Continuous Code Reviews

A Social Coding tool for Code Reviews inside the IDE

Tobias Dürschmid
Hasso Plattner Institute, University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
tobias.duerschmid@student.hpi.de

ABSTRACT

Code reviews play an important and successful role in modern software development. But usually they happen only once before new code is merged into the main branch. We present a concept which helps developers to continuously give feedback on their source code directly in the integrated development environment (IDE) by using the metaphor of social networks. This reduces context switches for developers, improves the software development process and allows to give feedback to developers of external libraries and frameworks.

CCS CONCEPTS

•**Software and its engineering** → *Integrated and visual development environments;* Collaboration in software development; •**Human-centered computing** → **Collaborative and social computing systems and tools;**

KEYWORDS

code review, code quality, social coding, feedback

ACM Reference format:
Tobias Dürschmid. 2017. Continuous Code Reviews. In *Proceedings of Programming '17, Brussels, Belgium, April 03-06, 2017,* 3 pages.
DOI: http://dx.doi.org/10.1145/3079368.3079374

1 PROBLEM AND MOTIVATION

Performing code reviews is an important practice for professional companies and open source projects [3, 8, 10, 14, 24, 26]. Reasons are finding defects, improving code quality, discussing alternative solutions, transferring knowledge, and improving team awareness [2, 24]. Recent studies confirmed, that review coverage and review participation have a significant impact on code quality and the correctness of software [20, 21, 27, 29].

But traditionally, code reviews are done only once before the code is merged into the main branch [24]. This process does not continuously give feedback on code quality (especially of legacy code), does not support questions of new developers concerning existing code, and forces the developers to leave their IDE for commenting on code.

Social networks like Facebook[1], Twitter[2], or Stack Overflow[3] enable interaction with any content by adding a comment or pushing an "I like" button. We transfer this metaphor to programming by offering developers to comment on any code and to like clean code snippets in the project using their IDE.

Code reviews of large changes (20 files or more) are less useful [10]. Therefore, the proposed tool tries to continuously give feedback just while reading a piece of code inside the IDE. This approach lowers the barrier to comment on source code and increases the amount of feedback. Furthermore, it can be enriched by applying gamification techniques to commenting on source code and finding bugs [19].

2 BACKGROUND

2.1 Code Reviews

A code review is a systematic examination of source code. Reviews range from very informal (e.g., pair programming [7]) to very formal (e.g., software inspections [1, 13]). Convergent peer reviews are predominantly lightweight, flexible processes, that happen early, quickly and frequently [24–27]. Therefore, we refer to code reviews as informal peer reviews.

Currently the main activity of reviews changes from defect finding to discussions about alternative solutions and long-term code maintainability [10, 24]. Only about 15% of the peer review comments point out possible defect while 50% address code quality [10]. Non-technical factors like reviewer load and activity, author experience, and the company structure significantly impact the review outcome [5, 6].

One challenge developers face during code reviews is context switching, because they have to understand another issue and stop doing their current work [10, 18]. Switching from one problem space to another reduces developers' productivity and should be avoided [22].

2.2 Social Coding

Social coding is the community-based development of software. Social coding sites like GitHub[4] and BitBucket[5] enable substantially more collaborations among developers [30] and shift the focus of interactions to individual contributors and their activities with electronic artifacts [11, 28]. Therefore, social coding is a new and promising approach for improving the software development process by encouraging the collaborations in the team.

[1] https://facebook.com [2] https://twitter.com [3] https://stackoverflow.com
[4] https://github.com [5] https://bitbucket.org

Programming '17, Brussels, Belgium

978-1-4503-4836-2/17/04…$15.00
DOI: http://dx.doi.org/10.1145/3079368.3079374

3 WALKTHROUGH

This section describes a hypothetical scenario, how the proposed tool can be used.

The developer Alice reads a piece of code relevant to her current issue. She does not understand, why this is done like it is and adds a comment on this piece of code. Because Bob did the last change to this code, he gets informed about the new comment by a notification in the IDE and via email. He opens the concerned method and answers the questions. Alice recommends Bob to refactor the code to directly reveal this intention. He does so and pushes the done button of the comment in order to hide the comment of the discussion. After Alice has finished her issue, Bob enjoys reading her code and therefore pushes the "I like" button of the new method in order to help new developer getting to know the coding styles of the project.

4 MAIN DESIGN DECISIONS

This section describes the design decisions we made during implementation of the tool in Squeak, a Smalltalk environment, using a Ruby server for exchanging the comments.

4.1 Client-Server Architecture

In order to exchange the comments across the team, a central data storage is needed. One simple solution would be to attach the comments in custom fields of a commit of the version control software. But this enforces users to perform a commit after commenting on source code, which is impossible when the developer has no write access to the code that was commented. Therefore, we implemented this tool using a client-server architecture. The server manages the comments in a database and the client creates, modifies and shows the comments in the IDE.

