Software quality is crucial for the success of software systems, and detecting and addressing code smells is an important aspect of ensuring high-quality software development. Code smells are indicators of potential design problems or quality issues in the source code of a software system. They can occur in the code of both novice and experienced coders and can lead to decreased software maintainability and increased development time.

In this graduate project, we aim to explore the concept of code smells and their impact on software quality. Specifically, we will investigate the different types of code smells and the approaches used to detect them. Additionally, we will examine whether there is a difference in the occurrence of code smells between students and professional coders, and how early detection and resolution of code smells can impact the quality of the software.

The project will be structured as follows. First, we will provide an overview of code smells and their types. Next, we will discuss the approaches used to detect code smells, including the use of static code analysis tools such as PMD and other four tools. We will then compare the occurrence of code smells between students and professional coders and investigate the impact of code smells on software quality. Finally, we will conclude the project by discussing the importance of early detection and resolution of code smells and propose future research directions in this field.

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

Chapter 2: Literature Review

In this chapter, we will review the existing literature on code smells and their impact on software quality. Specifically, we will investigate the different types of code smells and the approaches used to detect them. Additionally, we will examine the occurrence of code smells between students and professional coders and the importance of early detection and resolution of code smells.

2.1 Types of Code Smells Code smells are indicators of potential design problems or quality issues in the source code of a software system. The literature categorizes code smells into various types, including Long Method, Duplicated Code, Large Class, and more. Each type of code smell is indicative of a specific design problem and requires a unique approach to resolution.

2.2 Approaches to Detect Code Smells Static code analysis tools such as PMD and other four tools can be used to detect code smells. These tools analyze the source code and provide a report of identified code smells for review and improvement. Various studies have shown that the use of static code analysis tools can significantly improve software quality by detecting and resolving code smells early in the development process.

2.3 Code Smells Between Students and Professional Coders Studies have shown that both students and professional coders can make mistakes in their code that lead to design problems and quality issues, which are identified as code smells. However, the occurrence and types of code smells may differ between the two groups. For example, students may be more likely to introduce certain types of code smells due to their limited experience, while professional coders may be more likely to introduce code smells related to complex systems.

2.4 Importance of Early Detection and Resolution of Code Smells Early detection and resolution of code smells are crucial to preventing potential design problems and improving the overall quality of the software. This is true for both students and professional coders, and regular practice of detecting and resolving code smells is important to ensure high-quality software development. Studies have shown that the use of static code analysis tools can significantly improve software quality by detecting and resolving code smells early in the development process.

In conclusion, this chapter has provided an overview of the existing literature on code smells and their impact on software quality. We have investigated the different types of code smells and the approaches used to detect them. Additionally, we have examined the occurrence of code smells between students and professional coders and the importance of early detection and resolution of code smells. This literature review provides a strong foundation for the subsequent chapters of this project.

2.2 Approaches to Detect Code Smells Static code analysis tools such as PMD and other four tools can be used to detect code smells. These tools analyze the source code and provide a report of identified code smells for review and improvement. Additionally, these tools can be integrated into the development environment to provide real-time feedback to developers as they write code.

The use of static code analysis tools has been shown to significantly improve software quality by detecting and resolving code smells early in the development process. These tools can help developers identify and fix code smells before they become more difficult and expensive to address later in the development process. However, it's important to note that these tools are not perfect and may not catch all code smells or may produce false positives. Therefore, it's important to combine the use of static code analysis tools with manual code reviews and other quality assurance techniques.

2.3 Code Smells Between Students and Professional Coders Research has shown that both students and professional coders can introduce code smells into their code. However, the occurrence and types of code smells may differ between the two groups. For example, students may be more likely to introduce certain types of code smells due to their limited experience, while professional coders may be more likely to introduce code smells related to complex systems.

Studies have also shown that the experience level of the developer can impact the effectiveness of static code analysis tools in detecting code smells. Inexperienced developers may not be familiar with the types of code smells that exist or how to interpret the results from static code analysis tools. On the other hand, experienced developers may be more efficient at using static code analysis tools to identify code smells and may be more likely to address them before they cause problems.

It's also important to consider the impact of organizational culture on the occurrence of code smells. A culture that values code quality and invests in developer training and tooling may have fewer code smells compared to a culture that prioritizes rapid development and meeting deadlines over code quality. Therefore, understanding the organizational context is important when investigating the occurrence of code smells.

2.4 Importance of Detecting Code Smells Early Detecting code smells early in the development process is important for several reasons. First, it can help prevent the introduction of further code smells and reduce the technical debt associated with maintaining and updating code with code smells. Second, it can improve software quality and reduce the likelihood of bugs and other issues. Finally, it can save time and money by reducing the need for expensive rework and refactoring later in the development process.

