

IDENTIFICATION OF SOFTWARE QUALITY ATTRIBUTES FROM CODE DEFECT PREDICTION: A SYSTEMATIC LITERATURE REVIEW

Lukas RUMBUTIS, Asta SLOTKIENĖ , Birutė PLIUSKUVIENĖ 


Department of Information Systems, Faculty of Fundamental Sciences, Vilnius Gediminas Technical University, Vilnius, Lithuania

Article History:

- received 10 April 2024
- accepted 31 May 2024

Abstract. Identifying and understanding reasons for deriving software development defects is crucial for ensuring software product quality attributes such as maintainability. This paper presents a systematic literature review and the objective is to analyze the suggestions of other authors regarding software code defect prediction using machine learning, deep learning, or other artificial intelligence methods for the identification of software quality. The systemic literature review reveals that many analyzed papers considered multiple software code defects, but they were analyzed individually. However, more is needed to identify software quality attributes. The more profound analysis of code smells indicates the significance when considering multiple detected code smells and their interconnectedness; it helps to identify the software quality sub-attributes of maintainability.

Keywords: code smell, machine learning, software quality attribute, systematic literature review.

 Corresponding author. E-mail: asta.slotkiene@vilniustech.lt

1. Introduction

Code smells are an implementation and design defect of the software source code that has an essential effect on the efficiency of software development processes. For this issue, the developers catalog and describe code defects and code smells with suggestions for their solutions to maintain and evolve the application throughout its life cycle. Nowadays, programming environments and static code analyzers use manual and automatic methods to identify potential code smells. The results of Al Hilmi et al. (2023) research shows a subjective understanding and low agreement between code defect detectors, applicability to limited contexts and unbalanced datasets, limited developer knowledge, unclear priorities, and unclear thresholds. This determines the complexity of the process for detecting code smells in two aspects. First are the developers' subjective perceptions of potential code smell because these scents may not be harmful (Fowler, 2002). The second aspect that impacts the process is the need for more consensus in the detection process and the difficulty in setting appropriate detection thresholds. Di Nucci et al. (2018) research confirms that working experience influences the results of code defect detection, while developers use semi-automatic or automatic detectors, where developers can provide valuable information for refactoring judgments and enhancing confidence in smell detection. This demonstrates the need for more sophisticated approaches, which help to indicate code smells more objectively based on the data and experiences. In this paper, we analyze the proposed

methods of researchers, which suggest that machine learning could help decide to avoid the subjective identification of code smells for software quality evaluation. In this analysis, we are looking for the answer to this scientific question: *What code smells are predicting for software quality identification?* This paper presents a systematic literature (SLR), review to answer the questions raised. The paper is structured as follows. Section 2 introduces the background of software code defects and their relations with software quality attributes and sub-attributes. Section 3 presents the review methodology. Section 4 shows the results obtained from the SLR. Other sections discuss the results obtained in the paper, and finally, we present future works and conclusions.

2. Background of the relationship between software quality attributes and code defects

In the 1990s, Kent Beck described the importance of design quality in developing software and popularized the word code smell. This word grew into a universal term in coding when it was introduced in the book *Refactoring: Improving the Design of Existing Code* by Fowler et al. (1990). The initial list has grown over the years and is described more in detail by Fowler (2002). As defined by Fowler (2002), most of the code smells affect more than one external software quality attribute, such as maintainability, which consists of reusability, testability, analyzability, and modularity based on standard ISO/IEC 25010:2023 (International Organization for Standardization, 2023). Therefore, some quality attributes have more influence than others.

Nowadays, different approaches are suggested to detect code smells: manual or automatic detection methods. The types of automatic detectors are classified, according to the algorithms used, into search-based techniques (Kaur & Kaur, 2017) and metric-based techniques (Di Nucci et al., 2018). In contrast, metric-based techniques (Kaur et al., 2016) are classified into software development process and product metrics.

According to recent research (Paiva et al., 2017; Albuquerque et al., 2023), some limitations were noticed in code smell different types of detectors:

1. Code smells are subjective and dependent on interpretations of their definitions and experiences of developers;
2. Automatic detectors of code smell have different static code analysis rules, different measurement algorithms of metrics, and thresholds of these metrics. This impacts the number of detected code smells;
3. Code smell detectors can get false positives, not representing real problems, because these detectors depend on information related to the subject domain and the context. However, they need to pay more attention to the information related to the system design;
4. Detecting individual code smells does not fully identify the code quality because code smells are interrelated and influence each other. Their combinations determine software quality attributes such as maintainability.

