From Transient Information to Persistent Documentation: Enhancing Software Documentation

Felipe Ebert
Department of Mathematics and Computer Science
Eindhoven University of Technology (TU/e)
Eindhoven, The Netherlands
f.ebert@tue.nl

***Abstract*—Developers usually need to use different kinds of tools in the software development process, *e.g.*, version control systems, code review tools, and bug or issue tracking systems. Some of those tools provide an enhanced approach for developers to communicate, *i.e.*, discussions are recorded in the communication feature of those tools. There is important information, *i.e.*, discussions and decisions, registered by those tools which are not propagated to the project documentation: the *transient* information. Thus, this work investigates how to improve the software documentation by making such *transient* important information into *persistent* documentation as source code comments. The results are relevant because they will aid the comprehension of the source code not only during the development process, but also in code reviews.**

***Index Terms*—confusion, transient information, software documentation, development process**

I. Introduction

The development process requires developers to use different kinds of tools. For example, version control systems are used to manage changes in the source code, code review tools are used to verify the quality of the code changes by other team members, bug tracking systems are used to report and track bugs in the source code, and issue tracking systems are used to report and manage issues in any artifact. Usually, these tools also provide developers a means for communication, *i.e.*, the discussions among developers related to that specific artifact are recorded within these tools. For instance, developers can discuss a certain method in the code review tool, or a specific issue design in the issue tracking system.

While all these tools improve the communication and the development process, they also add a problem for the project documentation. There are important discussions and decisions being recorded by those tools (such as code review, or bug and issue tracking systems) [1], [2] and they are *transient* for the project, *i.e.*, they are kept within the tool and do not propagate further in the project artifacts. Such a scenario can cause confusion in developers because they might not be able to understand the source code they need to change or review. For example, a code review in which the class design was discussed and a decision was taken. Some time later, a developer working on that same class might miss the rationale, thus becoming confused. Thus, we believe that such *transient* information which is relevant for the project should become *persistent* documentation in the form of source code comments, as they are permanent artifacts of the software projects.

This paper describes a post-doctoral research proposal based on my PhD thesis [3]. The background related to code review is presented in Section II. Next, a synopsis of my PhD dissertation is discussed by providing a summary of the general research problem and pointing out its main contributions (Section III). The actual research proposal is presented in Section IV. Finally, the conclusions are presented in Section V.

II. Code Review

Code review is a technique of systematic examination of code change, which can be conducted before or after the change is integrated into the main code repository [4]. Code changes submitted by a developer are reviewed by one or more of their peers. It is an iterative process and can be instantiated in different ways. As input, a code review receives the original code change and the outcome is the reviewed change, which might be either accepted or rejected. The developer who wrote the code change is the author, and might also be responsible for submitting the change for review. The reviewers are responsible for assuring that the code change is functionally correct, meets the performance requirements, and follows the quality standards of the project. Code review tools usually provide a set of common features: they show the difference between the files (*i.e.*, before and after the change) and developers can either add general comments pertaining to the entire change or pinpoint concerns or shortcomings about a specific part of the change using inline comments.

Code reviews are an important practice for software quality assurance [5]–[9]. Several open source projects, *e.g.*, Android, Qt, and Eclipse, and companies, *e.g.*, Microsoft, Oracle, and Samsung, have already adopted code review as part of their development process. Likewise, several studies have also shown that code review can provide multiple benefits in the development process [4], [10]–[13]. The main goals of code reviews are to find bugs in the code change, and verify whether the project guidelines and coding style are being respected [4], [14]–[17]. Furthermore, code reviews help to improve the quality of the code in production, find better ways to implement the change, spread the knowledge about the source code, and create awareness of the changes in the project [4], [10]–[13].

III. Understanding Confusion in Code Reviews

In this section, we briefly present a synopsis of my thesis.