However, it's important to note that detecting code smells early is not always easy. Inexperienced developers may not be familiar with the types of code smells that exist or how to detect them. Additionally, some code smells may not manifest until later in the development process or may only become apparent during testing or in production.

Conclusion

In conclusion, code smells are a common issue in software development that can have negative impacts on software quality, maintainability, and cost. The use of static code analysis tools, such as PMD and other four tools, can help developers detect and resolve code smells early in the development process, but it's important to combine this with manual code reviews and other quality assurance techniques.

While both students and professional coders can introduce code smells into their code, the occurrence and types of code smells may differ between the two groups. Therefore, it's important to consider the experience level of the developer and the organizational culture when investigating the occurrence of code smells.

Detecting code smells early is important for improving software quality, reducing technical debt, and saving time and money. However, detecting code smells early can be challenging, particularly for inexperienced developers. Therefore, continued research and investment in developer training and tooling is needed to improve the detection and prevention of code smells in software development.

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

CHAPTER 3---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

Chapter 3: Methodology

This chapter describes the methodology used to investigate the occurrence of code smells in software developed by students and professionals. The chapter includes a description of the research design, data collection, and data analysis methods used in this study.

3.1 Research Design The research design for this study is a comparative analysis of code smells in software developed by students and professionals. The study will use a mixed-methods approach, including both quantitative and qualitative data collection and analysis methods.

3.2 Data Collection The data for this study will be collected from two sources: student-developed software and professional-developed software. The student-developed software will be collected from a sample of computer science students in their final year of study who have completed a software development project. The professional-developed software will be collected from a sample of software development companies. The companies will be selected based on their size, industry, and geographical location.

For both groups, we will collect the source code of the software developed and use static code analysis tools, including PMD and other four tools, to detect code smells. We will also conduct semi-structured interviews with the developers to gather qualitative data on their experiences with code smells.

3.3 Data Analysis The data collected from the two sources will be analyzed separately, using both quantitative and qualitative data analysis techniques. The quantitative analysis will involve statistical analysis of the code smell detection results to compare the occurrence and types of code smells between the two groups. The qualitative analysis will involve thematic analysis of the interview data to identify common themes related to code smells and software development.

3.4 Limitations There are several limitations to this study, including the sample size and the generalizability of the results. The sample size for both groups may be small and may not be representative of the larger population of students or professionals. Additionally, the results may not be generalizable to other industries or types of software development.

Despite these limitations, this study provides valuable insights into the occurrence of code smells in software developed by students and professionals and can inform future research and development efforts to improve software quality and maintainability.

3.1 Research Design:

* Clarify the research question and objectives to ensure that the research design is appropriate for addressing them.
* Justify the choice of a mixed-methods approach and explain how it will enhance the study's findings.
* Consider potential biases and limitations of the research design and attempt to mitigate them.

The research design describes the overall approach that will be used in the study. In this case, we are conducting a comparative analysis of code smells in software developed by students and professionals. This means we will be comparing the occurrence and types of code smells between the two groups to identify any differences or similarities. We will use a mixed-methods approach, meaning we will use both quantitative and qualitative data collection and analysis methods to provide a more comprehensive understanding of the topic.

3.2 Data Collection:

* Clearly define the criteria for selecting participants and the sample size for each group.
* Ensure that the code samples collected are representative of each group's software development practices and are comparable.
* Consider using multiple static code analysis tools to increase the reliability of the results.
* Develop a semi-structured interview guide to ensure that the qualitative data collected is relevant to the research question.

The data collection section describes how we will collect the data for the study. In this case, we will be collecting the source code of software developed by students and professionals. For the student-developed software, we will collect code from a sample of computer science students who have completed a software development project. For the professional-developed software, we will collect code from a sample of software development companies. We will use static code analysis tools, including PMD and other four tools, to detect code smells in the source code. Additionally, we will conduct semi-structured interviews with the developers to gather qualitative data on their experiences with code smells.

3.3 Data Analysis:

* Clearly define the statistical analysis methods that will be used to compare the occurrence and types of code smells between the two groups.
* Use a codebook to guide the thematic analysis of the qualitative data and ensure consistency in the analysis process.
* Consider using a third-party reviewer to ensure the accuracy and reliability of the data analysis.

The data analysis section describes how we will analyze the data collected in the study. We will use both quantitative and qualitative data analysis techniques. For the quantitative analysis, we will use statistical analysis to compare the occurrence and types of code smells between the two groups. For the qualitative analysis, we will use thematic analysis to identify common themes related to code smells and software development. By using both quantitative and qualitative data analysis techniques, we can provide a more comprehensive understanding of the topic.

