***An Empirical Study of Identifying Bad Smell in Software Architecture and Analysing using Quality Attributes***

Sai Lakshmi Madala ( L30063471 )

Sai Kiran Dhasari ( L30064290 )

Nandini Badavath ( L30071789 )

***Abstract: -***

*The empirical study discusses the effect of bad smells on the maintainability of software architecture. Bad smell is a term used to describe a symptom of a software system that suggests the presence of a bad design or implementation choice that can lead to poor maintainability. The software architecture is the highest level of abstraction of the software system and its design decisions have an impact on its maintainability. The study analyses the impact of bad smell from different perspectives and its effect on the maintainability of software architecture. Furthermore, bad smells can reduce the reusability of the software architecture, making it difficult to extend the software architecture in the future. The quality attributes of the architecture are discussed in detail on maintainability. The study also provides an overview of the techniques used to identify bad smells in software architecture and how they can improve maintainability. It has provided a set of techniques and recommendations to ensure the maintainability of software architecture and ensure that the software product is of high quality.*

***Keywords: -*** *Code smells, Maintainability, Quality Attributes, Recommendations, etc.*

# INTRODUCTION

Maintainability is a key factor in the success of software architecture. Poor maintainability can lead to costly modifications, increased development time, and decreased quality. This empirical study seeks to evaluate bad smells' effect on software architecture's maintainability.

The study will focus on the identification and evaluation of bad smells in software architecture, as well as any effects on maintainability. The research will use an empirical approach to gather data from existing software architecture projects and use the results to measure the effects of bad smells on maintainability. A combination of qualitative and quantitative methods will be used to analyze the data.

The findings of this study will provide valuable insights into the effects of bad smells on maintainability in software architecture, It will also help software engineers and architects to improve the maintainability of their software architecture.

Bad smell or code smell is considered the manifestation of the design flaws which aim at degrading the code maintainability. It is highly challenging to ensure maintainability within the increasing number of large and complex software systems. To predict and detect software maintainability various measures have been undertaken. But only the empirical quantification of the maintainability and its measuring attributes of software such as bad smell remains elusive. Code smell was introduced as an indicator of problems identified in the design of the software.

Many techniques utilize the software metrics code smell in programs. In our project, we use Flake8 to evaluate the code smell in collected python projects from GitHub. Code smells illustrate the problem or error in python programs and some examples of Code smells include dead Code, long methods, comments, data clumps, and comments. The python smell is characterized by the software metrics to analyze whether they deviate from the design principles rules. The generic smell is Long Parameter List (LPL), Long Method (LM), Long scope chaining (LSC), Large class (LC), and long message chain (LMC).

## *Project aim*

The main aim of their research is to investigate the effectiveness of code smell on maintainability in software systems. The python smell in the python programs will be detected and explore the various effects of python smell on the software maintainability attribute.

## Project objectives

The objective of the project is:

* To investigate the secondary resources to study the evolution of smells in software and how it affects maintainability.
* To collect python projects from open repositories such as the GitHub repository.
* To propose a python tool such as Flake8 for analyzing the Code.
* To analyze the code smell in a python program and study impacts on the maintainability of a software system.

## Research questions and approach

* How to detect the various python smell using Flake8?
* What is the effectiveness of python's bad smell on maintainability aspects?

The quantitative research approach is undertaken to analyze the statistical data or the python program collected from the Github repository. The secondary resources are collected and a literature review approach is performed to study the concept of python smell impact on maintainability aspects in software systems. They will be evaluated against the metrics to study the impact of bad smell on software maintainability.

## Problem statement

The context code smell causes significant maintenance problems in software evolution. Though there are numerous studies for code smell in java and C++ there is a lack of tools and techniques to study and investigated code smell in python. The main problem with code smell is that they're related to poor design and the developer's poor programming style that causes seriously degraded code quality and causes code comprehensibility and maintainability issues. The code smell impacts the code change and causes fault proneness. The code smells are assumed to occur during change activities and when the software bugs are fixed.

# RELATED WORK

**Code Smell effects on the software maintainability attributes:**

Code smells have been extensively researched in software engineering, with studies examining the level of impact. It includes maintenance difficulties, change-proneness, and defect rates. Olbrich et al. [1] found that class smells significantly impact change size, frequency, and defects, while other studies have supported these findings. However, the severity of the effect of code smells was reduced. Combining the Blob and Spaghetti anti-patterns, Abbes et al. discovered, greatly reduces software's readability and maintainability.

