Learning to Boost the Efficiency of Modern Code Review

Robert Heumüller
Otto von Guericke University Magdeburg, Germany
robert.heumueller@ovgu.de

***Abstract*—Modern Code Review (MCR) is a standard in all kinds of organizations that develop software. MCR pays for itself through perceived and proven benefits in quality assurance and knowledge transfer. However, the time invest in MCR is generally substantial. The goal of this thesis is to boost the efficiency of MCR by developing AI techniques that can partially replace or assist human reviewers. The envisioned techniques distinguish from existing MCR-related AI models in that we interpret these challenges as graph-learning problems. This should allow us to use state-of-science algorithms from that domain to learn coding and reviewing standards directly from existing projects. The required training data will be mined from online repositories and the experiments will be designed to use standard, quantitative evaluation metrics. This research proposal defines the motivation, research-questions, and solution components for the thesis, and gives an overview of the relevant related work.**

***Index Terms*—modern code review, deep learning, automated software engineering**

I. Motivation

Today, many organizations - ranging from open-source projects to global players like Microsoft or Google - have adopted some variation of a lightweight yet systematic peer code review process [1], [2]. To distinguish them from previous more rigid practices, contemporary processes that are tailored to the needs of specific development teams and projects are known as Modern Code Review (MCR) [3]. For example, the MCR process at Google involves five steps: *Creating, Previewing Changes, Commenting, Addressing Feedback, and Approving Changes* [1].

To get an idea of the average time invested for MCR, we interviewed project leads and developer teams of three medium-sized projects ($>$ 25 kSLOC) at a mid-sized software-development company in Germany[1]. In addition to these qualitative assessments, we also consulted statistics of the project management software, to determine how much time issues spend in the code-review state. The results indicate that, between projects, 5.2h-10h per week (13%-25%) of developers' time is spent on MCR at that company. These results are similar to what was reported for open-source projects (6.4h/wk) but higher than the findings of a case study at Google (3.2h/wk) [1], [4]. While these numbers' generality is limited, we believe they do illustrate the potential gains of optimizing MCR processes. **Thus, the goal of this thesis is to design AI models capable of assisting or partially replacing human reviewers, to reduce the time per review without compromising review quality. Trained on source code and review data, these models should learn both coding and reviewing standards from real-world data.**

[1]The author has been working part-time as a product owner and as a systems architect at that company for several years.

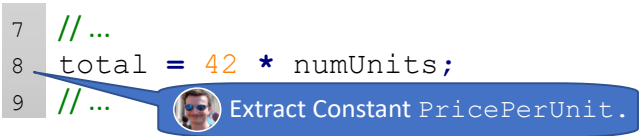

Fig. 1. Example MCR comment requesting a refactoring

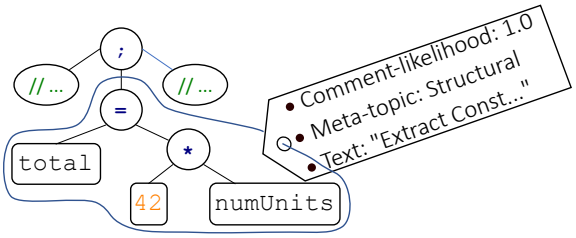

Fig. 2. Labeled AST for the example from Figure 1

II. Research Questions

Using only source code, review comments and meta-data:

RQ1: How can we design a model that can partially replace a human reviewer by predicting some aspects of their comments with respect to a given code change?

Here, the goal is to predict different aspects of what a human reviewer would remark on in an MCR situation. This includes predicting the comment locations, i.e. likelihood scores for the line numbers of a change under review, predicting the comment *meta-topics* (e.g. style-violations, structural issues, bugs, use-case issues) for these locations, and ultimately generating meaningful comment texts.

The first major challenge is deciding what can be learned only from source code and review data, to accordingly define the concrete learning tasks. For example, some comments address issues between an implementation and its software specification. Since a formal specification is typically not available for most source code, predicting such comments will be impossible in the majority of cases. To overcome such problems, we are already working on models classifying the *meta-topic* of given comments. This should enable us to define achievable learning tasks, for example detecting style-violations, structural issues, or some types of bugs.

The second challenge is providing the necessary ground-truth, including positive and negative labels for features like the comment-likelihood. How do you identify code that is not likely to be commented on? Our starting point is to use code

that was recently changed due to a review but then remained stable for a minimum period.

