AI Assisted/Automated code refactoring

How the development of AI may impact the future of code refactoring

Loris Tomassetti
Linz, Austria
loris.tomassetti@outlook.com

Alexander Weißenböck
Linz, Austria
alewei934@gmail.com

***Abstract*—This Paper aims to shed light onto developments in AI Assisted/Automated Code refactoring, how it can help the indtustry and which models work most efficiently to tackle different challenges code refactoring brings. This will be done by going over various literature describing first the challenges at hand and afterwards discussing several possible solutions that have been tested to gain a greater understanding and to generate an informed outlook into further developments of this technology.**

***Index Terms*—machine learning algorithms, software code refactoring, deep neural network**

I. Introduction

Refactoring, as defined by Fowler [8], is "the process of changing a software system in such a way that does not alter the external behavior of the code yet improves its internal structure". More and more empirical studies have since established a positive correlation between refactoring operations and code quality metrics. All this evidence hints at refactoring being a high-priority concern for software engineers. [1].
However, deciding when and how to refactor can prove to be a challenge for developers. Refactoring in an early stage may be cost too much for what youre getting out of it, and refactoring too late may cause the refactor to be an even bigger time comittment. [11]
Tools have been in the hands of many developers to make this process more streamlined for years now. Analytics tools to sniff out bugs or give hints on how to improve code quality such as PMD, ESLint, and Sonarqube can be integrated in different stages of a developers' workflow, e.g. inside IDEs, during code review or as an overall quality report. [1]
Taking a closer look at these tools, however, reveals that they commonly have a lot of false positives, making developers lose their confidence in them. Often, the detection strategies are based on hard threshholds of just a handfull of metrics, such as lines of code in a file (e.g. PMD's famous "problematic" classification occuring once a method reaches 100 lines per default). These simplistic ways of detection simply aren't able to capture the full complexity of modern software systems.
Manually analyzing hundreds of metrics and figuring out which ones are the cause of technical dept is very hard and almost impossible for tool-developers, which is where machine learning-based solutions come into play.
We will take a closer look at how exactly different ML-Models go about this task in section III

References

[1] Mauricio Aniche, Erick Maziero, Rafael Durelli, and Vinicius HS Durelli. The effectiveness of supervised machine learning algorithms in predicting software refactoring. *IEEE Transactions on Software Engineering*, 48(4):1432–1450, 2020.
[2] Abdulrahman Ahmed Bobakr Baqais and Mohammad Alshayeb. Automatic software refactoring: a systematic literature review. *Software Quality Journal*, 28(2):459–502, Jun 2020.
[3] Gabriele Bavota, Andrea De Lucia, Andrian Marcus, and Rocco Oliveto. Recommending refactoring operations in large software systems. In *Recommendation Systems in Software Engineering*, pages 387–419. Springer, 2013.
[4] Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. Programming is hard - or at least it used to be: Educational opportunities and challenges of ai code generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2023, pages 500–506, New York, NY, USA, 2023. Association for Computing Machinery.

[5] Saikrishna Chinthapatla. Unleashing the future: A deep dive into ai-enhanced productivity for developers. *Journal Homepage: http://www.ijmra.us*, 13(03), 2024.

[6] Sethukarasi Thirumaaran Chitti Babu Karakati. Issue information. *Concurrency and Computation: Practice and Experience*, 35(4):e7073, 2023. e7073.

[7] Kayla DePalma, Izabel Miminoshvili, Chiara Henselder, Kate Moss, and Eman Abdullah AlOmar. Exploring chatgpt's code refactoring capabilities: An empirical study. *Expert Systems with Applications*, 249:123602, 2024.

[8] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Signature Series (Fowler). Pearson Education, 2018.

[9] Saki Imai. Is github copilot a substitute for human pair-programming? an empirical study. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, pages 319–321, 2022.

[10] Chitti Babu Karakati and Sethukarasi Thirumaaran. Software code refactoring based on deep neural network-based fitness function. *Concurrency and Computation: Practice and Experience*, 35(4):e7531, 2023.

[11] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. Technical debt: From metaphor to theory and practice. *Ieee software*, 29(6):18–21, 2012.

[12] Thainá Mariani and Silvia Regina Vergilio. A systematic review on search-based refactoring. *Information and Software Technology*, 83:14–34, 2017.

[13] Yiming Tang, Raffi Khatchadourian, Mehdi Bagherzadeh, Rhia Singh, Ajani Stewart, and Anita Raja. An empirical study of refactorings and technical debt in machine learning systems. In *2021 IEEE/ACM 43rd international conference on software engineering (ICSE)*, pages 238–250. IEEE, 2021.

[14] Burak Yetiştiren, Işık "Ozsoy, Miray Ayerdem, and Eray T"uz"un. Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt. *arXiv preprint arXiv:2304.10778*, 2023.