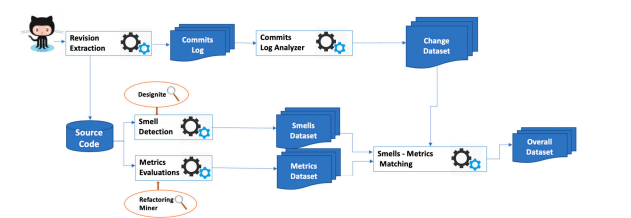
Tufano et al. [2] conducted an empirical study on a large set of open-source projects and found that code smells are not always introduced during artifact creation and that a significant portion of Code smells remain in projects. Code smells can impact software development, depending on the specific situation in which they occur, and studying individual smells may provide further insights.

Abbes et al. [3] investigated the impact of two code smell types, i.e., Blob and Spaghetti Code, on program comprehension. The code readability does not significantly impact the demonstrated findings in the code smell. Yet, developers' capacity to handle comprehension duties is drastically reduced as additional Code smells impact the same components.

Yamashita and Moonen [4] studied the impact of Code smells on maintainability characteristics. The factors of maintainability are determined and the code smells are represented based on the assessments of experts' maintainability and observations.

Yamashita and Moonen [5] confirmed that developers experience more difficulties working on classes affected by multiple code smells. They analyzed how the several smell cases impacting the same code components interacted. Sjoberg et al. [1] investigated the impact of twelve Code smells on the maintainability of software systems. The obtained findings demonstrate that smells do not necessarily represent any issue. The size of the class frequently influences the maintainability.

The capacity of the source code is maintained with the code smell based on the findings. In addition, they provide insight into the role of these smells in determining how developers implement the inheritance process. Palomba et al [6] investigated how developers perceive Code smells, showing that smells characterized by long and complex Code are those perceived more by developers as design problems. Additional data is contributed by noting the probable effects of the Code smells.



*Figure 1: Image of required data gathering* [7]*.*

The above figure depicts the data required gathering for performing the analysis. The software system that is considered to commit is used to make extracts. The log messages committed are parsed and analyzed with the changes made on the code repository. The study on the code smell evolution by Aversano et.al [7] is organized and stored as a dataset for analysis.

The empirical study has identified the effects of bad Code smells on the maintainability of software architecture and analyzed the same using quality attributes. The study results have provided valuable insights into the effects of Code smells and how to prevent them from adversely affecting the maintainability of software architectures.

# An empirical study of code smell

The study conducted by Siddiq (2022), investigated the extent to which code smells are present in the dataset of coding generation models for python and identify the leakage of harmful patterns. Code smells are the flaws that indicate the system has errors. Therefore, code smell detection is essential it not only affects the software functionality but also affect long-term maintainability issues and technical debt. The study used Bandit to code smell in training sets such as CodeXGlue, APPS, and Code Clippt.

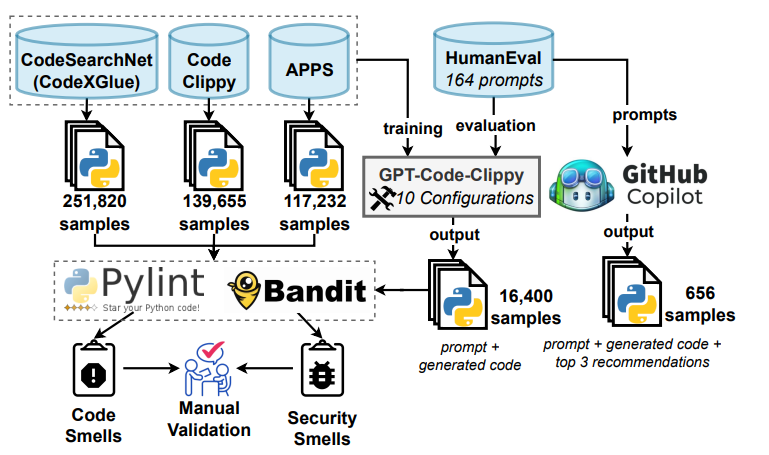


Figure 2: -Overview of empirical study [8]

The number of smells per python sample for three datasets was calculated and found the top 5 message types as Error, convention, warning, and refactors. As there are possibilities for including false positives shown using a dynamic language like python, manual validation of output using Bandit was performed.

