Towards code reviewer recommendation: a systematic review and mapping of the literature

Vinicius Schettino[1], Marco Antônio P. Araújo[1,2], José Maria N. David[1], Regina Braga[1], and Victor Ströele[1]

[1] Federal University of Juiz de Fora - Juiz de Fora, Brazil
[2] Federal Institute of Education, Science and Technology of Southeast Minas Gerais - Juiz de Fora, Brazil
vinicius.schettino@ice.ufjf.br

Abstract. (Context) Code review is a well-established practice for early detecting and reducing software defects. Researchers have shown that reviewer reputation and experience applied on the evolving component are important factors for review efficiency. In global software development, manual and human based methods for choosing a reviewer for a certain patch can be even more inefficient, due the developer's lack of knowledge about their teammate's expertise and physical contact with their peers. **(Objective)** This paper aims to present a review of Information Technology (IT) based tools and methods for code reviewer recommendation, with emphasis on application contexts, input data and empirical validations. **(Method)** To reach this goal, we conduct a systematic mapping and review compliant with existing evidence-based Software Engineering literature. **(Results)** Our results show that many authors seek real and open source datasets to validate their work, especially from GitHub. Also, previous works propose development and revision expertise and social relationships as inputs for recommendation methods, evaluating them with metrics such as Top-k and review activeness. **(Conclusion)** We classified these methods and conclude that new research should consider focusing on open/real datasets, and propose new validation approaches and method inputs especially for global software development.

Keywords: code review, code reviewer recommendation, global software development.

1 Introduction

Code reviewing is considered as an important technique for defect control and detection in software based systems [1]. The practice consists of a developer analyzing a colleague's code searching for defects, based on a pre-agreed checklist. Morales et al. [2] report lower incidence of anti-patterns through active participation and coverage of reviewed code. Kemerer and Paulk [3] show results where code reviewing is responsible for avoiding bad design practices.

The firt reports about code reviewing are from the 80s. Fagan et al. [4] describe a well-defined process, based on checklists, review meetings and auditing. These activities were designed for waterfall development approach, focusing on code quality and early bug detection.

As software development evolved, code review followed its path. The development of agile methods, concerns about software quality, and the increasing demand for higher levels maintainability are among the main factors that have led its evolution. Collaboration between authors and reviewers have become more important, and tools have emerged to support such interaction, e.g. Gerrit. This recent version of code review is called Modern Code Review (MCR) [5].

Pull-based development is a growing distributed development flavor [6], supported by tools such as GitHub and Gitlab. In this approach, the developer who wants to contribute create a pull request (PR). It contains the changeset and further explanations about its motivation and operation that will be analyzed and may be approved by a core contributor with sufficient permissions [7]. Code review plays a key role in this method, taking advantage of decentralized and branch based version control systems. The revision is conducted on the software platform, usually supported by features such as reviewer assignment, pull request search, tags and snippet-specific comments.

Previous works show that reviewer characteristics might impact code review efficiency. Baysal et al. [8] and Bosu [9] report that reviewer experience is important for code review. The number of reviewers and amount of interaction are analyzed by Kononenko et al. [10]. Thongtanunam et al. [11] show when a unsuitable reviwer is assigned, the review can last 12 days longer. Xia et al. [12] show that reviewer experience on the changed component is also significant on code review efficiency. In particular in a pull-based development context, Jiang et al. [13] show that activeness is the most important attribute sought in a suitable reviewer.

In distributed development context, manual reviewer assignment can be even harder than in collocated teams [14]. In global development there are more reviewers, submitters and different aspects such as timezone, availability, culture, experience and knowledge. It gets harder for a human to take all these factors into consideration. Since previous studies show that the choice of a reviewer who turns out to be unfit affects process efficiency, tools that support the code reviewer search and assignment are important.

In the Free/Open Source Software domain (FOSS), there is a special attention to code review. Major projects incorporate this practice as a mandatory step in order to accept contributions [15, 16].