3.4 Limitations:

* Consider potential limitations of the study from the outset and attempt to mitigate them where possible.
* Be transparent about the study's limitations and potential impact on the generalizability of the results.
* Consider discussing how the study's limitations could be addressed in future research.

The limitations section describes the potential limitations of the study. In this case, the sample size for both groups may be small and may not be representative of the larger population of students or professionals. Additionally, the results may not be generalizable to other industries or types of software development. It's important to identify and acknowledge these limitations to ensure the study's results are interpreted appropriately and not overgeneralized.

Overall, the methodology chapter outlines the research design, data collection, data analysis methods, and potential limitations of the study. It provides a clear roadmap for how the study will be conducted and how the data will be analyzed to answer the research questions.

Overall, it's important to be transparent about the methodology used in the study and to acknowledge any limitations or biases that may affect the results. By carefully designing the research approach, collecting high-quality data, and conducting thorough data analysis, the study can provide valuable insights into the occurrence and types of code smells in software developed by students and professionals.

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

1. Literature Review 2.1 Definition of Code Smells Code smells are a set of common programming issues that can lead to bad code design and structure. These issues can make code difficult to read, maintain, and modify. In recent years, there has been a growing interest in detecting code smells early in the development process, to prevent them from becoming bigger problems down the line.

2.2 Types of Code Smells There are many types of code smells that have been identified in the literature. These include but are not limited to, long methods, duplicated code, complex conditionals, data clumps, feature envy, and large classes. Each of these types of code smells has been studied extensively, and many tools have been developed to detect them automatically.

2.3 Approaches to Detecting Code Smells There are several approaches to detecting code smells. One approach is to use automated tools, such as PMD, FindBugs, and SonarQube, which can analyze the code and detect potential issues. Another approach is to use code review techniques, where a team of developers reviews the code and identifies potential problems. A third approach is to use metrics-based techniques, which measure various aspects of the code to identify potential problems.

2.4 Code Smells in Students and Professional Coders Research has shown that students and professional coders tend to make similar types of code smells. However, there is a lack of research comparing the prevalence of specific code smells between the two groups. This research gap is important because it can help to identify potential areas of focus for code quality education in academic settings. By comparing the types and prevalence of code smells between students and professional coders, we can determine whether certain types of code smells are more common in one group than the other.

In conclusion, this chapter has provided an overview of code smells, their types, and approaches to detecting them. We also discussed the importance of studying code smells in both students and professional coders to identify potential areas for code quality education. In the next chapter, we will describe the methodology used to compare code smells between students and professional coders.

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------

Chapter 3: Methodology

This chapter describes the research design, data collection and analysis, and limitations of the study.

3.1 Research Design The research question for this study is: What are the similarities and differences in the occurrence and types of code smells in software developed by students and professionals? The objective of the study is to identify the most common code smells in each group and to compare the occurrence and types of code smells between the two groups. A mixed-methods approach will be used, which involves collecting both quantitative and qualitative data. This approach will enhance the study's findings by providing a more comprehensive understanding of the research question.

3.2 Data Collection Participants will be selected based on the following criteria: 30 students enrolled in a software engineering course at a university, and 30 professionals with at least three years of experience in software development. The code samples collected will be representative of each group's software development practices and comparable. To detect code smells in the code samples, PMD will be used as the static code analysis tool. The output generated by PMD will be reviewed and the type of code smells detected will be categorized based on the taxonomy proposed by Fowler. In addition, semi-structured interviews will be conducted with 10 participants from each group to collect qualitative data.

3.3 Data Analysis To compare the occurrence and types of code smells between the two groups, a Chi-squared test will be used. The results will be presented using descriptive statistics and visualizations. The qualitative data collected from the semi-structured interviews will be analyzed using thematic analysis. A codebook will be developed to guide the analysis process, and the data will be reviewed by a second reviewer to ensure accuracy and reliability.

3.4 Limitations One potential limitation of the study is the small sample size. To mitigate this, efforts will be made to ensure that the participants are representative of each group's software development practices. Another potential limitation is the use of a single static code analysis tool. To address this, the results will be compared to those obtained using other tools in future research. Finally, the generalizability of the results may be limited by the specific context of the study, but future research could replicate the study in different contexts to increase the generalizability of the findings.

Overall, the study will provide valuable insights into the similarities and differences in the occurrence and types of code smells in software developed by students and professionals. By carefully designing the research approach, collecting high-quality data, and conducting thorough data analysis, the study will help to identify areas for improvement in software development practices for both groups, particularly in the context of software development education.