Table 1. Code smells explanation and relations with software quality sub-attributes

Code Smell	Code Smell Explanation	Code Smell's Type	Impacts on Quality Sub-attributes
Long Method	The method is excessively lengthy, hindering comprehension.	Composite	Reusability, Testability, Analyzability, Modifiability
God Class	To maintain clarity and manageability, a class should ideally be focused on a single responsibility. Complexity arises when a class attempts to handle more than one.	Composite	Reusability, Testability, Analyzability, Modifiability, Modularity
Long Parameter List	Excessive parameters in methods violate SRP, reducing clarity and maintainability.	Simple	Analyzability, Testability
Feature Envy	The method relies heavily on another class's functionalities, suggesting poor design.	Composite	Reusability, Testability, Analyzability, Modifiability
Message Chains	Sequences exceeding three method calls can become clearer, hindering code flow and readability.	Composite	Testability, Analyzability, Modifiability
Data Class	The class primarily serves as a container for data, needing more substantial functionality.	Simple	Reusability
Duplicated Code	Identical code structures are found in multiple locations, increasing the maintenance burden.	Composite	Analyzability
Spaghetti Code	Inconsistent code structure, lacking adherence to established patterns and conventions, hinders understanding and maintenance.	Composite	Reusability, Testability, Analyzability, Modifiability
Functional Decomposition	Every function becomes a class, disregarding object-oriented principles.	Composite	Reusability, Testability, Analyzability, Modifiability, Modularity
Lazy Class	The class lacks sufficient value or functionality to justify its existence.	Simple	Reusability

Researchers apply different types of machine learning-based techniques to overcome the previous limitations because of their ability to detect the severity of code smell, construct code smell detection rules, and evaluate any new candidates by building the learning models. In this paper, we suggest reviewing and identifying current methods for code smell detection using machine-learning techniques. In this review, we decided to analyze 10 code smells important to software quality and impact quality attributes such as maintainability and their sub-attributes: reusability, testability, modifiability, and analyzability. Reusability describes the possibility of a part of code being used in multiple places to build other code. Analyzability and modifiability are quality sub-attributes that make it possible to understand what and where should be modified and whether it is possible to modify without affecting the quality of the software product. Testability indicates that it is possible to cover code with test cases and determine whether they pass.

In Table 1, we describe the code smells with explanations, which will be used in the review, and the type of code smells assigned to software quality sub-attributes of maintainability quality attributes based on ISO/IEC 25010:2023 standard.

3. Methodology of the systematic literature review

The review methodology was developed and executed according to the guidelines and hints provided by Kitchenham and Brereton (2013), Kitchenham et al. (2009) and Dybå and Dingsøyr (2008) respectively, and presented in Table 2 as a review protocol.

Table 2. Systematic literature review protocol

Question formulation
Question Focus: code-smells, smell-based defect, quality attributes, machine learning, detection system, defect prediction.
Main Scientific Question: "What code smells are predicted for software quality identification?" Research Question (RQ1): "What kind of code smells are predicted or detected?" Research Question (RQ2): "What can be used for code smell detection or prediction?" Research Question (RQ3): "How does code smell prediction impact software quality attribute maintainability?"
Keywords and Synonyms: code-smells, qualit*, machine learning, detect*, defect*
Effect: What are the different quality attributes considered for defect prediction in code?
Field/Scope/Confines: Scientific papers regarding machine learning and code smells.
Application: Computer Science (CS)
Sources Selection
Studies Language: English.
Search string: ("code* smell*" OR "code* defect*") AND ("qualit*") AND ("machine learning" OR "deep learning" OR "artificial intelligent*")
Sources list: Web of Science (WoS), https://apps.webofknowledge.com/
Studies Selection
Studies Inclusion Criteria (IC): IC1: Studies that analyze quality attributes to focus on code defect predictions using machine learning. IC2: Papers must be available to download.
Studies Exclusion Criteria (EC): EC1: Exclude papers that contain relevant keywords, but utilization of machine learning methods for code defect prediction is not the main topic of the paper. EC2: Exclude relevant sources that repeat ideas described in earlier works. EC3: Exclude papers with less than ten pages since such short papers can present only a general idea but not describe the overall approach. EC4: If there are several papers of the same authors with a similar abstract, i.e., one paper is an extension of another, the less extended (i.e., containing fewer pages) paper is excluded.
Studies Type Definition: Journal publications (research papers)
Information Extraction
Information Inclusion and Exclusion Criteria Definition: The extracted information from papers must contain an analysis of code quality attributes for code smell detection.
Synthesis of findings: The information extracted from the papers was tabulated and plotted to present basic information about the research process.