Given the difficulty and the importance of code reviewer recommendation methods in association with the broad range of methods already proposed in several contexts, we present a systematic review [17] of the literature towards code reviewer recommendation. After reviewing existing evidence-based Software Engineering literature, we aim to present the state-of-the-art of code reviewer recommendation tools, methods and relevant features observed on the choice of potential reviewers. We seek to present how this research has grown, and who

the top contributors are. We also aim to show the contexts in which this kind of technique is often used, and how these works evaluate and compare their results. These analysis may help to settle grounds on code review future recommendation approaches, evaluate the efficiency of existent tools and aid future reseachers on this field.

Previous works have attempted to gather and evaluate different methods of code reviewer recommendation, though none of them have proposed a systematic approach to find them. Furthermore, no clear classification model of the methods, nor the context where they best fit in have been presented.

Yang et al. [18] present several studies and apply a popular method for a large pull request dataset, drawing conclusions about the importance of activeness of reviewers and inactive reviewers in the process. They also provide an in-depth analysis about this pull request dataset, helping evaluate the relevance of review behaviors on recommendations. Hannebauer et al. [19] recognize the difficulty of assigning suitable reviewers and compare empirically eight algorithms with different approaches in four major FLOSS projects, finding that those based on review expertise yield better recommendations. Consentino et al. [20] conduct a broader systematic mapping of the pull-based development in GitHub, addressing code review and other development processes.

Although we praise the efforts of previous works, we find that a systematic approach is more suitable in order to bring evidence in this research field. The formalism and scientific rigor of the systematic review and mapping make the process and the results reproducible and adaptable, avoiding researcher bias and fostering evidence-based conclusions [21].

Besides this Introduction, this paper is structured as follows: Section 2 explains the sytematic review and mapping protocol. The reports for review and mapping are presented on Section 3 and 4. A clousure of this research and continuation possibilities are provided on Section 5 and 6.

2 Systematic review and mapping

The main goals of a systematic review and mapping can be condensed as "give a complete, comprehensive and valid picture of the existing evidence" [21]. Thus, it is necessary to carry out both identification, analysis and interpretation in a scientific and rigorous way [21].

A systematic approach to scientific computer science research has been proposed by Kitchenham [17], adapting systematic review guidelines from other fields, medicine in particular. A mapping study may be applied for an overview of the state-of-the-art, whereas a review focuses on a specific research question related to the outcomes of the studies [21]. Both may follow the same protocol, but differ when they might deal with inclusion/exclusion criteria and the scope. In this work, both approaches can be helpful, given the sought goals. We need both a general view of the research field and a narrow outcome/efficiency and context analysis to present the desirable objectives.

This review was conducted based on the steps suggested by Wholin et al. [21]: planning, conducting and reporting the related studies.

All evaluation steps were carried out by four different experts, in independent analysis. The research questions, inclusion/exclusion criteria and any other significant divergence were widely discussed before taking the next step.

2.1 Review and mapping planning

The planning phase includes the identification of the need for a review, the development of a review protocol and the specification of research questions [17]. The need is grounded as no recent systematic review of this field was found. As we defined our objective as "find evidence of IT based tools and methods for code reviewer recommendation, with emphasis on application contexts, input data and empirical validations", we propose a compatible goal to the systematic review and mapping. According to the Goal/Question/Metric (GQM) approach [22], we aim to "**analyze** the metrics, measures, criteria and aspects used in the code reviewer recommendation, **in order to** classify them regarding code review efficiency **from the point of view** of software team members **in relation to** their colleagues in the context of software maintenance and evolution."

Mapping questions: In order to characterize the research field, we gathered a set of questions towards code reviewer recommendation. These are the research questions of the systematic mapping:

MQ1: Who are the most active authors in the area? We identify the main authors in the field, offering a comprehensive start point to future researchers.

MQ2: Which publication vehicles are the main targets for research production in the area? The publication type and weight can provide evidence of field maturity and attention from the academic community.

MQ3: In which context code review recommendation solutions are used? (Global Software Development, Industry, Open Source) We want to understand the context that nourishes code review recommendation research, as we hypothesize that this technique is important in Global Software Development.