A. Research Problem

Despite the benefits code reviews bring, they can incur costs on software development projects, as they can delay the merge of a code change in the repository and, consequently, slowdown the overall development process [1], [18]. The time invested by a developer in reviewing code is non-negligible [5] and may take up to 10%–15% of the overall time invested in software development activities [12], [17]. Furthermore, performing a code review might not be such an easy task. In fact, understanding the code change and its context is one of the major issues reviewers face during code reviews [4], [12], [19]–[21]. The merge of a code change in the repository can be further delayed when reviewers experience difficulties in understanding the change, *i.e.*, when they are not certain of its correctness, run-time behavior and impact on the system [4], [12], [19]–[21].

We believe that *confusion*, *i.e.*, a reviewer not being able to understand something during the code review, can affect the artifacts that developers produce and the way they work, and hence, negatively impact the development process [4], [12], [19]–[21]. For instance, the code review might take longer than it should, the quality of the review might decrease, more discussions might take place, or even the code change might be blindly accepted or summarily rejected. As such, we believe that a proper understanding of the phenomenon of *confusion* in code reviews is a necessary starting point towards reducing the cost of code reviews and enhancing the effectiveness of this practice, thereby improving the overall development process.

In my PhD thesis, we tackle two important problems related to confusion in code reviews. The first one is the *lack of knowledge* as to how confusion influences and affects the code review process. So far, the phenomenon of confusion in code reviews was not well-understood by the community: what confusion is, what the reasons and impacts of confusion are, and how developers cope with it. Understanding confusion is crucial for researchers studying the impact of affective aspects of software development on the development process and on software itself, as well as for development teams aiming to reduce the negative consequences of code reviews, such as the delay of the development process.

The second problem, the *lack of tools* for confusion identification in code reviews, stems from the lack of knowledge about confusion. Such tools are important for identifying developers experiencing confusion, and in designing interventions to support them. The goal of my thesis is to mitigate these two problems: the lack of knowledge about confusion in code reviews, and the lack of tools for confusion identification.

B. Methodology

We conducted three different in-depth studies to address the two problems described above. Firstly, we needed to understand *what confusion is*. Thus, in our first study [22], by researching studies from the field of Psychology, we proposed a definition of *confusion*. Then, we explored machine learning techniques in order to build an automated approach for confusion identification, comprising two classifiers that can automatically identify confusion in code review comments.

Subsequently, we focused on investigating the reasons for confusion, its impacts, and the way developers cope with it [2]. We triangulated [23] data from a survey of 54 developers with the manual analysis of 307 code review comments manually labeled by the researchers as confusing: 156 are general and 151 are inline comments. As result, we produced a comprehensive model for *confusion in context* in code reviews including 30 reasons, 14 impacts, and 13 coping strategies.

Finally, we deepened our focus further to investigate the communicative intention of developers' questions, *i.e.*, the talkative goal of the questions, as they are one of the causes of confusion in code reviews [24]. Our findings suggest that questions in code review serve diverse communicative goals, *e.g.*, requesting clarifications, discussing hypothetical scenarios, and suggesting improvements.

C. Main Contributions

My PhD thesis offers several contributions. Most of them provide new results and insights related to confusion in code reviews. All the gold standard sets and classification models are publicly available[1] for research purposes [25].

- A confusion coding scheme, a gold standard set with code review comments from ANDROID, labeled as *confusion* and *no confusion* (comprising of 1,542 general and 1,190 inline comments), together with an automated approach for detecting confusion in code review comments;
- A model of *confusion in context*, with the 30 reasons and 14 impacts of confusion in code reviews, as well as 13 strategies that developers adopt to cope with confusion;
- A series of practical and actionable suggestions to improve code review tools;
- A classification of the communicative intentions expressed in the developers' questions in code reviews and its gold standard set.

D. Discussion

We believe the three studies of my PhD theses are innovative as, to the best of our knowledge, there was no research on those topics before it. To accomplish this work, we needed to carry out a multi-disciplinary research, involving works from different areas such as Software Engineering, Education, and Psychology, in order to characterize what confusion means, the ways it manifests itself, and how it can be identified. For instance, one of the contributions of our work is a consolidation of several different views in a taxonomy about confusion. Although this taxonomy targets mainly the software development area, it can be employed in other contexts.

