<u>Cameron Katz (100531516) articles:</u>

1)

Sehgal, R., Mehrotra, D., Nagpal, R., & Sharma, R. (2022). Green software: Refactoring approach. *Journal of King Saud University - Computer and Information Sciences, 34(7),* 4635–4643. https://doi.org/10.1016/j.jksuci.2020.10.022

The study focuses on the significance of refactoring for reducing the energy consumption of software, particularly Java-based applications. By focusing on "code smells," which are inefficiencies in code that can lead to excess power usage, the research outlines a methodical approach to identify these code smells and apply refactoring techniques to correct them. Using tools like Microsoft Joulemeter to measure energy consumption before and after refactoring, the study provides empirical evidence that refactoring can significantly reduce power consumption. This finding is crucial as it not only enhances software performance but also contributes to energy efficiency which is an essential aspect in the era of sustainable technology development. This makes the study highly relevant to both refactoring code in software engineering and the broader goal of green computing.

2)

Almogahed, A., Omar, M., & Zakaria, N. H. (2022). Refactoring Codes to Improve Software Security Requirements. *Procedia Computer Science, 204,* 108–115. https://doi.org/10.1016/j.procs.2022.08.013

This study explores the impact of refactoring techniques on software security, specifically focusing on information hiding. While previous research has extensively examined the effects of refactoring on software quality attributes like maintainability and testability, the relationship between refactoring and security was relatively understudied. The research identifies specific refactoring techniques that can influence software security and categorizes them based on their impact on security metrics. This categorization assists developers in selecting appropriate refactoring strategies to enhance software security effectively. The findings of this study are crucial as they fill a gap in understanding how refactoring can be strategically used to improve security aspects of software, making it an essential contribution to both research on refactoring and practical application in software development.

<u>Emily Chang (100531462) Articles:</u>

1)

DePalma, K., Miminoshvili, I., Henselder, C., Moss, K., & AlOmar, E. A. (2024a). Exploring CHATGPT's code refactoring capabilities: An empirical study. *Expert Systems with Applications*, *249*, 123602. https://doi.org/10.1016/j.eswa.2024.123602

The article "Exploring ChatGPT's code refactoring capabilities: An empirical study" evaluates the effectiveness of ChatGPT in software engineering, specifically in code refactoring. The researchers assessed whether ChatGPT could refactor code, preserve the behavior of the original code, and provide documentation for refactored code. They used eight quality attributes to analyze refactoring of 40 Java code segments. The study found that ChatGPT generally made beneficial minor modifications, such as improving variable names or code formatting, and was able to refactor effectively 39 out of 40 times. However, it struggled with complex errors and operations, showing inconsistency in its output, which raises concerns about its reliability. Despite these limitations, ChatGPT proved useful in aiding programming tasks, although it cannot entirely replace human oversight. The study highlights both the potential and the limitations of using AI for code refactoring.

2)

M. Wyrich and J. Bogner, "Towards an Autonomous Bot for Automatic Source Code Refactoring," 2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE), Montreal, QC, Canada, 2019, pp. 24-28, doi: 10.1109/BotSE.2019.00015

Wyrich and Bogner's work discusses the development and potential of the RefactoringBot, an autonomous software bot designed to improve source code quality by automatically refactoring code to address code smells. Unlike traditional approaches, the RefactoringBot integrates seamlessly into developers' workflows via existing version control systems like GitHub, where it autonomously performs refactorings and submits pull requests for review. This integration allows developers to review changes asynchronously without disrupting their current tasks, thereby enhancing productivity and maintainability of the code without requiring significant effort for configuration or integration into existing development environments. We elaborate on the challenges of manual refactoring, the limitations of existing automated tools, and how the RefactoringBot proposes a practical solution by blending into the developers' existing workflow, thereby encouraging higher acceptance and utilization.

John Bettinger (100531510) articles:

1)

“Using Machine Learning to Guide the Application of Software Refactorings: A Preliminary Exploration | Proceedings of the 6th International Workshop on Machine Learning Techniques for Software Quality Evaluation.” *ACM Conferences*, 2022, dl.acm.org/doi/abs/10.1145/3549034.3561178. Accessed 25 Apr. 2024.

The article discusses how machine learning can help improve the process of refactoring in software development. Specifically, the researchers looked at when refactoring would be most advantageous. Refactoring at one point or another in the development process can have beneficial or adverse impacts on the maintainability of the code. Knowing the outcome of when you decide to refactor isn’t always predictable and these researchers sought to create a prediction model.