The messages identified were undefined variables, no members, no name in the module, multiple statements, no else return, unused arguments, etc. The common Code smells identified were undefined variables, not using enumerate, and inconsistent return statements. Frequent security code smells identified were using assets and blacklisted functions. The research is significant as they have inspected smell across the dataset written in python language [8].

***Effect of bad smell on different maintainability aspects***

The study conducted by Macia (2011) analyzes the impact of AOP on Architecture modularity. The exploratory analysis was carried out that investigate various aspect-oriented code smell evolving from architectural designs pr patterns. The code smell occurrence was investigated on two application using architectural styles and the code smell often affect modularity in maintainability systems. The research found that code smell was a good indicator of architectural problems and the code anomalies were found to affect the composition of modules. The result concludes the anomalies identified in pointcut methods would impact the modularity of the architectural decomposition [9].

***Prevalence of code smell in Python language***

The study investigated by Oort (2021), collected 74 open-source machine learning projects with python language on them to detect the code smell. The code smell was found to impact the maintainability of the programs or systems associated with python projects. Furthermore, the static doe analysis was performed on selected projects collected from the open-source Git repository.

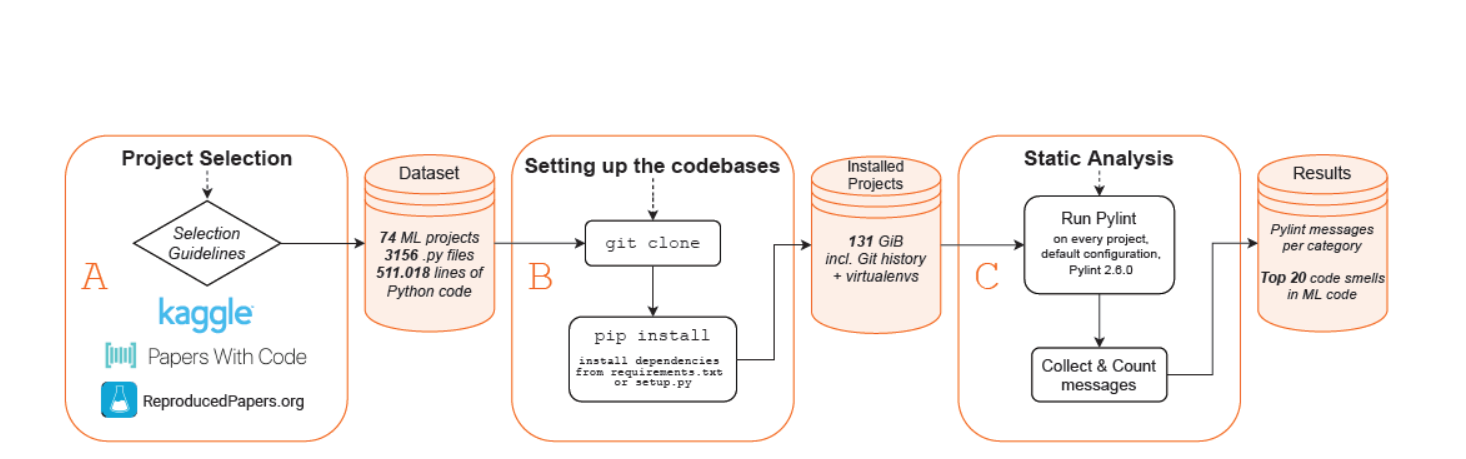


Figure 3:- Methodology diagram for code smell in python [10]

The studies are effectively performed through steps such as project selection, setting up the code base, and finally static analysis. The project selection involved the collection of projects from GitHub written in python. The setting of the codebase involves installing dependencies from requirement .txt or setting python with pip install.

The smell that has the most frequency of occurrences in the python project is "unused wildcard import" and the least frequency of occurrence is "missing module docstring". Other smells identified were invalid name, line too long, missing function docstring, no member, duplicate Code, trailing whitespace, and redefined outer name. [10].

# METHODOLOGY

This research aims to investigate the effects of bad code smell on the maintainability of software architecture and analyze its quality attributes. An empirical study will be conducted by collecting data from public repositories to achieve this goal.