End of Table 2

Studies Selection

Procedures for Papers Selection (PPS):

PPS-1. Run the search strings at the selected source. A primary list of papers is obtained.

PPS-2. Extract the title, abstract, and keywords of papers for the primary set.

PPS-3. Evaluate a primary list of papers (the title, abstract, and keywords) according to inclusion and exclusion criteria. A secondary list of papers is obtained.

PPS-4. The eligibility step was applied to avoid papers that do not include experimentation to predict code defects using machine learning or deep learning.

For the analysis, scientific papers were chosen based on the search strategy, and the validity of the results was performed by applying the following selections: PPS-2, PPS-3, and PPS-4 (see Table 2). To begin, PPS-1 must perform search queries on academic databases. PPS-2 needs to analyze the complete content of articles comprehensively, but reviewing 2016 papers (after PPS-1) is time-intensive. PPS-2 is the list of papers by extracting the title, abstract, and keywords. Reading the abstract and the title of the papers introduces a threat because the abstract and the title allow for excluding non-relevant papers at first glance (Ivarsson & Gorschek, 2010). Using these fields and based on predetermined criteria, it is possible to determine if they meet the requirements for inclusion or exclusion. This process eliminates papers that should be sufficiently suitable for the general research idea and scope. PPS-3, we get a more concentrated list of analyzed papers. The final eligibility step filters out papers (PPS-4) that do not apply the application of machine learning or deep learning methods or algorithms for predicting code defects. The number of papers after each selection procedure is presented in Table 3.

Table 3. Number of papers for each PPS

Years	PPS-1	PPS-2	PPS-3	PPS-4
2014–2023	2016	119	77	21

The chronological distribution of the papers found in WoS is presented in Figure 1. They were published in the period from 2014 to 2023.

The chart in Figure 1 illustrates the research patterns in software engineering, with a particular emphasis on code smells. The blue columns, “PPS-2”, represent the cumulative number of papers obtained yearly from the Web of Science database. These papers were selected

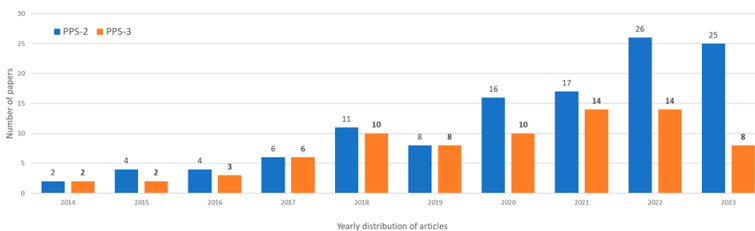


Figure 1. Chronological distribution between the primary set of papers (PPS-2) and after evaluation with IC and EC criteria (PPS-3)

based on the presence of keywords from the search string and the perceived relevance of their abstracts to the issue. This graph demonstrates a significant surge in publications, beginning with an insignificant figure in 2014 and escalating to 25 publications by 2023. The increased interest in the topic might be attributed to the rising significance of software code quality and the need for efficient defect prediction systems.

In contrast, the orange columns, "PPS-3", reflect the relevant papers after a more thorough evaluation. This approach included reading the papers to verify that quality standards were examined and debated. Including these papers in the Review Protocol indicates their direct significance to the subject. The trajectory for "PPS-3" exhibits slight variation, with the yearly publication count fluctuating between 8 and 14, ultimately reaching eight publications in 2023. This suggests that many scientific papers are being published, and only a few re-research studies have analyzed how the quality attributes and code defects are related.

From chronological distribution (see Figure 1) could indicate that although there is a significant overall interest in the topic, as shown by the "PPS-2" trend, the number of papers comprehensively examining software code defect prediction using machine learning is significantly smaller. It is only PPS-4 = 21 papers. This highlights the need for a deeper investigation and debate on the software code defect and software quality attributes.