In the study, the authors used a machine learning model trained with a large amount of data from previous software projects to predict the outcomes of refactoring efforts. Their model can forecast whether changes made during refactoring will have a positive, negative, or neutral impact on the software's maintenance. They specifically looked at the impact on Technical Debt (TD) interest which refers to the extra effort required to maintain and update software. This prediction helps software developers make better decisions about when to refactor, ensuring that their efforts improve the software's operation and maintenance rather than accidentally causing issues.

The model provides insights into the effects of refactoring which in turn support developers in making decisions about when and where to implement their refactorings. This approach not only saves time and resources but also helps improve the quality and maintainability of the software. Using machine learning offers a systematic and data-driven approach to managing software development.

2)

DePalma, Kayla, et al. “Exploring ChatGPT’s Code Refactoring Capabilities: An Empirical Study.” *Expert Systems with Applications*, vol. 249, Elsevier BV, Sept. 2024, pp. 123602–2, https://doi.org/10.1016/j.eswa.2024.123602. Accessed 25 Apr. 2024.

This article investigates ChatGPT's ability to assist in the code refactoring process. In the study, the researchers evaluated ChatGPT’s efficacy across various metrics. They demonstrated that ChatGPT successfully refactored code in 39 out of 40 tests. It was able to perform well in a range of improvements from simple naming conventions to more complex structural changes. It confirmed that in 311 out of 320 trials, the refactored code preserved its original functionality. This indicates that the changes did not negatively impact the software's operations. Additionally, the accuracy of ChatGPT in generating documentation for the refactored code was acceptable, with only a few inaccuracies noted in the documentation comments. Overall, the findings confirmed that ChatGPT can be a helpful tool for automating aspects of the refactoring process. However, the researchers also highlighted the necessity for human oversight. The research concluded that ChatGPT could not yet fully replace human programmers, primarily due to occasional errors and inconsistencies in its outputs. The researchers emphasized the need for careful review to ensure the quality and relevance of refactoring suggestions.

Jacob Schorr (100531487) Articles:

1)

Concurrency and Computation: Practice and Experience: Software code refactoring based on deep neural network-based fitness function, Chitti Babu Karakati, Sethukarasi Thirumaaran. November 23, 2022
https://onlinelibrary.wiley.com/doi/full/10.1002/cpe.7531?casa_token=TXRvYvyKkJoAAAAA%3A_rZdzDvbN7vOzAfUGAt6okOjOBrAgjhcHLdGDllcNTzeS8bxXQF__ZUG1CGBEW19Q131v8P4snaBykE.

This article discusses the ability to refactor code using a model based on deep neural networks (DNNs)). These networks are implemented with a genetic algorithm, and are tasked with the goal of refactoring code to enhance its internal structure while keeping its external behavior the same. The authors created a fitness function driven by deep learning to consistently assess the quality of refactoring solutions. The genetic algorithms generate potential refactoring solutions, and the efficacy is evaluated by the DNN function. The approach automates the detection of code refactoring opportunities and recommends possible solutions. This enhances various metrics that are used to evaluate the structure of code such as execution time or defect correlation ratio. The study evaluates the model empirically and shows that it performs better than traditional methods using real-world datasets to validate its effectiveness. The results show that implementing a neural network based function to refactor code can increase code quality and performance while keeping accuracy very high. This could be a great alternative for those who want to spend less time refactoring while still getting a high quality final product.

2)

Software: Evolution and Process–Model-based source code refactoring with interaction and visual cues, Iman Hemati Moghadam, Mel Ó Cinnéide, Ali Sardarian, Fazeh Zarepour. 13 July 2023

https://onlinelibrary.wiley.com/doi/10.1002/smr.2596

This article discusses an approach to refactoring that they call ‘model-based source code refactoring,’ which combines interaction with the design and automated methods. They use two tools that are Design-Imp, and Code-Imp. Design-Imp allows developers to interactively modify and experiment with a UML model of the software design, providing visual cues to help understand the current state of the design and impact of possible refactoring. Code-Imp automates refactoring to align with whatever design the user picked, and ensures the behavior remains the same. They tested this approach with professional Java developers and compared it to a version without visual cues and an automated method. The model-based approach was found to be much more effective in producing coherent and productive refactorings. Also, the participants reported positive experiences when using the system. This model-based refactoring makes it possible to get a design that meets quality metrics and also aligns with the developer's understanding and goals. This interactive tool can be very beneficial to someone who would like a more intuitive way to design and refactor their code.