Review Questions: For a more specific and in-depth review, we propose the following questions to be answered in the systematic review:

RQ1: How is the code review efficiency measured? In order to propose new recommendation methods and tools, it is important to have ways to compare new results with previous works.

RQ2: What information from software repositories are used to recommend code reviewers? In order to propose new recommendation methods and tools, it is important to understand which variables were used so that newcomers can discuss and extend their applications.

PICOC: Through the PICOC approach, proposed by Petticrew and Roberts [23], the research questions in this study were described by the five elements of a researchable question:

- Population (P): software developers
- Intervention (I): code reviewer recommendation
- Comparison (C): -
- Outcomes (O): methods and tools that support code reviewer recommendation
- Context(C): code review

Inclusion/Exclusion Criteria: We submitted the results to an inclusion/exclusion criteria checklist to get a fine adjustment of the search and to evaluate the relevance of the papers found. To be considered, a paper must meet all the criteria shown below:

- IC1: The paper proposed a code reviewer recommendation method AND
- IC2: The algorithm inputs are clear AND
- IC3: The efficiency measurement approach is clear AND
- IC4: There is empirical evidence of the method efficiency AND
- IC5: The paper are written in English language AND
- IC6: The paper are reported at a peer reviewed Workshop or Conference or in a Journal.

Sources: After we defined the research questions, we chose the sources of papers to execute our search string. This process was conducted based on the following: Capacity for executing using logical expressions; capacity for full-length or specific paper topic searches; available in the researcher's institution; papers on Computer Science. Four sources met these characteristics: Scopus, IEEExplore, ScienceDirect, Compendex

Query String: Based on the research goals defined by GQM approach [22] and the chosen terms on PICOC [23], the string was generated by gathering the terms with Boolean OR/AND operators. Similar terms and synonymous where also included in order to obtain a broader spectrum of papers. Terms such as "pull-request" were added to be sure we would find papers about new developments in Modern Code Review in the context of pull-based software development. The following query string was chosen:

> *("software developer" AND developer) AND ("reviewer recommendation" OR "commenter recommendation") AND (method OR tool OR solution OR framework) AND ("code review" OR "pull-request" OR "pull request")*

The proposed query string was reviewed by all the authors separately, as well by an external researcher in order to validate its composition and synonyms. All terms were found in related works to ensure they are well known and usual on this research field.

2.2 Review and mapping conduction

In this phase we executed the query string in each database, with slight modifications in order to comply with each engine syntax. The results were imported to Parsifal (http://parsif.al), a useful open source tool to support the systematic framework proposed by [17]. This tool was used to remove duplicated papers and classify the remaining as relevant or not to this research, based on the inclusion/exclusion criteria presented above.

Overall, 106 papers were found. Of those, 29 were duplicated and 60 were rejected based on the exclusion criteria above, during the abstract reading. The main reason of rejection was not presenting a recommendation system with efficiency empirical evidence (IC1 and IC4). The remaining 17 papers were classified for complete reading, fulfilling all the inclusion criteria.

3 Systematic mapping report

The 17 selected papers were analyzed in order to answer the mapping questions (MQ). MQ1 (Who are the most active authors in the area?) is included in order to reveal the main contributors, leading newcomers to have good starting points in this research field. Table 1 summarizes the distribution of works by author. Raula Gaikovina Kula appeared in three of the selected papers, while Jing Jiang appeared in 2 of them as well as in related works [13].

Name	Count
Raula Gaikovina Kula	3
Norihiro Yoshida, Hajimu Lida, Xin Yang, Jing Jiang, Jia-Huan He, Ken-ichi Matsumoto	2
Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Mohammad Masudur Rahman, Chanchal Roy, Jason Collins, Yue Yu, Huaimin Wang, Gang Yin, Charles Ling, Vipin Balachandran, Catarina Costa, Jair Figueredo, Leonardo Murta, Anita Sarma, Motahareh Zanjani, Huzefa Kagdi, Christian Bird, Haochao Ying, Liang Chen, Jian Wu, Tingting Liang, Xin Xia, Xinyu Wang, David Lo, Xiaohu Wang, Xue-Yuan Chen	1

Table 1. Author names and number of publications

Another way to visualize how mature a research field is is to look at the main channels of research publication, such as workshops, conferences and journals. To answer MQ2 (Which publication vehicles are the main targets for research production in the area?) we found that 23% of the selected articles came from journals, and 59% of them were published at conferences. The rest (18%) were presented at workshops. Although in Computer Science the number of conferences is significantly higher compared with journals, this ratio may indicate that this is not yet a mature field, with a relevant proportion of in-progress research.