4. Results of systematic literature review

The answer to the first research question (RQ1) is presented in Table 4 and Figure 2. It consists of twelve columns, ten presenting the code smells for software code defect prediction found in the papers. They are as follows: references (authors of papers), year (year of papers), and code smells (long method, god class, long parameter list, feature envy, message chains, data class, duplicated code, spaghetti code, functional decomposition, lazy class). All these code smell explanations are described in Table 1.

From Table 4, it can be noticed that 2022 had the highest number of papers on code smell prediction using machine learning, and our SRL has identified only one or two papers every year that performed research on the analyzed topic from 2014 to 2019. The analyzed code distribution by years is shown in Figure 2, where the size of the bubbles indicates the number of papers analyzing each code smell.

The colours of the bubble (see Figure 2) represent different code smells (each code smell description in Table 1). For example, the blue bubble indicates the "Long Method" code smell, the orange bubble represents the "God Class" code smell, and so on. The size of each colored bubble indicates the relative number of papers in that kind of code smells for a given year. More giant bubbles mean more scientific papers that apply AI to predict code smells, while smaller bubbles signify less research on the analyzed topic. The numbers within each coloured bubble show the count of papers per year.

The result of the review indicated that certain code defects, such as feature envy and data class, were continuously relevant from 2014 to 2023. These code defects are crucial in software engineering and are often scrutinized in literature. Conversely, the significance of the lengthy method and God class code smell has escalated since 2018, suggesting a heightened recognition of these concerns in recent years. In the recent year 2023, the code smells such

Table 4. The set of papers after the PPS-4 procedure (1 – code smell was analyzed in the paper, 0 – not analyzed).

References		Year	Long Method	God Class	Long Parameter List	Feature Envy	Message Chains	Data Class	Duplicated Code	Spaghetti Code	Functional Decomposition	Lazy Class
[1]	Sandouka and Aljamaan	2023	0	0	1	1	0	1	0	1	1	1
[2]	Luburić et al.		1	0	0	0	0	0	0	0	0	0
[3]	Albuquerque et al.		1	1	0	0	0	0	0	0	0	0
[4]	Dewangan et al.	2022	1	1	0	1	0	1	0	0	0	0
[5]	Oliveira et al.		1	1	0	1	1	1	0	0	0	0
[6]	Alkharabsheh et al.		0	1	0	0	0	0	0	0	0	0
[7]	Kovačević et al.		1	1	0	0	0	0	0	0	0	0
[8]	Liu et al.	2021	1	0	0	1	0	0	0	0	0	0
[9]	Boutaib et al.		0	0	1	1	0	1	0	1	1	1
[10]	Draz et al.		1	0	1	1	0	1	0	1	1	1
[11]	Agnihotri and Chug	2020	1	0	0	1	0	1	0	0	0	0
[12]	Guggulothu and Moiz		1	0	0	1	0	0	0	0	0	0
[13]	Mhawish and Gupta		1	1	0	1	0	1	0	0	0	0
[14]	Sousa et al.	2019	1	0	0	1	0	1	0	0	0	0
[14]	Singh et al.		0	1	0	0	0	0	0	0	0	0
[15]	Hozano et al.	2018	1	1	1	1	1	1	1	0	0	0
[16]	Sae-Lim at al.		0	1	0	0	1	1	0	0	0	0
[17]	Mansoor et al.	2017	0	0	0	1	0	1	0	1	1	0
[18]	Tufano et al.		0	0	0	0	0	0	0	1	1	0
[19]	Fontana et al.	2016	1	1	0	1	0	1	0	0	0	0
[20]	Sahin et al.	2014	0	0	1	1	0	1	0	1	1	1
[21]	Kessentini et al.		0	0	1	1	0	1	0	1	1	1

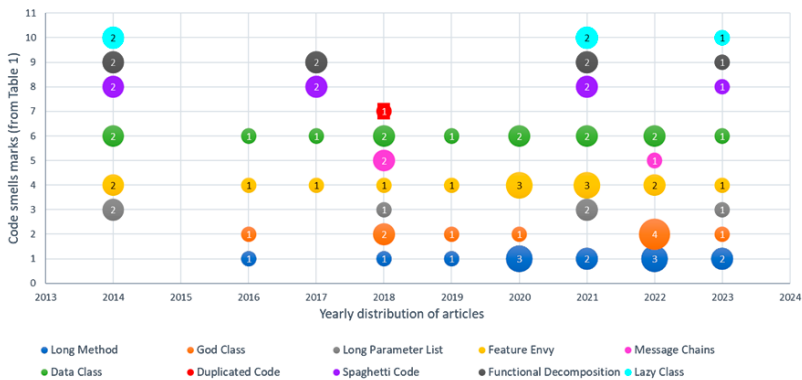


Figure 2. Found software code smells in the papers according to years

as message chains and duplicate code are not predicted using machine learning and are not analyzed in the papers. Nevertheless, it was noted that defects, such as message chains, duplicated code, and extensive parameter lists, should be explicitly acknowledged since they are often analyzed in combination with other problems.

