Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pages 141–150, 2015.
132. 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. In Proceedings of the 38th International Conference on Software Engineering, ICSE '16, pages 1039–1050. ACM, 2016.
133. P. Thongtanunam, S. Mcintosh, A. E. Hassan, and H. Iida. Review participation in modern code review. Empirical Softw. Engg., pages 768–817, 2017. ISSN 1382-3256.
134. J. Tsay, L. Dabbish, and J. Herbsleb. Influence of social and technical factors for evaluating contribution in github. In Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, pages 356–366, 2014.
135. J. Tsay, L. Dabbish, and J. Herbsleb. Let's talk about it: Evaluating contributions through discussion in github. In Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014, pages 144–154, 2014.
136. H. Uwano, M. Nakamura, A. Monden, and K.-i. Matsumoto. Analyzing individual performance of source code review using reviewers' eye movement. In Proceedings of the 2006 Symposium on Eye Tracking Research &Amp; Applications, ETRA '06, pages 133–140, 2006. ISBN 1-59593-305-0.
137. M. van Genuchten, C. van Dijk, H. Scholten, and D. Vogel. Using group support systems for software inspections. IEEE Software, 18(3):60–65, 2001.
138. P. Vitharana and K. Ramamurthy. Computer-mediated group support, anonymity, and the software inspection process: an empirical investigation. IEEE Transactions on Software Engineering, 29(2):167–180, 2003.
139. R. J. Walker, S. Rawal, and J. Sillito. Do crosscutting concerns cause modularity problems? In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE '12, pages 49:1–49:11, 2012.
140. J. Wang and J. M. Carroll. Behind linus's law: A preliminary analysis of open source software peer review practices in mozilla and python. In 2011 International Conference on Collaboration Technologies and Systems (CTS), pages 117–124, 2011.
141. J. Wang, P. C. Shih, and J. M. Carroll. Revisiting linuss law: Benefits and challenges of open source software peer review. International Journal of Human-Computer Studies, pages 52 – 65, 2015.
142. J. Wang, P. C. Shih, Y. Wu, and J. M. Carroll. Comparative case studies of open source software peer review practices. Inf. Softw. Technol., 67:1–12, 2015.
143. G. M. Weinberg and D. P. Freedman. Reviews, walkthroughs, and inspections. IEEE Trans. Softw. Eng., 10(1):68–72, Jan. 1984.
144. P. Weißgerber, D. Neu, and S. Diehl. Small patches get in! In Proceedings of the 2008 International Working Conference on Mining Software Repositories, MSR '08, pages 67–76, 2008.
145. R. Wen, D. Gilbert, M. G. Roche, and S. McIntosh. BLIMP Tracer: Integrating Build Impact Analysis with Code Review. In Proc. of the International Conference on Software Maintenance and Evolution (ICSME), page 685694, 2018.
146. R. Wieringa, N. Maiden, N. Mead, and C. Rolland. Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. Requir. Eng., 11(1):102–107,

Dec. 2005. ISSN 0947-3602. doi: 10.1007/s00766-005-0021-6. URL http://dx.doi.org/10.1007/s00766-005-0021-6.

147. J. W. Wilkerson, J. F. Nunamaker, and R. Mercer. Comparing the defect reduction benefits of code inspection and test-driven development. IEEE Transactions on Software Engineering, 38: 547–560, 2012.
148. X. Xia, D. Lo, X. Wang, and X. Yang. Who should review this change?: Putting text and file location analyses together for more accurate recommendations. In 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 261–270, 09 2015.
149. X. YANG, N. YOSHIDA, R. G. KULA, and H. IIDA. Peer review social network (person) in open source projects. IEICE Transactions on Information and Systems, pages 661–670, 2016.
150. Y. Yu, H. Wang, G. Yin, and C. X. Ling. Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration. In 2014 21st Asia-Pacific Software Engineering Conference, volume 1, pages 335–342, 2014.
151. Y. Yu, H. Wang, G. Yin, and T. Wang. Reviewer recommendation for pull-requests in github. Inf. Softw. Technol., 74:204–218, 2016.
152. Y. Yu, G. Yin, T. Wang, C. Yang, and H. Wang. Determinants of pull-based development in the context of continuous integration. Science China Information Sciences, 59(8):080104, Jul 2016.
153. M. B. Zanjani, H. Kagdi, and C. Bird. Automatically recommending peer reviewers in modern code review. IEEE Trans. Softw. Eng., pages 530–543, 2016.
154. T. Zhang, M. Song, J. Pinedo, and M. Kim. Interactive code review for systematic changes. In Proceedings of the 37th International Conference on Software Engineering - Volume 1, ICSE '15, pages 111–122. IEEE Press, 2015.
155. X. Zhang, Y. Chen, Y. Gu, W. Zou, X. Xie, X. Jia, and J. Xuan. How do multiple pull requests change the same code: A study of competing pull requests in github. In 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 228–239, 2018.
156. Y. Zhang, G. Yin, Y. Yu, and H. Wang. A exploratory study of @-mention in github's pull-requests. In 2014 21st Asia-Pacific Software Engineering Conference, volume 1, pages 343–350, Dec 2014.
157. J. Zhu, M. Zhou, and A. Mockus. Effectiveness of code contribution: From patch-based to pull-request-based tools. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, pages 871–882, 2016.