As well as the channel category, its weight is more evidence of field maturity. In table 2 we show the distribution of papers in the respective channels and their h5-indexes, found on Google Scholar[3]. This table is sorted by highest h5-index.

[3] https://scholar.google.com/citations?view_op=top_venues&hl=en

Reference	Channel	h5-index
[24][25]	ICSE - IEEE/ACM International Conference on Software Engineering	68
[26]	IEEE Transactions on Software Engineering	52
[27]	GLOBECOM - IEEE Global Communications Conference	48
[28]	SIGSOFT - ACM International Symposium on Foundations of Software Engineering	43
[29]	Journal of Intelligent Information Systems	22
[30]	Journal of Computer Science and Technology	22
[31]	IEICE Transactions on Information and Systems	17
[32]	SANER - IEEE International Conference on Software Analysis, Evolution and Reengineering	13
[33, 34]	APSEC Asia-Pacific Software Engineering Conference	12
[35]	IEEE/ACM International Workshop on Software Mining	*
[36]	International Workshop on CrowdSourcing in Software Engineering	*
[37][38][39]	ICSME - IEEE International Conference on Software Maintenance and Evolution	*
[40]	IWCSN - International Workshop on Complex Systems and Networks	*

Table 2. Papers per channel and their respective H-indexes

In order to seek answers to MQ3, we classified each paper using one or many categories, based on the context in which the study was conducted. We only took into consideration direct mentions to some terms, for example "Open Source" or "Distributed Development". Even if the study used open software repositories or distributed development tools (such as GitHub), we looked for direct mentions of this information being relevant to their conclusions.

Code review recommendation may be harder in some contexts, such as in global and open source software development. Thus, it might be that some of them need more sophisticated recommendation techniques than others. MQ3 (In which context code review recommendation solutions are used?) was formulated to provide evidence for this hypothesis. Table 3 shows the distribution we found, sorted by highest publication year. The contexts used in the categorization were found in related works and accepted papers.

Reference	Pull Based	Open Source	Distributed Development	Modern Code Review	General Development
[29]					[X]
[40]		[X]	[X]		
[27]	[X]	[X]	[X]		
[35]				[X]	
[28]			[X]		
[26]			[X]	[X]	
[36]	[X]		[X]		
[24]	[X]		[X]		
[31]		[X]			
[39]				[X]	
[30]	[X]	[X]			
[32]				[X]	
[38]	[X]		[X]		
[37]	[X]		[X]		
[33]	[X]		[X]		
[25]					[X]
[34]		[X]			

Table 3. Context related keywords

Code reviewer recommendation seems to be tightly related with distributed development and the pull-based approach, although there are some occurrences of general development and Modern Code Review. This special attention seems to be tied to the inherent challenges of global software development. Ying et al. [36] cite the greater number of reviews (through pull requests) of this method as motivation to automatic code reviewer recommendation. Yu et al. [33] say that finding the right reviewer to a PR in an open source project is a typical

crowdsourcing job. Thus, it needs to solicit opinions of the community. Also in this context, reliable and rich information that helps to identify an appropriate code reviewer is often hard to find and aggregate, especially manually [24].

Almost every paper used known Open Source projects in order to evaluate their proposals, but some of them used specific data of this kind of development in order to motivate and execute code reviewer recommendation. In this context, a rapid online collaboration encourages more and more developers to join in OSS projects, while on the other hand, integrating new developers with teams is challenging and pivotal to the success of a project [40]. While the amount of contribution is high, experienced developers who are responsible for ensuring that only high quality changesets will be merged can get swamped with a large number of patches to review [34]. This scenario can delay the PR merge and discourage new contributions [30].