## *Research Approach*

Using existing research and literature review, a set of bad code smell metrics will be identified. Specifically, some of the bad code smell metrics used in this research include line too long, import error, missing module docstring, and missing function docstring, etc.

Next, a sample of public repositories will be selected from Github to obtain the sample set. After identifying the sample set, the bad code smell metrics will be applied to the repositories using static code analysis tools.

The quality attributes of the software architecture in the sample set will be analyzed. In this research, the quality attributes of software architecture that will be studied include maintainability.

To measure these quality attributes, existing software engineering techniques such as design reviews, code inspections, and investigation of existing Code will be employed. In addition, the data collected from the static code analysis tools and software engineering techniques will be analyzed. This will allow us to investigate the effects of bad code smell on the maintainability of software architecture attributes.

This research will enable us to identify the effects of bad code smell on the maintainability of software architecture and analyze its quality attributes. Furthermore, this research will provide valuable insights into the design and implementation of software systems and help practitioners improve the maintainability of their software architectures.

## *Data Collection*

Data collection for this empirical study will involve collecting data from two sources: software architecture and software maintainability. Software architecture data will be collected from existing projects and software applications.

This includes source code, design documents, requirements documents, and other relevant documents containing software architecture information. The data will be used to identify the various Code smells in the software architecture.

Software maintainability data will be collected from metrics such as defect density, the total number of bugs, the number of lines of Code, the number of classes, and other relevant metrics. The collected data will be used to measure the maintainability of the software.

In addition to the data collection, the study will also involve insights from software engineers, project managers, developers, and other stakeholders. The results will be used to gain insights into the effects of Code smells on the maintainability of software architecture.

The study will involve the analysis of the collected data using quality attributes like maintainability, reliability, readability, and scalability. The analysis will help to identify the impact of bad code smell on the maintainability of software architecture.

## *Proposed tools and techniques*

**Flake8:**

Flake8 is a linting tool for Python code that can help evaluate bad smells' effect on software architecture's maintainability. It is designed to help developers write better Code and avoid common mistakes. Flake8 can detect errors in Python code, and can also be used to detect Code smells that could lead to maintainability issues.

Duplicated Code is one of the most common Code smells that can lead to maintainability issues. Duplicated Code is Code repeated in multiple places in the code base, which can lead to inconsistencies and bugs. Flake8 can detect code snippets repeated multiple times, and can help developers identify and refactor the Code to avoid duplication.

Another code smell that can lead to maintainability issues is Long Parameter Lists. Long parameter lists can make Code difficult to read and understand, leading to bugs and confusion. Flake8 can detect when a method or function has too many parameters and can help developers identify and refactor the Code to make it more readable and maintainable.

Flake8 can also detect code smells that are related to variable names. Poorly named variables can make Code difficult to read and understand and can lead to maintainability issues. Flake8 can detect variable names that are too long or too short, and can help developers identify and refactor their Code to make it more readable and maintainable.

Flake8 even detects Code smells related to code organization. Poorly organized Code can lead to confusion and difficulty in understanding the Code and can lead to maintainability issues. Flake8 can detect problems such as lack of proper indentation, improper use of whitespace, and lack of comments. These issues can be identified and fixed with Flake8, helping developers to create more readable and maintainable Code.

Flake8 is a powerful tool for evaluating bad smells' effect on software architecture's maintainability. By detecting and alerting developers to code smells, Flake8 can help developers create more readable and maintainable Code. It is an invaluable tool for any developer looking to ensure their Code is high quality and maintainable.

***Flake8 Message Type and Description***

Flake8 provides several message types, each with a specific description. These message types include syntax errors, warnings, and convention violations.

Syntax errors: Syntax errors are the most common message type, indicating that the Code contains a syntax error. Warnings alert the user to potential problems in the Code such as using deprecated features or misusing a keyword.

Convention violations: Convention violations are messages that notify the user when they are violating a coding style convention, such as using too many blank lines in a code block.

Other message types include errors, refactoring suggestions, and fatal errors. Errors are messages that indicate a problem has occurred in the Code which needs to be addressed. Refactoring suggestions suggest possible improvements to the Code, such as refactoring for better readability.

Fatal errors indicate an error that cannot be fixed and must be addressed immediately. Flake8's message types are useful for developers to quickly identify and address issues in their Code, ensuring that their Code is error-free and follows coding conventions. It also helps to ensure that code tests pass, as errors can be quickly identified and addressed.