4.2 Hiding Comments using an explicit Done Button

Comments that have been discussed and resolved should not be displayed to the developers any more. Our first intuition was, that a comment should be hidden when the method was changed. But this leads to problems when the change does not correspond to the comment (e.g., the comment criticizes the name of a method and a developer adds one call inside the method). Many other scenarios could lead to hiding a comment that is still applicable. Therefore, we decided to keep all comments until they are explicitly marked as done by one developer.

5 RELATED WORK

Many projects working with feature branches perform code reviews during pull requests before merging the changes into the main branch [9, 12, 16, 23, 31, 32, 35]. Tools like CodeFlow [8], Mondrian [17], Gerrit [15], Phabricator [33], ClusterChanges [4] and the Eclipse plug-in EGerrit[6] support these change-based code reviews. In contrast, our concept gives feedback on the current state of code. The reviews supported by these tools are made using a push model, meaning that developers request reviews, while our tool uses a pull model, meaning that developers can comment on any code without request of the author. Hence, our concept allows to comment on library code and framework code and therefore to give feedback to the developers of third party software. Furthermore, our feedback is continuous and therefore enables new developers to comment on old code. Similar to plug-ins like EGerrit, the proposed tool enables the developers to stay inside their IDE and avoid context switches. This provides a more self-sustaining environment which supports the liveness of the development process. However, a dedicated review process before merging changes into the main branch lets the developers focus on issues raised up by the specific changes. Therefore, we propose to use our concept in conjunction with pull requests or other change-based reviews and not instead of it.

Discussions and questions about the source code traditionally are done using mailing lists [34]. The recent emergence of questions & answers (Q&A) sites introduces gamification of archiving reputation for answering questions [34]. But like pull requests, they usually are not done inside the IDE but on a separate StackExchange network like StackOverflow.

[6] https://www.eclipse.org/egerrit/

6 DISCUSSION

Our approach encourages continuous conversations about source code quality, correctness, alternative solutions, and other topics. It simplifies discussions between distant developers in large organizations or open source projects by providing a social coding platform which forwards a comment automatically to the author of the concerned code. Furthermore, we argue, that using this approach, many source code comments will move out of the code base into the social coding comment database. This keeps the code clean and focused.

Unfortunately, we did not find any solution for dealing with developers working on the project but not having installed our tool. This could lead to inconsistent states, for examples, when this developer moves a method that is attached to a comment. Therefore, this tool currently does only work correctly if all working developers have installed our tool.

7 CONCLUSION

We proposed a social coding tool which helps developers continuously giving feedback on code quality and potential mistakes. This is one contribution for social coding towards a constructive programming community with collective code ownership. Furthermore, it is a contribution for the vision of a self-sustaining IDE by providing a tool for code reviews as one important process for code improvements.

ACKNOWLEDGMENTS

I would like to thank Patrick Rein for his support and the HPI software architecture group for the continuous feedback on the project.

REFERENCES

[1] A Frank Ackerman, Lynne S Buchwald, and Frank H Lewski. 1989. Software inspections: an effective verification process. *IEEE software* 6, 3 (1989), 31.
[2] Alberto Bacchelli and Christian Bird. 2013. Expectations, Outcomes, and Challenges of Modern Code Review. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 712–721.
[3] Vipin Balachandran. 2013. Reducing Human Effort and Improving Quality in Peer Code Reviews Using Automatic Static Analysis and Reviewer Recommendation.

In *Proceedings of the '13 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 931–940.

[4] Mike Barnett, Christian Bird, João Brunet, and Shuvendu K. Lahiri. 2015. Helping Developers Help Themselves: Automatic Decomposition of Code Review Changesets. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1 (ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 134–144.

[5] Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W Godfrey. 2013. The influence of non-technical factors on code review. In *Working Conference on Reverse Engineering (WCRE 2013)*. IEEE, 122–131.

[6] Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W Godfrey. 2016. Investigating technical and non-technical factors influencing modern code review. *Empirical Software Engineering* 21, 3 (2016), 932–959.

[7] Kent Beck. 2000. *Extreme programming explained: embrace change*. addison-wesley professional.

[8] Christian Bird, Trevor Carnahan, and Michaela Greiler. 2015. Lessons Learned from Building and Deploying a Code Review Analytics Platform. In *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR '15)*. IEEE Press, Piscataway, NJ, USA, 191–201.

[9] Fabio Calefato, Roberto De Nicolò, Filippo Lanubile, and Fabrizio Lippolis. 2015. Product Line Engineering for NGO Projects. In *Proceedings of the Fifth International Workshop on Product LinE Approaches in Software Engineering (PLEASE '15)*. IEEE Press, Piscataway, NJ, USA, 3–6.