Furthermore, the model of *confusion in context* reveals causes and mitigation strategies that, as far as it was possible to ascertain, are not studied in the literature. For example,

[1] https://github.com/felipeebert/confusion-in-code-reviews

as many situations of confusion are resolved through offline discussions that are not incorporated into the project's memory. In addition, it quantifies the prevalence of the causes of confusion identified. The proposed classification may serve as a basis for several new researches on code reviews.

Finally, the understanding of the communicative intention of developers' questions is important in scenarios where communication occurs mainly via text and several nuances of language are lost. It can also provide additional support to increase the inclusion of neurodiverse developers, who may have difficulty understanding this communicative intention, *e.g.*, individuals with autism or attention deficit hyperactivity disorder (ADHD).

E. Lessons Learned

We obtained a low number of responses in the survey we employed, even though we had sent emails with personalized salutation to developers. Another issue we see now is that a large number of the developers we sent the survey to were not contributing to the project for a long period. Thus, we learned that to get better response rates we should follow a more *personal* approach when contacting respondents. For instance, we plan to avoid such problem by using the firehouse method when contacting developers for interviews. In this way, we will reach developers just after they have submitted the code changes, and all information related to it will probably be fresh for them. Thus, we believe they will be more likely to accept our interview invitation.

IV. Research Proposal

The research proposal we envision is still related to *confusion* during the development process. However, it has a bit broader scope.

A. Context

Usually, developers use different channels for both communication and coding, *e.g.*, version control systems (such as Git), code review tools (such as Gerrit and GitHub), bug tracking systems (such as Bugzilla), and issue tracking systems (such as Jira and GitHub). On the one hand, all those kinds of tools have provided great support for developers as they, for instance, facilitate developers contributing to projects with branching and pull requests, and improve their communication by centralizing the discussion of related concerns, *e.g.*, code reviews and issues. On the other hand, we believe there is still a problem with the communication and documentation in the development process: the discussions, and possible relevant decisions made, among developers are usually transient, *i.e.*, they are recorded but only within each tool. For example, if a code review has an important discussion on why the code change has a certain design, it is likely that rationale will remain only in the code review comment. The same can happen with issue discussions. Thus, such relevant and *transient* information will not be properly documented, *i.e.*, they will remain available only within each tool. We envision that such *transient* information should become *persistent* within the project, *e.g.*, by transforming them into source code comments.

B. Research Goals

As shown by our model of *confusion in context*, several factors related to documentation causes confusion: the most prevalent one is *missing rationale*, but other factors are *missing documentation*, and *lack of context*. Thus, we believe that one step to reduce confusion and improve the development process overall is by making important and transient information persistent. Such approach can benefit developers in several ways, *e.g.*, a developer of a change will have "at hand" the rationale of a design decision of a certain class so they will not need to research or make questions about it, a reviewer will be able to easily note that there was an specific issue with an important discussion in the source code comment when reviewing a code change, and mostly important, each relevant information and decision about each part of the code will be document as a source code comment near to that piece of code.

Hence, the first goal of this proposal is to empirically explore which pieces of *transient* information from different tools, such as code review or issue tracker tools, are relevant for developers and managers. We acknowledge this might be a challenge, *e.g.*, to decide how much, how often, and when the information should be considered for documentation, but we believe this an important step towards achieving our final goal. Next, we need to be able to automatically identify such important information. This comprises the second research goal. Finally, the third goal is to use such information and automatically generate source code comments to transform the *transient* into *persistent* documentation.