Table 12 Listing of Collected Papers (Conference) with Summaries (citation count from Google Scholar)

Reference	Summary	#Citation	Snowball Reference
Sadowski et al. [115]	A novel case study in Google	7	[4, 19, 29, 41, 59, 71, 131]
German et al. [47]	Fairness of contribution in CR	1	[30, 60, 111, 149]
Kononenko et al. [74]	A novel case study of Shopify's active merchant in pull requests	3	[18, 69, 81, 90, 144]
Spadini et al. [123]	Test reviewing in CR	8	[36, 90, 99, 108, 111, 131]
Floyd et al. [45]	A controlled experiment to examine code comprehension, code review and prose review	23	[4, 39, 41]
Thongtanunam et al. [132]	New dataset to revisit the relationship between code ownership and software quality in CR	39	[88, 91]
Gousios et al. [50]	The work practices and challenges from integrator's	149	[18, 81, 90, 144]
Krusche et al. [76]	New workflows to teach students how to conduct code reviews	9	[41, 42, 94, 108, 143]
Kononenko et al. [73]	New aspect on how developers perceive the quality of CR	36	[18, 41, 57, 90]
Shimagaki et al. [117]	A novel case study of a proprietary system in Sony	17	[41, 66, 69, 88]
Rahman et al. [105]	Reviewer recommendation based on cross-project and technology experience	31	[131]
Zhang et al. [154]	An approach for inspecting systematic changes	38	[4, 37, 39, 41, 111]
Gousios et al. [51]	The work practices and challenges from contributor's perspective in pull-based development	86	[18, 81, 90, 111]
Barnett et al. [11]	An technique for decomposing changesets	64	[90, 108]
Gousios et al. [49]	An exploration on how pull-based development works	268	[90, 111, 144]
Tsay et al. [134]	Evaluation of contribution from social and technical factors in GitHub	189	[81, 90]
Gupta et al. [52]	An application of process mining repositories (ITS, PCR and VCS)	18	NA
Bacchelli and Bird [8]	An exploration on expectations, outcomes, and challenges in CR	343	[3, 4, 34, 41, 48, 111, 119, 125]
Balachandran [10]	A novel tool to reduce human efforts and improve quality in CR	103	NA
Rigby and Storey [110]	An empirical study to investigate the mechanisms and behaviours in peer review	143	[90, 93, 111, 116]
Rahman et al. [106]	A novel model to predict usefulness of review comments	2	[19, 91, 131]
Hellendoorn et al. [58]	A novel model to evaluate code contribution	34	[19, 30, 81, 90, 93, 102, 111, 111, 144]
Thongtanunam et al. [130]	Investigation of defective files in CR	45	[4, 7, 19, 41, 44, 66, 69, 88, 91, 92, 102, 104, 111, 114, 131, 131, 144]
Bosu et al. [32]	An exploratory qualitative analysis of useful reviews	59	[19, 29, 41, 66, 92, 102, 108, 111]
Bird et al. [26]	An experience report for CodeFlow Analytics	11	[88, 108]
Tao and Kim [126]	A novel approach to partition composite changes	31	[111]
Jiang et al. [65]	Estimation of acceptable patches	91	[111, 144]
McIntosh et al. [84]	The relationship between CR coverage and participation	173	[19, 41, 69, 90, 111]
Beller et al. [21]	Understanding of the practical benefits from the CR process	106	[19, 41, 57, 66, 69, 92, 136]
Baum et al. [16]	Investigation of the order of reading source code changes for review	12	[13, 15, 38, 40, 41, 86]
Ouni et al. [95]	A search-based approach to recommend reviewers	12	[4, 111, 131, 149], [30]
Shimagaki et al. [118]	Better understanding why commits are reverted	2	[91, 92]
Kononenko et al. [72]	An exploration on relationships between the reviewers code inspections and a set of factors	46	[18, 19, 41, 57, 69, 91, 92, 144]
Xia et al. [148]	An approach to recommend reviewers based on text and file location	32	[3, 19, 41, 91, 111, 131]
Bavota and Russo [17]	An exploration on the relationship between code review and bug introduction on the overall software quality	23	[19, 41, 69, 91, 111, 144]
Tao et al. [128]	A List of patch rejection reasons	33	[4, 18, 19, 41, 66, 92, 93, 111, 120, 144]