The code smells could be identified, detected, or predicted using manual/semi-automated or automated ways of artificial intelligence. Manual/semi-automated way are used for annotation code chunks based on developers' experiences (Hozano et al., 2018; Singh et al., 2019; Albuquerque et al., 2023; Luburić et al., 2023) or identification code defects based on code quality metrics such as coupling, size of methods, cohesion, or complexity (Tufano et al., 2017; Sousa et al., 2019; Alkharabsheh et al., 2022). The automated way uses detection from measured metrics of code (complexity, coupling, lines of code, depth of inheritance tree, weighted method count, etc.) often using machine or deep learning methods: various kind of Neural Networks (Dewangan et al., 2022; Liu et al., 2021), Support Vectors Machines (Sandouka & Aljamaan, 2023; Kovačević et al., 2022; Oliveira et al., 2022; Mhawish & Gupta, 2020; Fontana et al., 2016) Random Forrest (Sandouka & Aljamaan, 2023; Kovačević et al., 2022; Fontana et al., 2016), Naive Bayes (Oliveira et al., 2022; Fontana et al., 2016), Decision Tree (Sandouka & Aljamaan, 2023; Oliveira et al., 2022; Mhawish & Gupta, 2020; Fontana et al., 2016). In this SRL, the detection or prediction ways were classified into four groups, presented in Table 5.

For the last research question (RQ3), we analyzed the software quality that could be identified from code smells. We are looking for an answer to this question through an analysis of each selected paper of PPS-4. What is the primary purpose of software quality for code defect annotation/detection/prediction? Many analyzed papers use software code quality metrics to identify code quality and anticipate that there may be a software bug that impacts software quality overall. Scientific papers published in 2019 highlight software quality attributes, such as maintainability, difficulty maintaining, and issues with repairing the detected bug. The authors (Luburić et al., 2023; Dewangan et al., 2022; Alkharabsheh et al., 2022; Kovačević et al., 2022; Liu et al., 2021; Boutaib et al., 2021; Sousa et al., 2019) noticed that left code defects strongly impacted software maintainability (see Table 6).

Table 5. Answer to RQ2: "What can be used for code smells detecting or predicting using AI?"

Ways of using artificial intelligence	References
Prediction using machine learning (ML)	Fontana et al. (2016), Mhawish and Gupta (2020), Guggulothu and Moiz (2020), Boutaib et al. (2021), Dewangan et al. (2022), Oliveira et al. (2022), Kovačević et al. (2022), Sandouka and Aljamaan (2023)
Prediction using deep learning	Mhawish and Gupta (2020), Liu et al. (2021), Dewangan et al. (2022).
Prediction using search-based approaches	Kessentini et al. (2014), Sahin et al. (2014), Tufano et al. (2017) (empirical investigation), Mansoor et al. (2017), Draz et al. (2021)
Other ML processes, such as annotation or classification code, smell by code quality metrics.	Sae-Lim et al. (2018), Hozano et al. (2018), Sousa et al. (2019), Singh et al. (2019), Alkharabsheh et al. (2022), Albuquerque et al. (2023), Luburić et al. (2023)

Table 6. Software quality aspects identification

Software quality assurance purpose	References
Software code quality metrics are used for code smell detection	Sandouka and Aljamaan (2023), Luburić et al. (2023), Dewangan et al. (2022), Oliveira et al. (2022), Alkharabsheh et al. (2022), Kovačević et al. (2022), Liu et al. (2021), Boutaib et al. (2021), Draz et al. (2021), Guggulothu and Moiz (2020), Mhawish and Gupta (2020), Sousa et al. (2019), Singh et al. (2019), Hozano et al. (2018), Tufano et al. (2017), Fontana et al. (2016)
Software quality improvement based on code smell detection	Sandouka and Aljamaan (2023), Albuquerque et al. (2023), Oliveira et al. (2022), Draz et al. (2021), Guggulothu and Moiz (2020), Mhawish and Gupta (2020), Singh et al. (2019), Hozano et al. (2018), Sae-Lim et al. (2018), Mansoor et al. (2017), Tufano et al. (2017), Fontana et al. (2016), Sahin et al. (2014), Kessentini et al. (2014)
Software quality attribute – maintainability improvement based on code smell detection	Luburić et al. (2023), Dewangan et al. (2022), Alkharabsheh et al. (2022), Kovačević et al. (2022), Liu et al. (2021), Boutaib et al. (2021), Sousa et al. (2019)