The third challenge is designing and evaluating AI models for the defined learning tasks. Here, the different learning tasks induce particular types of models such as regression, classification, or text generation. However, we interpret all learning tasks as graph-learning problems for program representation graphs, e.g. ASTs (Abstract Syntax Trees). All models will thus learn to compute features, to classify, or to generate text for graphs or subsets of their nodes and edges. Figure 1 illustrates the graph-learning idea for ASTs using the source code and review comment from Figure 2.

RQ2: How can we design and train a model that judges the quality of review comments concerning a given change?

Here, the goal is to learn to quantify the quality of code review comments, e.g. for providing feedback to reviewers. The definition of *quality* encompasses many aspects, some of which were explored in related work (cf. Section IV). As a starting point, we define two features, *actionability*, i.e. whether the comment induces changes to the respective code, and *clarity*, e.g. how much further discussion the comment induced. Formally, we then interpret the judging of comment quality as a multimodal embedding and regression problem.

RQ3: How can the methods developed in RQ1 and RQ2 be effectively integrated into a real-world MCR workflow?

Here, the goal is to develop a practical assistance architecture that integrates the models from *RQ1* and *RQ2* to provide valuable assistance in different phases of MCR processes. We outline two examples for Google's review flow (cf. I): In the *previewing changes* phase, likely comment-locations and topics could be highlighted, and in the *commenting phase*, reviewers would receive feedback on comment usefulness. We anticipate that such assistance should help to increase MCR efficiency, which we intend to evaluate in an empirical study. Another important consideration for this RQ is how to keep models up to date, for example by utilizing online training.

III. Solution Components

We propose a concept involving four major components.

① Dataset: First, the essential data of software and code reviews must be sufficient in quality, quantity, and diversity. Thus, we will mine large-scale online repositories, for example specialized review tools like Gerrit, and, in particular, GitHub pull-requests. We are currently working on a first dataset for *Elasticsearch* that will include comments ($\approx 100k$), their corresponding change hunks, and relevant java file revisions ($\approx 47k$). Datasets must then be curated and labeled with input- and ground-truth-features, e.g. comment meta-topics.

② Input Representation: Second, we will analyze suitable input representations for source code, review comments, and meta-data. We will focus on graph-based program representations, i.e. ASTs, or other specialized representations, for two reasons: First, it has been observed that several program-analysis tasks benefit from structure-aware representations [5], [6]. Second, it allows us to experiment with recent graph-learning algorithms, which achieve state-of-the-art performance for various graph-analysis tasks [7]–[9]. Similarly, for review comments, numerous synthetic representations and representation learning techniques can be explored [10]–[14].

③ Model Design and Evaluation: Third, we want to design AI models that can learn relationships between source code and review data. Following the standard procedure, we intend to split the dataset into training and testing sets. For *RQ1* we will then train models to predict, e.g., whether the AST of a particular change will be commented on or what such a comment's meta-topic will likely be. Example models for *RQ2* will either process only comments, or jointly process comments and ASTs, to estimate how helpful a comment may be, first generally, and then respecting a particular change. For all evaluations we aim to compute standard accuracy metrics on the separate testing data to detect over-fitting.

④ Practical Evaluation: Fourth, we will implement relevant parts of the architecture from *RQ4*, probably as plugins to existing MCR tools. Then, we want to perform a case-study with professional developers to assess the real-world impact on efficiency and to gain insights into developer acceptance.

IV. Related Work

Many studies have analyzed aspects of the effectiveness of MCR [15]–[26]. However, proving MCRs effectiveness is not our research focus. Instead, for *RQ2* we are interested in what metrics they defined to quantify MCR effectiveness, e.g. good code-reviews uncover bugs or improve the software design.

An important distinction to previous work on code-review analysis is that our envisioned methodology will learn to directly correlate source code and review data. To the best of our knowledge, this has not been successfully attempted before. The most similar, yet still conceptually different, related approach analyzed how reviews can be mapped between projects via code clone detection [5].

Some review datasets which we can build on have been previously published [27]–[29]. However, after our preliminary screening, we believe that further repository mining is necessary to meet the particular needs of our research. Particularly *CROP* [27] gives important insights into the challenges and pitfalls of constructing review datasets.

Regarding neural and graph-based program-representations, we can draw on many promising, recent approaches [6], [30]–[33]. Further, various general graph-learning algorithms could be adapted, e.g. graph-convolutional- and graph-attention-networks [8], [9]. A recent survey is also available [7].

For representing comments, many existing learning techniques for text could be relevant, particularly word embeddings [14], RNN- [11], CNN- [12], and attention-models [13].

