Guided Exercise 3 Report

Software Development

Group 89 - Team 06

Full Name	**Student ID**
Marc Dotto	100529268
Joseph Papagno	100531712
Dustin Noonan	100530856

UC3M, April 26th, 2024

1 | Document 1

1.1 | Citation:

Nyamawe, A. S. (2023). Research on mining software repositories to facilitate refactoring. WIREs Data Mining and Knowledge Discovery, 13(5), e1508. https://doi.org/10.1002/widm.1508

1.2 | Summary:

Ally S. Nyamawe's research on leveraging mining software repositories (MSR) to facilitate refactoring is a valuable and insightful extension of what we've learned in class about refactoring, dealing with code smells, and class methods.

- Identifying Refactoring Opportunities: In class, we learned about various code smells and techniques for identifying areas of code that could benefit from refactoring. Nyamawe's research suggests that mining software repositories can provide additional insights into these opportunities as there is a breadth of data available to recognise and build off patterns to address common issues. By analyzing version control history, issue tracker data, and code review comments, developers can isolate specific areas of the codebase that have exhibited maintenance challenges or frequent changes, indicating potential refactoring candidates, ultimately providing frameworks for potential refactoring opportunities.
- Prioritizing Refactoring Efforts: Nyamawe's research also suggests that MSR techniques can help prioritize refactoring efforts based on real-world usage patterns and user feedback. For example, analyzing issue tracker data can reveal which parts of the codebase are associated with the most reported bugs or user complaints, guiding developers to focus their refactoring efforts where they are most needed. This would be useful when tackling a project such as the 3rd guided exercise since prior to the in class examples there was no clear path forward on where or what needed to be fully refactored. Thus through identifying patterns and key areas, the refactoring process was simplified.
- Validating Refactoring Decisions: Additionally, one key componet of GE3's refactoring process was to improve design, readability, and overall performance. Nyamawe's research can provide an additional layer of validation for these refactoring decisions as she indicates by analyzing data from software repositories, developers can assess the impact of their refactoring efforts over time. For instance, they can track metrics such as code churn (frequency of code changes), bug-fixing commits, or code review feedback before and after refactoring to evaluate the effectiveness of the changes.

Overall, Nyamawe's research extends the concepts we have learned about in class by demonstrating how mining software repositories can complement traditional refactoring techniques. By leveraging MSR techniques, as developers, we can gain deeper insights into our codebases, prioritize refactoring efforts and validate refactoring decisions we have made ultimately leading to improved software quality and maintainability.

2 | Document 2

2.1 | Citation:

Oliveira, E., Keuning, H., & Jeuring, J. (2023, June). Student code refactoring misconceptions. In Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (pp. 19-25).

2.2 | Summary:

Eduardo Oliveira's research on student code refactoring misconceptions offers valuable insights that have definitely complemented and enhance your understanding of key concepts covered class. Oliviera's research on the misconceptions or "a systematic pattern of errors" were very fascinating to learn more about as I found that in the beginning passes of refactoring, all the outlined errors were very commonly made however led to poor code quality or sometimes even extra errors that needed to be addressed at a later time when going back and addressing bugs within the code. These misconceptions were more basic examples to try an shorten the code however all were noteworthy and can be further adapted to content covered in this course when it comes to preserving functionality over readability and code length.

Overall, Oliveira's research underscores the importance of teaching students about refactoring, even in the context of smaller programs they develop as beginners. By learning about common misconceptions in refactoring at all stages of the learning process, this allows a student to enhance code quality, ensure that refactored code maintains or improves its original functionality while enhancing readability, performance, and maintainability. He also provides helpful documentation and links to possible softwares that aid a student in their refactoring practices and teach correct methods and processes to ensure code is refactored correctly which is very useful when carried over to concepts within this course.

3 | Document 3

3.1 | Citation:

White, J., Hays, S., Fu, Q., Spencer-Smith, J., & Schmidt, D. C. (2023). Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. arXiv preprint arXiv:2303.07839.