Table 13 Listing of Collected Papers (Conference) with Summaries (citation count from Google Scholar)

Reference	Summary	#Citation	Snowball Reference
Zhu et al. [157]	Comparison of code contribution between patch-based and pull-based tools	8	[30, 90, 93, 102, 111]
Baum et al. [12]	A novel case study in industries to see factors affecting CR	18	[13, 35, 41, 56, 61, 92, 102, 108, 124]
Allamanis et al. [5]	A novel framework to improve stylistic consistency	157	NA
Bosu et al. [31]	Identification of security vulnerabilities in CR	37	[29, 90]
Tsay et al. [135]	Evaluation of contribution through discussion in GitHub	95	[81, 90]
Rigby and Bird [109]	One of the first studies of contemporary review in software firms	11	[27, 41, 93, 102, 111, 116]
Tao et al. [127]	The first study on industrial practice in understanding code changes	109	NA
Walker et al. [139]	A novel case study in Mozilla to see crosscutting	12	[90]
Menarini et al. [87]	A semantic approach to assist CR	3	NA
Paixao et al. [96]	Understand the architectural impact of changes in CR	12	NA
Mirhosseini and Parnin [89]	Understanding whether these upgrading tools actually help developers	17	NA
Hannebauer et al. [55]	An expertise-based approach to recommend reviewers	8	[18, 62, 93, 131]

Table 14 Listing of Collected Papers (Journals) with Summaries (citation count from Google Scholar)

Reference	Summary	#Citation	Snowball Reference
Bernardo et al. [23]	Empirical evidence to support the claim that the adoption of CI is more quickly	15	[152]
Ram et al. [107]	Understanding of what makes a code change easier to review	15	[18, 71, 81, 112, 144, 156]
Wen et al. [145]	Propose a build impact analysis system named BLIMP Tracer	6	[30, 53, 91]
Zhang et al. [155]	An empirical study to explore competing pull requests	5	NA
Malavolta et al. [80]	An empirical study to understand maintainability issues of Android apps	2	NA
Rigby et al. [113]	Measure parameters for examination of peer review	51	[4, 27, 41, 42, 57, 66, 67, 71, 77, 79, 82, 90, 93, 99, 100, 102, 108, 111, 144]
Fan et al. [43]	A novel tool to early predict merged code changes	7	[19, 111, 119, 144]
Thongtanunam et al. [133]	Factors influence review participation in the CR process	33	[18, 29, 91, 93, 102, 108, 111, 116, 131, 144]
Baysal et al. [20]	Technical and non-technical factors influencing CR	31	[18, 19, 111, 144]
Mcintosh et al. [85]	Relationship between CR practices and long-term software quality	97	[19, 69, 90, 92, 102, 111]
McIntosh and Kamei [83]	A study to identify fix-inducing changes using Just-In-Time models	13	[91, 102]
Bosu et al. [33]	Better understanding of contemporary between OSS and industry	21	[19, 29, 30, 41, 56, 66, 108, 111]
Zanjani et al. [153]	Propose an approach named cHRev to recommend reviewers	39	[3, 4, 18, 19, 41, 71, 91, 102, 108, 111, 131, 144]
Kamei et al. [68]	A large-scale study to investigate Just-In-Time Quality Assurance	212	NA
Wilkerson et al. [147]	Comparative study for defect reduction benefits	30	[4, 7, 25, 41, 42, 46, 77, 78, 98, 137, 138]
Jiang et al. [63]	Analysis of inactive available assignees in GitHub	6	[62, 131, 150]
Jiang et al. [64]	Propose commenter recommendation for pull request with useful attributes	9	[19, 28, 62, 93, 111, 131, 150]
Yu et al. [151]	Reviewer recommendation for pull request using techniques from code review and bug assignment	69	[41, 90, 131, 150]
Wang et al. [142]	Compare differences of peer review practices across different OSS communities	7	[90, 111, 140, 141]
Soares et al. [122]	Understanding the factors that influence on assigning reviewers to pull requests and evaluating the extent of this influence	3	[62, 121]
Kovalenko et al. [75]	First study to evaluate a reviewer recommendation system	3	[131]