The answer to the main scientific question (MRQ) is more complicated because to detect or predict code smell separately and does not consider possible connections between them, the impacts on software quality identification should be more accurate. Code smells often occur together, and their collective influence on quality attributes and sub-attributes, such as testability, analyzability, reusability, modifiability, and modularity, may be substantial. Knowing the connections between various code smells helps the ability to evaluate their influence on defect prediction accurately. Identifying “message chains” alone may not be enough without considering additional concurrent concerns such as duplicated code. That way, the described relation between code smells and software quality attributes and sub-attributes allows us to more deeply understand that recognizing individual code smells does not help developers perceive the reason and more effectively refactor the place of code to improve software maintenance.

The answer to the main research question we found when we summarized by code smells relations with software quality sub-attributes. From the data on the prediction of code smell, we noticed that software quality attributes such as testability and analyzability were continuously the most focused from 2014 to 2023, with a significant rise in attention over the latter years. Reusability and modifiability were consistently emphasized during the analyzed period; while modularity gained more attention with time, it remained the least explored quality attribute by code smells god class and functional decomposition.

Therefore, future research should consider the interplay between code smells and their combined effect on external quality attributes. This would provide a comprehensive knowledge of how code smells impact software maintainability and predict potential failures for end-users. The limitation of this analysis is that we do not analyze the identified code smell impact via all levels of defect from bug to failure, from developers’ mistakes to end-users’ usability.

5. Discussion

The performed systematic literature review allows us to analyze what kind of code smells are predicted using machine learning. A deeper analysis of the nature of code smells shows that some code smells are composite from other code smells. For example, the long method might be a part of a god class. Researchers might study them together to understand the compound effect or propose comprehensive refactoring strategies that address multiple issues simultaneously. Common root causes, where multiple code smells may stem from the same underlying design or implementation issues. Analyzing them together, the root causes and how they manifest can be better understood. Also, refactoring code to address one smell might impact or introduce other smells.

Therefore, code smells should be analyzed in groups to achieve better results regarding defect prediction and code refactoring. Moreover, this review offers insights into how predicted code smells influence the software quality attribute – maintainability and its sub-attributes. However, it is essential to note that the study's conclusions are derived only from a survey of existing literature, which imposes limitations regarding the range and reliability of the studies covered. Future studies will be enhanced by conducting empirical studies investigating code defects' influence on internal and external quality attributes.

6. Future works

The systematic literature review has provided a thorough comprehension of code defect prediction and their relevancy for identifying software quality attributes – maintainability. Nevertheless, there is a significant need for this research area to continue developing. First, for relation identification between code smells, it is necessary to create an ontology or knowledge base. Through systematic documentation of code defects and their relations, researchers and developers may better understand how different categories of code smells impact different attributes of software quality. This knowledge base would be very beneficial in determining the impact of the different levels of software defect: from code to failure for end-users, from the requirement to software code, or the database structure. This knowledge and relations allow more effective quality assurance of the software development process and avoid various software defects, which influence internal and external software quality attributes.

7. Conclusions

The systemic literature review reveals that many analyzed papers considered more than one software code defect: the minimum two code smells for predicting software code using machine learning. The researchers analyzed for prediction code smells, often left via a programming process, but quality identification is not enough for the software. This indicates multiple justifications for analyzing a set of code smells rather than individual code smells to identify quality attributes. A deeper analysis of selected papers shows the limitations of analyzed papers: researchers should study code smells and predict software defects by interrelated code smells to understand the compound effect or propose comprehensive refactoring strategies that address multiple issues simultaneously and, at the same time, to improve several sub-attributes of software maintainability.