Open Source codebases are important to ensure a given experiment is auditable and reproducible. Thus, we suggest that new method proposals should be evaluated also in real situations, especially with already used open codebases, in order to make their contribution value clear.

4 Systematic review report

The two Review Questions (RQs) were planned in order to give a narrow perspective of how the selected works handle the recommendation and assert their efficiency. With RQ1 (How the code review efficiency is measured?) we seek to understand how past authors discovered how good their methods are, giving newcomers a entry point on how new methods may be evaluated. In Table 4, papers are categorized by their efficiency measurement methods and sorted by latest publication year.

Reference	Top-K	Precision/Recall	MRR	Activeness	Fostering	Performance
[29]		[X]				[X]
[40]	[X]	[X]	[X]			
[27]		[X]				
[35]	[X]	[X]				
[28]	[X]					
[26]	[X]					
[36]	[X]	[X]				
[24]	[X]	[X]	[X]			
[31]				[X]		
[39]	[X]	[X]				
[30]	[X]					
[32]	[X]		[X]			
[38]	[X]		[X]			
[37]	[X]	[X]				
[33]	[X]	[X]				
[25]	[X]					
[34]	[X]	[X]				

Table 4. Efficiency Methods

The most used efficiency assertion method is the Top-K ranking, with Precision and Recall analysis. Top-K is the similarity percentage between a K-sized set the algorithm recommended in comparison with a same sized standard set. In the presented cases the comparison default group consist of actual humans: by comparing the recommendation with the posteriori choice or with a manual analysis and classification of "good" and "bad" reviews. So these methods are evaluated by how close to a human recommendation they are. Precision is the percentage of relevant items (thus, among human recommendation set) out of all recommended items, and recall is the measure of how many relevant items were left out of the method recommendation.

Beside Top-K ranking, Thongtanunam et al. [32] also proposed an analysis based on the Mean Reciprocal Rank (MRR). This approach takes in consideration not only the set but also the position of the recommendations. So if the first recommendation is inside the user recommendation set, the method gets a higher MRR.

Yang et al. [31] applies a social network approach, with different evaluation methods. They measure how the recommended reviewers could affect the activeness of the code review process.

Comparisons with previous works are important to show the contribution of new approaches, and this is simpler when the same efficiency metrics are used. Hence, new researchers might want to address the main metrics presented here to evaluate new methods. However, some metrics might be more suitable in certain contexts. As shown in MQ3 response, aspects of Global Software Development make code reviewer recommendation a non-trivial task for a human, which might not have all the information needed [24]. Challenges such as culture and timezone conflicts, language and lack of personal contact also make this task harder.

Since traditional metrics such as Top-k are based on how close a set of recommendations is to a human recommendation, they might not be enough to evaluate a recommendation method in this context. Yang et al. [31] use an approach which takes into consideration how the recommended reviewer impacted the process observing the activeness of a review. Previous works also discussed the positive traits of a code review, focused on comment usefulness, changeset configuration and reviewer profile [9, 41].

Addressing RQ2 (What information from software repositories are used to recommend code reviewers?), we define categories of code reviewer recommendation methods, based on the main inputs they use and extending previous classifications [19]. The next subsections describe these categories, using some of the papers to exemplify. Table 5 summarizes the results of this classification. Four categories were identified: review expertise, development expertise, social relationships and hybrid approaches.

4.1 Review Expertise

Methods in this category recommend a reviewer based on their experience in similar changesets. The similarity can be measured by file/code or by PR text proximity, or even a blend of both. While Fejzer et al. [29] propose a reviewer

Category	Articles
Review Expertise	[29, 27, 26, 39, 32, 38]
Development Expertise	[28, 24, 25, 34]
Social Relationships	[40, 35, 31, 33, 37]
Hybrid Approaches	[36, 30]

Table 5. Method classification summary

profile and similarity function between such profiles and change proposals to be reviewed, Liao et al. [27] define a set of topics to describe the expertise of a reviewer and the expertise needed for a given review. Thongtanunam et al. [32] leverage the similarity of a previously reviewed file path to recommend an appropriate code reviewer.