3.2 | Summary:

In this article, the writer is examining the most effective ways to prompt LLM's (language learning models) to create new code and refactor existing code. For example, the writer first examines the behavior of using the phrase, "from now on," to automate the LLM to always preform an operation in the future. The focus is on examining patterns to get an LLM to accomplish software engineering tasks that would normally be done by a software engineer. One pattern in particular that is examined is the Data-guided Refactoring Pattern. This differs from other patterns since it does not ask the LLM to change any logic, only to change the schema that the code is written in. "In many cases, the refactoring can be completely automated through this process (Data-guided Refactoring Pattern), or at least bootstrapped" (page 13). This article is relevant to the assignment because it focuses on the best way to automate the work asked in the project description, refactoring. LLM's are powerful tools that can be used to automatically accomplish work that will take a human more time to do. However, to reach this potential, we need researchers to dive deep into how an LLM works by examining different prompts effectiveness. This article gave a deeper understanding of what prompts work well and potential consequences of each.

4 | Document 4

4.1 | Citation:

Naik, P., Nelaballi, S., Pusuluri, V. S., & Kim, D. K. (2024). Deep Learning-Based Code Refactoring: A Review of Current Knowledge. Journal of Computer Information Systems, 64(2), 314–328. https://doi.org/10.1080/08874417.2023.2203088

4.2 | Summary:

This paper focuses on leveraging machine learning to refactor code in a high level and deep way. It makes the argument that if you can set up good design principles and patterns at an early and high level in a project, that you can eliminate needing to worry about rudimentary code "smells" when refactoring. Specially this article focuses on creating deep neural network architectures to figure out how to refactor based on object-oriented design principles over the classic machine learning models. The article explains and references several studies involving artificial intelligence and machine learning models to show the capabilities of a deep neural network. Building a deep neural network to detect design patterns can be beneficial in many ways. In relation to our project, a design pattern detection model could implement the Attribute or Singleton Pattern given jumbled code. This kind of refactoring is more advanced than normal but would help implement patterns into our code when otherwise it would take a team of software engineers. I would be specifically impressed if a machine learning model could recognize the core similarities between the three methods in Hotel Manager and be able to abstract it into general Json Store classes, all while maintaining the same functionality of the original jumbled code. These similarities are hard to spot on the outside and require us to look deeper into the purpose and components of the code, something an advanced neural network could accomplish. This article gave an insight into the current limits of machine learning and its potential power of deep refactoring.

5 | Document 5

5.1 | Citation:

Mashaqbeh, Khitam, et al. ”Correspondence of OOP Refactoring Techniques in PL¡ SQL Environment: A Case Study of Agile Project Code.” Appl. Math 17.6 (2023): 993-1002. https://digitalcommons.aaru.edu.jo/cgi/viewcontent.cgi?article=3364&context=amis

5.2 | Summary:

The article presents an experimental study examining the impact of refactoring techniques on code quality within the context of Scrum sprints and software maintainability. Through a comprehensive literature review, it addresses the lack of studies on refactoring and code quality improvement, emphasizing the need for further research in this area. The study investigates various refactoring techniques applied to sprint code, identifying nine techniques that positively influence code maintainability, including replacing methods with method objects and decomposing conditionals. These findings are valuable for Scrum practitioners and PL / SQL developers in the industry, providing practical insights into enhancing code quality and maintainability, particularly in dynamic development environments. In relation to this Guided Exercise, this article offers valuable insight into the real world of development in the industry and how these techniques are being used and will be relevant beyond this class.

6 | Document 6

6.1 | Citation:

Tan, Ivan, and Christopher M. Poskitt. ”Fixing Your Own Smells: Adding a Mistake-Based Familiarisation Step When Teaching Code Refactoring.” (2024). https://cposkitt.github.io/files/publications/mistake-based_refactoring_sigcse24.pdf

6.2 | Summary:

The article presents insights from participants and the implementation of a ’mistake-based’ teaching approach for refactoring. Participants shared stories about the value of making mistakes in learning, validating the effectiveness of their approaches. The study found that students were more proficient and confident with this methodology compared to traditional teaching methods. Challenges remain in designing exercises to balance learning objectives and opportunities for mistakes. Future research could explore automation in generating exercises. The findings suggest broader applications for mistake-based teaching in software engineering education beyond refactoring, such as SQL/database management and web development. This relates to the Guided Exercise because the purpose of this assignment was to experiment and learn refactoring techniques. Therefore it is interesting that there are many different strategies to learning these practices.