For both representation learning tasks, we are particularly interested in attention-based approaches [13] for their superior trainability and ability to focus on relevant sub-structures of graphs and long sequences.

REFERENCES

[1] 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*, ser. ICSE-SEIP '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 181–190.

[2] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. IEEE Press, 2013, p. 712–721.

[3] 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*, ser. ESEC/FSE 2013. New York, NY, USA: Association for Computing Machinery, 2013, p. 202–212.

[4] 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, Baltimore, Maryland, USA, October 10-11, 2013*. IEEE Computer Society, 2013, pp. 133–142.

[5] C. Guo, D. Huang, N. Dong, Q. Ye, J. Xu, Y. Fan, H. Yang, and Y. Xu, "Deep review sharing," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 61–72.

[6] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: learning distributed representations of code," *Proceedings of the ACM on Programming Languages*, vol. 3, pp. 40:1–40:29, 2019.

[7] F. Chen, Y. Wang, B. Wang, and C. C. J. Kuo, "Graph representation learning: A survey," *APSIPA Transactions on Signal and Information Processing 9 (2020) e15*, 2019.

[8] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[9] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *CoRR*, vol. abs/1710.10903, 2017.

[10] K. S. Jones, "Index term weighting," *Information Storage and Retrieval*, vol. 9, no. 11, pp. 619–633, nov 1973.

[11] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014, pp. 3104–3112.

[12] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, A. Moschitti, B. Pang, and W. Daelemans, Eds. ACL, 2014, pp. 1746–1751.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017, pp. 5998–6008.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., vol. 26. Curran Associates, Inc., 2013, pp. 3111–3119.

[15] J. Czerwonka, M. Greiler, and J. Tilford, "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*, ser. ICSE '15. IEEE Press, 2015, p. 27–28.

[16] O. Kononenko, O. Baysal, and M. W. Godfrey, "Code review quality: How developers see it," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1028–1038.

[17] M. Caulo, B. Lin, G. Bavota, G. Scanniello, and M. Lanza, "Knowledge transfer in modern code review," in *Proceedings of the 28th International Conference on Program Comprehension*, ser. ICPC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 230–240.

[18] 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, apr 2015.

[19] 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)*, 2015, pp. 171–180.

[20] 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)*, 2015, pp. 81–90.

[21] 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 Transactions on Software Engineering*, vol. 35, no. 4, p. 534–550, Jul. 2009.

[22] 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*, ser. MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 192–201.

[23] 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 2015, Bremen, Germany, September 29 - October 1, 2015*, R. Koschke, J. Krinke, and M. P. Robillard, Eds. IEEE Computer Society, 2015, pp. 111–120.

[24] A. Edmundson, B. Holtkamp, E. Rivera, M. Finifter, A. Mettler, and D. Wagner, "An empirical study on the effectiveness of security code review," in *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, pp. 197–212.

[25] M. Paixao, J. Krinke, D. Han, C. Ragkhitwetsagul, and M. Harman, "The impact of code review on architectural changes," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

[26] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 146–156.

[27] M. Paixao, J. Krinke, D. Han, and M. Harman, "Crop: Linking code reviews to source code changes," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, 2018, pp. 46–49.

[28] X. Yang, R. G. Kula, N. Yoshida, and H. Iida, "Mining the modern code review repositories: a dataset of people, process and product," in *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*, M. Kim, R. Robbes, and C. Bird, Eds. ACM, 2016, pp. 460–463.

[29] M. Mukadam, C. Bird, and P. C. Rigby, "Gerrit software code review data from android," in *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013*, T. Zimmermann, M. D. Penta, and S. Kim, Eds. IEEE Computer Society, 2013, pp. 45–48.

[30] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstract syntax tree," in *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, J. M. Atlee, T. Bultan, and J. Whittle, Eds. IEEE / ACM, 2019, pp. 783–794.

[31] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, "Convolutional neural networks over tree structures for programming language processing," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, D. Schuurmans and M. P. Wellman, Eds. AAAI Press, 2016, pp. 1287–1293.

[32] L. Chen, W. Ye, and S. Zhang, "Capturing source code semantics via tree-based convolution over api-enhanced AST," in *Proceedings of the 16th ACM International Conference on Computing Frontiers, CF 2019, Alghero, Italy, April 30 - May 2, 2019*, F. Palumbo, M. Becchi, M. Schulz, and K. Sato, Eds. ACM, 2019, pp. 174–182.

[33] H. Wei and M. Li, "Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, C. Sierra, Ed. ijcai.org, 2017, pp. 3034–3040.