4.2 Development Expertise

Some authors used expertise in development to infer that a subject might be a good reviewer. To recommend an expert to merge two branches, Costa et al. [28] analyse their past changes in the project, especially dependencies of the current changeset. Rahman et al. [24] presented CORRECT, a method that leverages cross project development expertise in order to find suitable reviewers.

4.3 Social Relationships

Another approach is to use the social relationships of past revisions and collaborations in order to find a suitable reviewer. The links are usually designed as graphs that are built with help of textual analysis looking for similarity of the given pull request and project history. Fu et al. [40] propose recommendations based on a graph mapping work status and the social relationships of developers. Xia et al. [35] use latent factor models and neighborhood methods to capture implicit relations seeking higher recommendation efficiency. Yu et al. [33, 37] extend traditional Machine Learning methods of bug triaging to reviewer recommendation, mining comment networks of pull request discussions on global software development. Yang et al. [31] employ social network analysis (SNA) techniques based on reviewer contribution and activeness. Much of the research that focuses on this kind of approach is new, leading us to believe that it is an interesting matter to be analyzed and discussed in depth.

4.4 Hybrid Approaches

These methods bring together different inputs in order to take the best of both worlds. Through a graph, Ying et al. [36] simultaneously considers developer expertise and authority to recommend. With CoreDevRec, Jiang et al. [30] propose to taking into account file paths of modified codes, relationships between contributors and core members, and activeness of core members.

In order to summarize and make the contributions of this work clear, Table 6 briefly lists responses to our Mapping and Review Questions.

Question	Answer
MQ1	Raula Gaikovina Kula.
MQ2	Mainly Conferences (58%).
MQ3	Specially in Global Software Development.
RQ1	Mainly Top-k Rank with precision and recall.
RQ2	Review and Development Expertise, Social Relationships and Hybrid Approaches

Table 6. Review and Mapping Responses Summary

5 Conclusions

Code review is a well-established practice, with effects which have been studied and supported by many authors over the years. From the traditional process that included review meetings and a rigid workflow born in the 70s, the technique has evolved to a lightweight flexible workflow that values knowledge transfer and interpersonal relationships.

Recent studies have shown the impact of reviewer adequacy on code review efficiency, leading to meaningfully discussions and early defect detection. These results motivate researchers to find new methods to recommend suitable reviewers given the context of a revision, such as changeset composition, author and technology involved and textual composition of the review request.

This paper provides evidence on IT based tools and methods for code reviewer recommendation, with emphasis on application contexts, input data and empirical validations. Through a systematic mapping and review complying with existing evidence-based Software Engineering literature, we presented the state of the art of this research field, supporting new research to fill the gaps left by previous works. Although there are some initiatives to provide empirical evidence in this research field, this is the first systematic rigorous attempt to reach this goal.

This is a growing search field, with several ongoing works. Many contexts, inputs and results have been shown, and there is still no silver bullet method to find the best reviewer for a code review process.

Many works argue that in global software development, especially in Open Source and Pull-based contexts, automatic reviewer recommendation becomes more valuable: different cultures, schedules, availability, activeness and knowledge can influence the choice, and it is hard for a human to gather all this information and select a suitable reviewer.

The analyzed papers employ different inputs, such as reviewing expertise, development expertise and social relationships on one or many projects. Historical data are compared with information of the given review, and the methods recommend a set of reviewers that might be suitable in each case.

In order to evaluate their method's efficiency, most authors evaluate how close their recommendations to a real human choice, through Top-k with precision and recall analysis. There are approaches that measure how efficient the review of a reviewer they recommended was, in order to evaluate the quality of the method. Using standard top-k analysis it is important to compare with previous works,

and we encourage new research to also evaluate results with review efficiency metrics that have already been discussed in mature literature.