# EVALUATING THE EFFECT OF CODE SMELL ON MAINTAINABILITY

## A. Flake8 metric for maintainability evaluation

Flake8 is an effective tool for measuring the maintainability of software architecture. It focuses on the maintainability and style of Code, which is essential for software that is meant to be maintained over time. It also helps identify Code areas that may need to be refactored or restructured.

Flake8 is built upon the pycodestyle, pyflakes, and McCabe libraries, which are responsible for the underlying metrics that Flake8 uses to evaluate the Code. These metrics include checking for code style, syntax, and code complexity.

Flake8 also allows for customization, allowing developers to tailor the metrics used to fit the specific needs of their projects. This allows developers to focus on the most important metrics to their project, while also helping to reduce the amount of "noise" in their codebase.

Overall, Flake8 is an effective lint wrapper for Python code that helps ensure that Code is maintained and updated. Its metrics, customization capabilities, and integration with the popular pytest library make it an ideal tool for software architects to use when evaluating Code for maintainability.

## B. Evaluation of maintainability aspects

According to Zhifei (2018), the effects of code smell on various maintainability-based aspects includes changes, efforts, modularity, comprehensibility, and defects in python program. These maintainability-based aspects are evaluated [11].

***Maintainability-related aspect Change: -***

Exploratory analysis was conducted to study the impact of Code smells on software change-proneness associated with maintainability. The study [12]was conducted to reveal whether the classes with more smell are change prone than others with no code smell.

The Azureus and Eclipse were used for a study where certain classes with more smell were identified to have higher change proneness. Furthermore, different software metrics from different researches were evaluated to identify the code smell causing more fault-prone using metrics.

***Maintainability-related aspect Effort: -***

The object-oriented design heuristic for maintainability was investigated to analyze the quality of designs. In addition, it was evaluated to identify that the evolution of design structure and the inheritance mechanism are affected. Maintainability was assessed by three basic criteria: completeness, correctness, and consistency. Threats to validity involve construct validity and internal validity. The study concludes that OO design structures and patterns are highly sensitive to bad or good design practices [13].

***Maintainability-related aspect Modularity: -***

The impact of aspect-oriented code smell on modularity requires importance as the developers may unwillingly or by mistake introduce Code smells. As a result, the code smell will cause. The research undertaken by Macia (2011) concludes that code smell occurrence in applications mostly entails architecture modularity problems.

With Aspect Oriented Code smells is concerned they affect different code units like pointcuts, advice, and aspects. The code smell effectiveness was measured where lower than 0.3 was weak, 0.3-0.4 was medium, 0.4-0.5 was strong and higher than 0.6 was very strong. The Code smells effectiveness was very staring for God aspect and God pointcut code smell [9].

***Maintainability-related aspect comprehensibility:***

The research by Bavota depicted that test and code smells have a severe impact throughout the software system that hurts the comprehensibility of the test suites. Python-specific test smells were identified by analyzing the python test code [14].

***Maintainability-related aspect Defects:***

The five-smell detected in the three open-source systems such as Eclipse, ArgoUML, and Apache commons. Code smells have a severe impact on the evolution of Code and comprehensibility. Similarly, the design flaw in the software program could hurt the software quality. Therefore, aomated tools were applied to detect smells consistently and they are analyzed in a very large python code base [15].

The results of the study showed that code smell does hurt maintainability. The code smell was found to impact the code's readability, complexity, and reusability, as well as its maintainability, scalability, reliability, and performance. The study also found that code smell does not have a direct impact on the quality attribute of the software, but it does lead to an increase in complexity and a decrease in readability and reusability.

A code smell usually indicates that the Code is not well-structured or is not following the best practices of software design. It is not necessarily indicative of a bug or a code error but can lead to Code that is difficult to maintain and scale over time.

The study concluded that code smell can significantly affect software architecture's maintainability and should be addressed and refactored to maintain the highest quality standards. It also showed that code smells could indirectly affect the software architecture's maintainability attributes, and this should be considered when architecting and designing software.