[10] Jacek Czerwonka, Michaela Greiler, and Jack Tilford. 2015. Code Reviews Do Not Find Bugs: How the Current Code Review Best Practice Slows Us Down. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2 (ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 27–28.

[11] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW '12)*. ACM, New York, NY, USA, 1277–1286.

[12] Vincent Driessen. 2010. A successful Git branching model. (2010). http://nvie.com/posts/a-successful-git-branching-model/

[13] Michael E Fagan. 2001. Design and code inspections to reduce errors in program development. In *Pioneers and Their Contributions to Software Engineering*. Springer, 301–334.

[14] Dror G Feitelson, Eitan Frachtenberg, and Kent L Beck. 2013. Development and deployment at facebook. *IEEE Internet Computing* 17, 4 (2013), 8–17.

[15] Google Inc. 2016. Gerrit Code Review. (2016). https://www.gerritcodereview.com/

[16] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An Exploratory Study of the Pull-based Software Development Model. In *Proceedings of the 36th International Conference on Software Engineering (ICSE '14)*. ACM, New York, NY, USA, 345–355.

[17] Niall Kennedy. 2006. How Google does web-based code reviews with Mondrian. (30 November 2006). http://www.niallkennedy.com/blog/2006/11/google-mondrian.html

[18] Oleksii Kononenko, Olga Baysal, and Michael W. Godfrey. 2016. Code Review Quality: How Developers See It. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. ACM, New York, NY, USA, 1028–1038.

[19] Rafael Lotufo, Leonardo Passos, and Krzysztof Czarnecki. 2012. Towards improving bug tracking systems with game mechanisms. In *IEEE Working Conference on Mining Software Repositories (MSR 2012)*. IEEE Press, 2–11.

[20] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2014. 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 '14)*. ACM, New York, NY, USA, 192–201.

[21] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21, 5 (2016), 2146–2189.

[22] Mary Poppendieck and Tom Poppendieck. 2003. *Lean Software Development: An Agile Toolkit: An Agile Toolkit*. Addison-Wesley.

[23] Mohammad Masudur Rahman and Chanchal K. Roy. 2014. An Insight into the Pull Requests of GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR '14)*. ACM, New York, NY, USA, 364–367.

[24] Peter C. Rigby and Christian Bird. 2013. Convergent Contemporary Software Peer Review Practices. In *Proceedings of the '13 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE '13)*. ACM, New York, NY, USA, 202–212.

[25] Peter C. Rigby, Brendan Cleary, Margaret-Anne Storey, and Daniel M German. 2012. Contemporary Peer Review in Action. *IEEE software* 29, 6 (2012), 56–61.

[26] Peter C. Rigby, Daniel M. German, Laura Cowen, and Margaret-Anne Storey. 2014. Peer Review on Open-Source Software Projects: Parameters, Statistical Models, and Theory. *ACM Trans. Softw. Eng. Methodol.* 23, 4, Article 35 (Sept. 2014), 33 pages.

[27] Junji Shimagaki, Yasutaka Kamei, Shane McIntosh, Ahmed E. Hassan, and Naoyasu Ubayashi. 2016. 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)*. ACM, New York, NY, USA, 212–221.

[28] Margaret-Anne Storey. 2012. MSR 2012 keynote: The evolution of the social programmer. In *IEEE Working Conference on Mining Software Repositories (MSR 2012)*. IEEE, 140–140.

[29] Patanamon Thongtanunam, Shane McIntosh, Ahmed E. Hassan, and Hajimu Iida. 2015. 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)*. IEEE Press, Piscataway, NJ, USA, 168–179.

[30] F. Thung, T. F. Bissyand, D. Lo, and L. Jiang. 2013. Network Structure of Social Coding in GitHub. In *17th European Conference on Software Maintenance and Reengineering (CSMR '13)*. 323–326.

[31] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of Social and Technical Factors for Evaluating Contribution in GitHub. In *Proceedings of the 36th International Conference on Software Engineering (ICSE '14)*. ACM, New York, NY, USA, 356–366.

[32] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. 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 '14)*. ACM, New York, NY, USA, 144–154.

[33] Alexia Tsotsis. 2011. Meet Phabricator, The Witty Code Review Tool Built Inside Facebook. (7 August 2011). https://techcrunch.com/2011/08/07/oh-what-noble-scribe-hath-penned-these-words/

[34] Bogdan Vasilescu, Alexander Serebrenik, Prem Devanbu, and Vladimir Filkov. 2014. How Social Q&A Sites Are Changing Knowledge Sharing in Open Source Software Communities. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW '14)*. ACM, New York, NY, USA, 342–354.

[35] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan Vasilescu. 2015. Wait for It: Determinants of Pull Request Evaluation Latency on GitHub. In *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR '15)*. IEEE Press, Piscataway, NJ, USA, 367–371.