6 Future work

In future research, we aim to further investigate the assertions and hypothesis based on the evidence identified in this systematic mapping and review. In depth studies can be made in order to understand the impacts of good reviewers on Open Source ecosystems. Also, besides using the main metrics found to address RQ1 to evaluate new methods, we intend to explore new ways of evaluating methods for global software development, measuring how good a review done by a recommended reviewer was. This approach is still mostly under explored and may be relevant for distributed contexts. Recent studies have shown that social relationships are growing in relevance in code reviewer recommendation, so this is an aspect that we would like to investigate as well.

References

1. Boehm, B., Basili, V.R.: Software defect reduction top 10 list. Computer **34**(1) (12 2001) 135–137
2. Morales, R., McIntosh, S., Khomh, F.: Do code review practices impact design quality? a case study of the qt, vtk, and itk projects. 2015 IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER) **00** (2015) 171–180
3. Kemerer, C.F., Paulk, M.C.: The impact of design and code reviews on software quality: An empirical study based on psp data. IEEE Transactions on Software Engineering **35**(4) (07 2009) 534–550
4. Fagan, M.E.: Design and code inspections to reduce errors in program development. IBM Syst. J. **15**(3) (09 1976) 182–211
5. Bacchelli, A., Bird, C.: Expectations, outcomes, and challenges of modern code review. In: Proceedings of the 2013 International Conference on Software Engineering. ICSE '13, IEEE Press (2013) 712–721
6. Gousios, G., Storey, M.A., Bacchelli, A.: Work practices and challenges in pull-based development: The contributor's perspective. In: Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on, IEEE (2016) 285–296
7. Gousios, G., Pinzger, M., Deursen, A.v.: An exploratory study of the pull-based software development model. In: Proceedings of the 36th International Conference on Software Engineering, ACM (2014) 345–355
8. Baysal, O., Kononenko, O., Holmes, R., Godfrey, M.: The influence of non-technical factors on code review. (2013) 122–131
9. Bosu, A., Greiler, M., Bird, C.: Characteristics of useful code reviews: An empirical study at microsoft. In: Proceedings of the 12th Working Conference on Mining Software Repositories. MSR '15, Piscataway, NJ, USA, IEEE Press (2015) 146–156
10. Kononenko, O., Baysal, O., Guerrouj, L., Cao, Y., Godfrey, M.: Investigating code review quality: Do people and participation matter? (2015) 111–120
11. Thongtanunam, P., Kula, R., Cruz, A., Yoshida, N., Iida, H.: Improving code review effectiveness through reviewer recommendations. (2014) 119–122

12. Xia, X., Lo, D., Wang, X.b., Yang, X.: Who should review this change?: Putting text and file location analyses together for more accurate recommendations. In: IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016). (2015) 261–270
13. Jiang, J., Yang, Y., He, J., Blanc, X., Zhang, L.: Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development. Information and Software Technology **84** (2017) 48–62
14. Yu, Y., Wang, H., Yin, G., Wang, T.: Reviewer recommendation for pull-requests in github. Inf. Softw. Technol. **74**(C) (June 2016) 204–218
15. Asundi, J., Jayant, R.: Patch review processes in open source software development communities: A comparative case study. In: System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on. (2007) 166c–166c
16. Baysal, O., Kononenko, O., Holmes, R., Godfrey, M.W.: The secret life of patches: A firefox case study. In: 2012 19th Working Conference on Reverse Engineering. (2012) 447–455
17. Kitchenham, B.: Procedures for performing systematic reviews. Keele, UK, Keele University **33**(2004) (2004) 1–26
18. Yang, C., Zhang, X., Zeng, L., Fan, Q., Yin, G., Wang, H.: An empirical study of reviewer recommendation in pull-based development model. In: Proceedings of the 9th Asia-Pacific Symposium on Internetware. Internetware'17, New York, NY, USA, ACM (2017) 14:1–14:6
19. Hannebauer, C., Patalas, M., Stünkel, S., Gruhn, V.: 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, New York, NY, USA, ACM (2016) 99–110
20. Cosentino, V., Izquierdo, J.L.C., Cabot, J.: A systematic mapping study of software development with github. IEEE Access **5** (2017) 7173–7192
21. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering. Springer Science & Business Media (2012)
22. Basili, V.R., Weiss, D.M.: A methodology for collecting valid software engineering data. IEEE Trans. Softw. Eng **SE-10, no. 6** (1984) 728–738
23. Petticrew, M., Roberts, H.: Systematic reviews in the social sciences: A practical guide. John Wiley and Sons (2008)
24. Rahman, M.M., Roy, C.K., Collins, J.A.: Correct: code reviewer recommendation in github based on cross-project and technology experience. In: Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on, IEEE (2016) 222–231
25. Balachandran, V.: Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In: Software Engineering (ICSE), 2013 35th International Conference on, IEEE (2013) 931–940
26. Zanjani, M.B., Kagdi, H., Bird, C.: Automatically recommending peer reviewers in modern code review. IEEE Transactions on Software Engineering **42**(6) (2016) 530–543
27. Liao, Z., Li, Y., He, D., Wu, J., Zhang, Y., Fan, X.: Topic-based integrator matching for pull request. GLOBECOM 2017 - 2017 IEEE Global Communications Conference (2017) 1–6
28. Costa, C., Figueiredo, J., Murta, L., Sarma, A.: Tipmerge: recommending experts for integrating changes across branches. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM (2016) 523–534