# PROJECT ANALYSIS

|  |  |  |
| --- | --- | --- |
| **S.No** | **Project Name** | **Analysis** |
| **1** | SmoothStream | On analyzing the SmoothStream project, errors are displayed as 'line too long' while running on the flake8 tool. |
| **2** | Account-management-system | The Account-management-system provides the output of various errors while analyzing using flake8. |
| **3** | Website-blocker | Analyzing the project using the flake8 provides the output errors like at least two spaces before inline comment', 'line too long', and no newline at end of the file. |
| **4** | Python-Media-Player | The project analysis through the flake8 tool results in the error outputs like 'missing whitespace around operator', 'trailing whitespace', etc. |
| **5** | sms\_slang\_translator | The flake8 tool project analysis provides the error results like expected 2 blank lines, found 1 and line too long. |
| **6** | atm-project | The too long line is identified as the error output result obtained using the flake8 tool. |
| **7** | Python-for-Hackers | The error outputs are obtained during the project using the flake8 tool with too many leading # for block content missing whitespace after ','. |
| **8** | Whatsapp\_Bot\_selenium | Most errors related to the missing whitespace around the operator are analyzed from the Python-Media-Player project. |
| **9** | Python-Speech-Recognition | Various error results are obtained as the output for the Code using the flake8 tool for the Python-Speech-Recognition project. |
| **10** | Whatsapp-live-cricket-bot | Several error outputs are obtained for the Whatsapp-live-cricket-bot project using the flake8 tool. Missing white space around operator is one of the most identified errors. |
| **11** | Email bomber | Line-to-long error is one of the most obtained error results for the email bomber project using the flake8 tool. |
| **12** | Are-you-Drowsy | Analyzing the Are-you-Drowsy project using the flake8 tool provides the output results of 'sys' imported but unused, missing whitespace around the operator, etc. |

# RESULTS AND FINDINGS

**Results:**

The empirical study on bad smells' effect on software architecture's maintainability and analysis using quality attributes have yielded interesting results. First, the study investigated the effects of bad smells on software maintainability. The results of the study suggest that bad smells hurt software maintainability.

The study assessed the perception of bad smells and their effect on maintainability. The most commonly mentioned bad smells were poor structure, improper use of patterns, lack of comments, and poor readability.

The results also showed that bad smells can be seen in the existing codebase. It is reported that the Code was difficult to read, lacked comments, and was poorly structured. The report is also made that it required spending more time debugging and fixing Code that had bad smells.

**Findings**

* The empirical study on bad smells' effect on software architecture's maintainability has revealed some interesting findings.
* The study results indicate a significant effect of bad smells on maintainability in software architecture and analysis using quality attributes.
* The study revealed that bad smells can lead to a decrease in the maintainability of the software architecture. In addition, bad smells can also increase the complexity of the software architecture, making it harder to understand and maintain.
* The study also found that bad smells can increase the number of code smells, which can further reduce the maintainability of the software architecture.
* Design smells are a type of bad smell present in the software's design and can lead to an increase in the complexity of the software, making it difficult to understand and maintain.
* The study also found that bad smells can increase the complexity of the software architecture, making it harder to understand and maintain.
* The findings of this study suggest that software architects should be aware of the effects of bad smells on maintainability and should strive to avoid bad smells in their design and implementation of the software architecture.
* The study also suggests that software architects should monitor for Code and design smells and take appropriate action when such smells are found.
* Finally, the study suggests that software architects should use quality attributes to evaluate their software architecture and take appropriate steps to improve the maintainability of the software architecture.

# CONCLUSION AND RECOMMENDATION

Software architecture is a critical component in the development of software applications. It provides a structure and framework for the software development process, which is essential for the maintainability of the software. Maintainability is an important factor in software engineering. It is essential to ensure that the software applications are designed and developed in a way that is easy to maintain and extend.

One of the main challenges in software architecture is dealing with bad smells. Bad smells are common problems that occur in software architecture and can significantly impact the overall maintainability of the software. In addition, bad smells can cause various issues, such as poor performance, scalability, and extensibility.

**Recommendation**

An empirical study should be conducted to evaluate the effect of bad smells on maintainability in software architecture. This study should focus on analyzing the impact of bad smells on maintainability are affected. Furthermore, the study should be conducted in a controlled environment and should include a variety of different software architectures.

The study should focus on the following aspects:

• Identifying the most common bad smells in software architecture.

• Analyzing the impact of the bad smells on maintainability.

• Investigating how bad smells impact different quality attributes.

• Examining how different approaches can mitigate the effects of bad smells.