As a final product of this research, we envision a bot that could be integrated into the development process to reduce confusion due to several factors and improve the documentation. For example, GitHub already provides several different kinds of bots which automatize several different processes. One of them is the TODO Bot[2] which creates an issue in the project when the developer pushes a code change with the tag "@todo". As such, we believe that a similar bot could be implemented to make important *transient* into *persistent* documentation. Thus, this bot could suggest source code comments for developers in the closure of their tasks, *e.g.*, when they close a issue or finish a code review they would be prompted with the suggestion. In this way, the developer could approve, change, or reject the code comments suggested by the bot, making this process semi-automated to avoid undesirable information being documented. Hence, the developer will be able to decide on how much of the information it should be documented. Additionally, by making the suggestion of the source code comments in the end of developers' tasks, we hope to improve the usefulness and accuracy of the comments, as the discussion will probably be finished.

[2]https://github.com/probot/probot

For instance, consider a real scenario: the code review 31831[3] of ANDROID. We envision a bot suggesting the following source code comment *"The dynamic section in the current MIPS compiler is writable, so MIPS does not require this exception now."* in the file *linker.c* as a response to the reviewers question.

The initial **research questions** of this proposal are:

- **RQ1:** Which *transient* information is relevant enough to become *persistent* documentation?
- **RQ2:** Which is the best approach to automatically identify such relevant information from different tools' discussions (such as code review and issue tracker systems)?

C. Research Design

This proposal will require a mixed-method approach [23]. We plan to start with the dataset from our previous study [2] which provides a classified set of code review comments and their reasons for confusion. We will focus on the comments with the reason for confusion related to documentation, *e.g.*, *missing rationale*, *lack of documentation*, and *lack of context*. Thus, we will survey developers which participated in those code reviews and ask them which comments from the review discussion they consider relevant enough to become persistent documentation. Furthermore, we will also ask them where in the source code they would like to have the comments. This will create a initial gold standard set of relevant information to be documented. In the same time, we plan to increase our dataset of possible relevant information by manually labeling new comments.

Furthermore, another approach we envisage is to use *firehouse research* [26], [27] to conduct interviews with developers. This research method has such name due to the fact that the research requires events to occur that cannot be induced by the researchers themselves, *i.e.*, we would literally sit in a "firehouse" waiting for a "fire to be reported" — a code review or issue submission in our case. This means that we would be "listening" for those events, and we would "run to the scene" after a new review or issue submitted, and we consider they might contain important information. Through this method, we will be able to interview developers within at most a few days from the day they submitted a code change to be reviewed or issue to be solved. Hence, we could get fresh feedback on their reasoning about the relevant parts of the discussions, and also where in the source code they would add that information.

The goal of the previous steps is to build a dataset labeled with relevant information for the project. With that, we plan to explore machine learning techniques in order to automatically identify *transient* and important discussions which should become *persistent* documentation. Furthermore, we also plan to experiment deep learning techniques for such a goal, as it is capable of learning from data that is unstructured or unlabeled. By comparing those two approaches we aim at answering the **RQ2**.

[3]https://android-review.googlesource.com/c/31831

Finally, the final step of this research will be the evaluation of our tool with developers. We plan to build a prototype which can accomplish this research goal and make an evaluation with the end users. Such evaluation will require a controlled experiment with developers and will assess the usefulness of the suggestion of source code comments.

D. Related Work

Robillard *et al.* [28] envisioned a paradigm for developers making queries and obtaining documentation on-demand. Ponzanelli *et al.* [29] presented a tool which automatically queries STACK OVERFLOW discussions and recommends them to the developer in the IDE. Our proposal is complementary for those studies as we plan to suggest documentation (source code comments) to be persisted for developers based on information from different sources.

Traceability link recovery studies [30]–[32], which proposed means to connect high-level (*e.g.*, requirements) and low-level artifacts (*e.g.*, source code), will be considered to aid linking discussion to code elements. Literature on discussion summarization [33]–[35] will aid the suggestion of concise comments. Research on identification of rationale [36], [37] and detection of knowledge types in software artifacts [38], [39] will provide insights on the kind of information to be used on the source code comments.