29. Fejzer, M., Przymus, P., Stencel, K.: Profile based recommendation of code reviewers. Journal of Intelligent Information Systems (Aug 2017)
30. Jiang, J., He, J.H., Chen, X.Y.: Coredevrec: Automatic core member recommendation for contribution evaluation. Journal of Computer Science and Technology **30**(5) (2015) 998–1016
31. Yang, X., Yoshida, N., Kula, R.G., Iida, H.: Peer review social network (person) in open source projects. IEICE TRANSACTIONS on Information and Systems **99**(3) (2016) 661–670
32. Thongtanunam, P., Tantithamthavorn, C., Kula, R., Yoshida, N., Iida, H., Matsumoto, K.I.: Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. (2015) 141–150
33. Yu, Y., Wang, H., Yin, G., Ling, C.X.: Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration. In: Software Engineering Conference (APSEC), 2014 21st Asia-Pacific. Volume 1., IEEE (2014) 335–342
34. Lee, J.B., Ihara, A., Monden, A., i. Matsumoto, K.: Patch reviewer recommendation in oss projects. In: 2013 20th Asia-Pacific Software Engineering Conference (APSEC). Volume 2. (Dec 2013) 1–6
35. Xia, Z., Sun, H., Jiang, J., Wang, X., Liu, X.: A hybrid approach to code reviewer recommendation with collaborative filtering. In: 2017 6th International Workshop on Software Mining (SoftwareMining). (Nov 2017) 24–31
36. Ying, H., Chen, L., Liang, T., Wu, J.: Earec: leveraging expertise and authority for pull-request reviewer recommendation in github. In: Proceedings of the 3rd International Workshop on CrowdSourcing in Software Engineering, ACM (2016) 29–35
37. Yu, Y., Wang, H., Yin, G., Ling, C.X.: Reviewer recommender of pull-requests in github. In: 2014 IEEE International Conference on Software Maintenance and Evolution. (Sept 2014) 609–612
38. Xia, X., Lo, D., Wang, X., Yang, X.: Who should review this change?: Putting text and file location analyses together for more accurate recommendations. In: Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on, IEEE (2015) 261–270
39. Ouni, A., Kula, R.G., Inoue, K.: Search-based peer reviewers recommendation in modern code review. In: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME). (Oct 2016) 367–377
40. Fu, C., Zhou, M., Xuan, Q., Hu, H.X.: Expert recommendation in oss projects based on knowledge embedding. (12 2017) 149–155
41. Rahman, M.M., Roy, C.K., Kula, R.G.: 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, Piscataway, NJ, USA, IEEE Press (2017) 215–226