The study should use a range of methods, focus on identifying the most common bad smells in software architecture and analyze the impact of these bad smells on the maintainability of the software.

The study should use various methods, including surveys, interviews, and case studies. The study should also focus on identifying the most common bad smells in software architecture and should analyze the impact of these bad smells on the maintainability of the software. Finally, the study should investigate the effectiveness of different approaches in mitigating the effects of bad smells.

**Conclusion**

The empirical study conducted on the effect of bad smells on maintainability in Software Architecture and Analysing using Quality Attributes has shown promising results. It has been found that when bad smells were considered in the software architecture, it resulted in higher levels of maintainability.

Moreover, the study showed that the presence of bad smells directly affected the quality attributes of the software. The quality attributes like maintainability are found to be significantly impacted by bad smells.

The findings of this study provide valuable insights into the effect of bad smells on the maintainability and quality attributes of software architecture.

It is essential for software architects and analysts to consider bad smells in their software architecture designs. The presence of bad smells can lead to a decrease in maintainability attributes. Hence, it is essential to identify and address these bad smells before the software architecture is implemented.

Overall, this empirical study has revealed the importance of considering bad smells in software architecture designs. It has been found that bad smells directly and significantly impact the software architecture's maintainability and quality attributes. Therefore, it is essential for software architects and analysts to consider bad smells while designing software architecture. It will ensure that the software architecture is of higher quality and easier to maintain.

# REFERENCES

|  |  |
| --- | --- |
| [1] | S. M. Olbrich, D. S. Cruzes, and M. S. Sjaberg, "Are all Code smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems," *IEEE International Conference on Software Maintenance (ICSM),* vol. 2, no. 4, pp. 1-10, 2010. |
| [2] | M. Tufano, "When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away)," *IEEE Transactions on Software Engineering,* vol. 43, no. 11, pp. 1063-1088, 2017. |
| [3] | M. Abbes and F. Khomh, "An empirical study of the impact of two anti-patterns, blob, and spaghetti code, on program comprehension," *15th European conference on Software maintenance and reengineering (CSMR),* vol. 1, no. 2, pp. 181-190, 2011. |
| [4] | A. Yamashita and L. Moonen, "Do Code smells reflect important maintainability aspects? " *In 28th IEEE international conference on software maintenance,* vol. 2, no. 4, pp. 306-315, 2012. |
| [5] | A. Yamashita and L. Moonen, "Exploring the impact of inter-smell relations on software maintainability:" *In 35th International conference on software engineering,* vol. 2, no. 4, pp. 682-691, 2013. |
| [6] | F. Palomba, G. Bavota and M. D. Penta, "On the diffuseness and the impact on the maintainability of code smells a large scale empirical investigation," *Empirical Software Engineering,* vol. 2, no. 6, pp. 1-34, 2017. |
| [7] | L. Aversano, U. Carpenito, and M. Iammarino, "An Empirical Study on the Evolution of Design Smells," *Information,* vol. 11, no. 1, pp. 1-21, 2020. |
| [8] | M. L. Siddiq, "An Empirical Study of Code Smells in Transformer-based Code Generation Techniques," *IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM),* 2022. |
| [9] | I. Macia, "On the Impact of Aspect-Oriented Code Smellson Architecture Modularity: An Exploratory Study," *Fifth Brazilian Symposium on Software Components, Architectures, and Reuse,* 2011. |
| [10] | B. v. Oort, "The Prevalence of Code Smells in Machine Learning projects," *arXiv:2103.04146v1,* 2021. |
| [11] | C. Zhifei, "Understanding metric-based detectable smells in Python software: A comparative study," *Information and Software Technology,* pp. 14-29, 2018. |
| [12] | F. Khomh, "An Exploratory Study of the Impact of Code Smells on Software Change-proneness," *16th Working Conference on Reverse Engineering,* 2009. |
| [13] | I. Deligiannis, "A controlled experiment investigation of an object-oriented design heuristic for maintainability," *The Journal of Systems and Software,* p. 129–143, 2004. |
| [14] | G. Bavota, "An empirical analysis of the distribution of unit test smells and their impact on software maintenance," *28th IEEE International Conference on Software Maintenance (ICSM),* 2012. |
| [15] | T. Hall, "Some Code Smells have a Significant but Small Effect on Faults," *ACM Trans. Softw. Eng. Methodol,* 2013. |