E. Significance of the Research

We believe this research can bring new and important insights about how and which *transient* information should be documented as source code comments. Such research will improve the project documentation as relevant pieces of information, such as critical decisions from code review comments, will remain persistent and easily accessible along the whole development process. Additionally, developers can benefit from having better documentation when either reviewing or implementing new code changes, making the code review process faster.

V. CONCLUSION

In this paper, we presented a short synopsis of my PhD thesis by focusing on the research problem and the main contributions. Based on the results of my PhD thesis, we propose a new research with the ultimate goal of making *transient*, but important, information *persistent* as source code comments. We hope that it can contribute with several problems related to the lack of documentation, rationale, and context, for instance. Two research questions were proposed for the starting point, but it of course can be re-evaluated during the course of the research. In the end, we envision a bot that can automatically identify and extract *transient* information and transform them into proper documentation.

ACKNOWLEDGMENTS

I would like to thank my advisors, Prof. Fernando Castor and Prof. Alexander Serebrenik, for all the support during my doctoral period.

REFERENCES

[1] L. Pascarella, D. Spadini, F. Palomba, M. Bruntink, and A. Bacchelli, "Information needs in contemporary code review," vol. 2, no. CSCW. New York, NY, USA: ACM, Nov. 2018, pp. 135:1–135:27.

[2] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Confusion in code reviews: Reasons, impacts, and coping strategies," in *SANER*, 2019, pp. 49–60.

[3] F. Ebert, "Understanding confusion in code reviews," Ph.D. dissertation, Federal University of Pernambuco, Recife, Brazil, 2019. [Online]. Available: https://felipeebert.github.io/post/phd-2019/phd-2019.pdf

[4] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *ICSE*. Piscataway, NJ, USA: IEEE Press, 2013, pp. 712–721.

[5] Y. Tao and S. Kim, "Partitioning composite code changes to facilitate code review," in *MSR*. Piscataway, NJ, USA: IEEE Press, 2015, pp. 180–190.

[6] 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)*. Bremen, Germany: IEEE Computer Society, Sept 2015, pp. 81–90.

[7] B. Boehm and V. R. Basili, "Top 10 list [software development]," *Computer*, vol. 34, no. 1, pp. 135–137, Jan 2001.

[8] M. V. MÄntylÄ and C. Lassenius, "What types of defects are really discovered in code reviews?" *TSE*, vol. 35, no. 3, pp. 430–448, 2009.

[9] 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*, ser. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 134–144. [Online]. Available: http://dl.acm.org/citation.cfm?id=2818754.2818773

[10] T. Pangsakulyanont, P. Thongtanunam, D. Port, and H. Iida, "Assessing MCR discussion usefulness using semantic similarity," in *2014 6th International Workshop on Empirical Software Engineering in Practice*. Osaka, Japan: IEEE Computer Society, Nov 2014, pp. 49–54.

[11] 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)*. Montreal, QC, Canada: IEEE Computer Society, March 2015, pp. 171–180.

[12] J. Cohen, S. Teleki, and E. Brown, *Best Kept Secrets of Peer Code Review*. Massachusetts, EUA: Smart Bear Inc., 2006.

[13] 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 Software Engineering*, vol. 21, no. 5, pp. 2146–2189, Oct 2016. [Online]. Available: https://doi.org/10.1007/s10664-015-9381-9

[14] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Syst. J.*, vol. 15, no. 3, pp. 182–211, Sep. 1976. [Online]. Available: http://dx.doi.org/10.1147/sj.153.0182

[15] K. E. Wiegers, *Peer Reviews in Software: A Practical Guide*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[16] J. Wang, P. C. Shih, Y. Wu, and J. M. Carroll, "Comparative case studies of open source software peer review practices," *Inf. Softw. Technol.*, vol. 67, no. C, pp. 1–12, Nov. 2015. [Online]. Available: https://doi.org/10.1016/j.infsof.2015.06.002

[17] 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*, vol. 43, no. 1, pp. 56–75, Jan 2017.