References

- Agnihotri, M., & Chug, A. (2020). A systematic literature survey of software metrics, code smells, and refactoring techniques. *Journal of Information Processing Systems*, 16(4), 915–934. <https://doi.org/10.3745/jips.04.0184>
- Albuquerque, D., Guimarães, E., Perkusich, M., Rique, T., Cunha, F., Almeida, H., & Perkusich, Â. (2023). On the assessment of interactive detection of code smells in practice: A controlled experiment. *IEEE Access*, 11, 84589–84606. <https://doi.org/10.1109/access.2023.3302260>
- Al Hilmi, M. A., Puspaningrum, A., Darsih, Siahaan, D. O., Samosir, H. S., & Rahma, A. S. (2023). Research trends, detection methods, practices, and challenges in Code Smell: SLR. *IEEE Access*, 11, 129536–129551. <https://doi.org/10.1109/access.2023.3334258>
- Alkharabsheh, K., Alawadi, S., Ignaim, K., Zanoon, N., Crespo, Y., Manso, E., & Taboada, J. A. F. (2022). Prioritization of god class design smell: A multi-criteria based approach. *Journal of King Saud University – Computer and Information Sciences*, 34(10), 9332–9342. <https://doi.org/10.1016/j.jksuci.2022.09.011>
- Boutaib, S., Bechikh, S., Palomba, F., Elarbi, M., Makhlouf, M., & Saïd, L. B. (2021). Code smell detection and identification in imbalanced environments. *Expert Systems with Applications*, 166, Article 114076. <https://doi.org/10.1016/j.eswa.2020.114076>
- Dewangan, S., Rao, R. S., Mishra, A., & Gupta, M. (2022). Code smell detection using ensemble machine learning algorithms. *Applied Sciences*, 12(20), Article 10321. <https://doi.org/10.3390/app122010321>
- Di Nucci, D., Palomba, F., Tamburri, D. A., Serebrenik, A., & De Lucia, A. (2018). Detecting code smells using machine learning techniques: Are we there yet? In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. <https://doi.org/10.1109/saner.2018.8330266>
- Draz, M. M., Salah, M., Abdulkader, S. N., & Gamal, M. (2021). Code smell detection using a whale optimization algorithm. *Computers, Materials & Continua*, 68(2), 1919–1935. <https://doi.org/10.32604/cmc.2021.015586>
- Dybå, T., & Dingsøy, T. (2008). Empirical studies of agile software development: A systematic review. *Information & Software Technology*, 50(9–10), 833–859. <https://doi.org/10.1016/j.infsof.2008.01.006>
- Fontana, F. A., Mäntylä, M., Zaroni, M., & Marino, A. (2016). Comparing and experimenting machine learning techniques for code smell detection. *Empirical Software Engineering*, 21(3), 1143–1191. <https://doi.org/10.1007/s10664-015-9378-4>
- Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (1990). *Refactoring: Improving the design of existing code*. Addison Wesley.
- Fowler, M. (2002). Refactoring: improving the design of existing code. In D. Wells & L. Williams (Eds.), *Lecture notes in computer science* (p. 256). Springer. https://doi.org/10.1007/3-540-45672-4_31
- Guggulothu, T., & Moiz, S. A. (2020). Code smell detection using a multi-label classification approach. *Software Quality Journal*, 28(3), 1063–1086. <https://doi.org/10.1007/s11219-020-09498-y>
- Hozano, M., Garcia, A., Fonseca, B., & De Barros Costa, E. (2018). Are you smelling it? Investigating how similar developers detect code smells. *Information & Software Technology*, 93, 130–146. <https://doi.org/10.1016/j.infsof.2017.09.002>
- International Organization for Standardization. (2023). *Systems and software engineering Systems and software Quality Requirements and Evaluation* (ISO Standard No. ISO/IEC 25010:2023). <https://www.iso.org/standard/78176.html>
- Ivarsson, M., & Gorschek, T. (2010). A method for evaluating rigor and industrial relevance of technology evaluations. *Empirical Software Engineering*, 16(3), 365–395. <https://doi.org/10.1007/s10664-010-9146-4>
- Kaur, K., & Kaur, P. (2017). Evaluation of sampling techniques in software fault prediction using metrics and code smells. In *International Conference on Advances in Computing, Communications, and Informatics*, (pp. 1377–1387). IEEE. <https://doi.org/10.1109/icacci.2017.8126033>
- Kaur, A., Kaur, K., & Jain, S. (2016). Predicting software change-proneness with code smells and class imbalance learning. *International Conference on Advances in Computing, Communications and Informatics*. IEEE. <https://doi.org/10.1109/icacci.2016.7732136>