[18] M. Greiler, "On to code review: Lessons learned @ microsoft," 2016, keynote for QUATIC 2016 - the 10th International Conference on the Quality of Information and Communication Technology. [Online]. Available: https://pt.slideshare.net/mgreiler/on-to-code-review-lessons-learned-at-microsoft

[19] 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*, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 51:1–51:11. [Online]. Available: http://doi.acm.org/10.1145/2393596.2393656

[20] A. Sutherland and G. Venolia, "Can peer code reviews be exploited for later information needs?" in *2009 31st International Conference on Software Engineering - Companion Volume*. Vancouver, BC, Canada: IEEE, May 2009, pp. 259–262.

[21] T. D. Latoza, G. Venolia, and R. DeLine, "Maintaining mental models: A study of developer work habits," in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 492–501. [Online]. Available: http://doi.acm.org/10.1145/1134285.1134355

[22] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Confusion detection in code reviews," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Shanghai, China: IEEE Computer Society, Sept 2017, pp. 549–553.

[23] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, *Selecting Empirical Methods for Software Engineering Research*. London: Springer London, 2008, pp. 285–311. [Online]. Available: https://doi.org/10.1007/978-1-84800-044-5_11

[24] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Communicative intention in code review questions," in *The 34th IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Madrid, Spain: IEEE Computer Society, Sept 2018, pp. 519–523.

[25] F. Ebert, "Material of the PhD thesis "Understanding Confusion in Code Reviews"," Feb 2019, permanent link: https://github.com/felipeebert/confusion-in-code-reviews. [Online]. Available: https://doi.org/10.5281/zenodo.2541203

[26] E. M. Rogers, *Diffusion of innovations*, 5th ed. New York, NY [u.a.]: Free Press, 08 2003.

[27] E. Murphy-Hill, T. Zimmermann, C. Bird, and N. Nagappan, "The design of bug fixes," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 332–341.

[28] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, M. Linares-Vásquez, G. C. Murphy, L. Moreno, D. Shepherd, and E. Wong, "On-demand developer documentation," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017, pp. 479–483.

[29] L. Ponzanelli, G. Bavota, M. D. Penta, R. Oliveto, and M. Lanza, "Prompter: A self-confident recommender system," in *2014 IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 577–580.

[30] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017, pp. 3–14.

[31] K. Moran, D. N. Palacio, C. Bernal-Cárdenas, D. McCrystal, D. Poshyvanyk, C. Shenefiel, and J. Johnson, "Improving the effectiveness of traceability link recovery using hierarchical bayesian networks," 2020.

[32] J. M. Florez, "Automated fine-grained requirements-to-code traceability link recovery," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2019, pp. 222–225.

[33] S. Rastkar, G. C. Murphy, and G. Murray, "Automatic summarization of bug reports," *IEEE Transactions on Software Engineering*, vol. 40, no. 4, pp. 366–380, 2014.

[34] N. Jha and A. Mahmoud, "Using frame semantics for classifying and summarizing application store reviews," *Empirical Software Engineering*, vol. 23, no. 6, pp. 3734–3767, 2018. [Online]. Available: https://doi.org/10.1007/s10664-018-9605-x

[35] L. Ponzanelli, A. Mocci, and M. Lanza, "Summarizing complex development artifacts by mining heterogeneous data," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15. IEEE Press, 2015, p. 401–405.

[36] G. Viviani, M. Famelis, X. Xia, C. Janik-Jones, and G. C. Murphy, "Locating latent design information in developer discussions: A study on pull requests," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

[37] R. Alkadhi, M. Nonnenmacher, E. Guzman, and B. Bruegge, "How do developers discuss rationale?" in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 357–369.

[38] W. Maalej and M. P. Robillard, "Patterns of knowledge in api reference documentation," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1264–1282, 2013.

[39] D. Arya, W. Wang, J. L. C. Guo, and J. Cheng, "Analysis and detection of information types of open source software issue discussions," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 454–464.