- Kessentini, W., Kessentini, M., Sahraoui, H., Bechikh, S., & Ouni, A. (2014). A cooperative parallel Search-Based software engineering approach for Code-Smells detection. *IEEE Transactions on Software Engineering*, 40(9), 841–861. <https://doi.org/10.1109/tse.2014.2331057>
- Kitchenham, B., & Brereton, P. (2013). A systematic review of systematic review process research in software engineering. *Information & Software Technology*, 55(12), 2049–2075. <https://doi.org/10.1016/j.infsof.2013.07.010>
- Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering – A systematic literature review. *Information & Software Technology*, 51(1), 7–15. <https://doi.org/10.1016/j.infsof.2008.09.009>
- Kovačević, A., Slivka, J., Vidaković, D., Grujić, K., Luburić, N., Prokić, S., & Sladić, G. (2022). Automatic detection of Long Method and God Class code smells through neural source code embeddings. *Expert Systems with Applications*, 204, Article 117607. <https://doi.org/10.1016/j.eswa.2022.117607>
- Liu, H., Jin, J., Xu, Z., Zou, Y., Bu, Y., & Zhang, L. (2021). Deep learning-based code smell Detection. *IEEE Transactions on Software Engineering*, 1. <https://doi.org/10.1109/tse.2019.2936376>
- Luburić, N., Prokić, S., Grujić, K., Slivka, J., Kovačević, A., Sladić, G., & Vidaković, D. (2023). Towards a systematic approach to manual annotation of code smells. *Science of Computer Programming*, 230, Article 102999. <https://doi.org/10.36227/techrxiv.14159183>
- Mansoor, U., Kessentini, M., Maxim, B. R., & Deb, K. (2017). Multi-objective code-smells detection using good and bad design examples. *Software Quality Journal*, 25(2), 529–552. <https://doi.org/10.1007/s11219-016-9309-7>
- Mhawish, M. Y., & Gupta, M. (2020). Predicting code smells and analysis of predictions: using machine learning techniques and software metrics. *Journal of Computer Science and Technology*, 35(6), 1428–1445. <https://doi.org/10.1007/s11390-020-0323-7>
- Oliveira, D., Assunção, W. K. G., Garcia, A., Fonseca, B., & Ribeiro, M. (2022). Developers' perception matters machine learning to detect developer-sensitive smells. *Empirical Software Engineering*, 27(7), Article 195. <https://doi.org/10.1007/s10664-022-10234-2>
- Paiva, T., Damasceno, A., Figueiredo, E., & Sant'Anna, C. (2017). On the evaluation of code smells and detection tools. *Journal of Software Engineering Research and Development*, 5(1), Article 7. <https://doi.org/10.1186/s40411-017-0041-1>
- Sae-Lim, N., Hayashi, S., & Saeki, M. (2018). Context-based approach to prioritize code smells for refactoring. *Journal of Software: Evolution and Process*, 30(6), Article e1886. <https://doi.org/10.1002/smr.1886>
- Sahin, D., Kessentini, M., Bechikh, S., & Deb, K. (2014). Code-Smell detection as a bilevel problem. *ACM Transactions on Software Engineering and Methodology*, 24(1), 1–44. <https://doi.org/10.1145/2675067>
- Sandouka, R., & Aljamaan, H. (2023). Python code smells detection using conventional machine learning models. *PeerJ*, 9, Article e1370. <https://doi.org/10.7717/peerj-cs.1370>
- Singh, R., Bindal, A. K., & Kumar, A. (2019). A user feedback centric approach for detecting and mitigating god class code smell using frequent usage patterns. *Journal of Communications Software and Systems*, 15(3). <https://doi.org/10.24138/jcomss.v15i3.720>
- Sousa, B. L., Bigonha, M. a. S., & Ferreira, K. a. M. (2019). An exploratory study on cooccurrence of design patterns and bad smells using software metrics. *Software: Practice and Experience*, 49(7), 1079–1113. <https://doi.org/10.1002/spe.2697>
- Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., & Poshyvanyk, D. (2017). When and why your code starts to smell bad (and whether the smells go away). *IEEE Transactions on Software Engineering*, 43(11), 1063–1088. <https://doi.org/10.1109/tse.2017.2653105>