Identification of Code Properties that Support Code Smell Analysis

S. Prokić, N. Luburić, J. Slivka and A. Kovačević
Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia
{simona.prokic, nikola.luburic, slivkaje, kocha78}@uns.ac.rs

***Abstract* – Code smells are structures in code that imply potential maintainability problems and may negatively impact software quality. One of the critical challenges with code smells is that their definitions are often vague, difficult to comprehend and subjective, making them hard to reliably and consistently detect and analyze by humans and automated systems. Most existing code smell detection approaches rely heavily on human interpretation and are typically supported by structural code metrics. Unfortunately, many of these approaches are incomplete and do not cover a range of code properties that could indicate potential code smells.**

This paper analyzes code smell detection approaches to identify code properties used for code smell detection and analysis. Informed by our previous work and the literature, we define five code properties used by humans and automatic detectors to identify code smells. We demonstrate how various code properties can be mapped to the 22 code smells defined by Martin Fowler. The resulting catalog of properties can help software engineers and code maintainability researchers analyze code smells and build automated code smell detectors that examine properties beyond the traditional structural metrics.

Keywords – clean code; code smell; maintainability; code quality

I. INTRODUCTION

Code smells are structures in code that indicate maintainability issues in software design or implementation [1]. These structures may negatively impact various attributes of software quality, such as maintainability and evolvability [2]. The presence of code smells in software leads to increased maintenance costs due to the inability to understand existing code, adapt and create new features to address new requirements, and fix bugs [2][3].

Software engineering experts agree that removing harmful code smells is important for sustainable software development and high-quality code [2][4]. Identifying code smells is a challenging task for both humans and automated detection systems. Previous research has shown a significant disagreement among developers, as the definitions of code smell can vary depending on the context, developer experience, and intuition [5]. Code smell definitions tend to be ambiguous and hard to grasp, as they vary depending on the programming language, framework, and coding conventions used. It is worth noting that removing code smells can be time-consuming [6] and requires a deep understanding of the codebase and the system's architecture.

Most existing code smell detection approaches are solely based on structural metrics extracted from source code [7]. Structural metrics alone are insufficient for identifying all the smells specified by Fowler [1], as they cannot capture other aspects of the system that may indicate the presence of code smells. Historical information can offer a deeper understanding of the code's evolution and help identify patterns that may indicate the presence of certain code smells (e.g., *Shotgun Surgery*) [7]. Code property such as AST (Abstract Syntax Tree) adds more context about the system's structure and organization, enabling the identification of code smells that might not be immediately obvious from the source code [8]. The presence of specific code smells (e.g., *Feature Envy*) also needs to be evaluated by analyzing the relationship between the system's classes [9][10].

This paper investigates code smell detection approaches to identify code properties used for code smell detection and analysis. Based on our previous work and the literature review, we define five code properties used by humans and automatic detectors to identify code smells: text, AST, structural metrics, change history, and relationships. We demonstrate how different properties can be mapped to the 22 code smells defined by Martin Fowler [1], highlighting which properties are useful for each code smell analysis[1].

The resulting catalog of code properties can help software engineers and code maintainability researchers analyze code smells and build automated code smell detectors that examine properties beyond the traditional structural metrics. Software experts can use the catalog in the code smell annotation process to gain guidance on the crucial properties needed for each code smell analysis.

Section 2 presents related work for previously identified code properties. Section 3 outlines our research methodology. We offer our results in Section 4 and discuss results and limitations in Section 5. In Section 6, we make our conclusions.

This research was supported by the Science Fund of the Republic of Serbia, Grant No 6521051, AI-Clean CaDET.

[1] We use the term **analysis** to bring together activities such as detection, scoping (in terms of what code is affected) and determining severity of code smell.

II. RELATED WORK

Several earlier studies have addressed identifying various code properties for code smell analysis. To the best of our knowledge, none of these studies were concerned with mapping properties to Fowler's code smells (i.e., determining the usefulness of specific properties for smell analysis).

Walter et al. [15] identified six data sources considered useful for code smell analysis: programmer's intuition and experience, metric values, AST, history of changes made in code, dynamic behavior of code, and the existence of other smells. The authors briefly described each of these data sources but did not address the mapping of these sources to code smells. Instead of determining the usefulness of specific properties for certain smells, their study was focused on Large Class detection based on some of the listed sources.

Several recent review papers have listed data sources and approaches for code smell analysis. In [6], we found a list of metrics (primarily structural) used for the machine learning detection approaches for code smells. Authors of [6] focused exclusively on the machine learning approaches. Our goal was to identify properties useful for analyzing smells, regardless of whether performed manually or automatically.

Haque et al. [11] have presented manual and various automatic approaches used for code smell analysis (e.g., metrics-based, rule-based, history-based). Still, their aim was not to map the data used in these approaches to specific smells. Furthermore, machine learning and hybrid approaches for code smell analysis are listed in [12]. Although the mentioned approaches provide insight into the data used for code smell analysis (e.g., software metrics, change history), there is no mapping between specific code smells and data.

Authors of [13] present various tools used for code smell analysis and techniques they are based on (e.g., metrics-based, history-based). Additionally, the mapping of code smells and source code metrics used for their analysis in [13] can be useful for our paper.

We have previously conducted a systematic literature review (SLR) of the most recent studies to survey the existing machine learning approaches for smell detection [14]. Through the SLR, we have identified several data sources used in these approaches. These include structural metrics, historical features, AST, and natural language features. The data sources used for each code smell were valuable for mapping the code properties and smells in this study.

III. METHODOLOGY

Our study aims to determine code properties from the existing literature, regardless of whether humans or automated systems used them for code smell analysis. The goal is not to gather properties that produce the best smell detection results but to collect the properties frequently used, either individually or in combination. Following the identification of properties, we map these properties to Fowler's code smells [1], highlighting their use for analyzing each specific smell.

To accomplish this goal, we conducted a procedure that started with reading recently published survey papers on code smell analysis [6][11][12][13] and using data from our SLR [14]. In this step, we aimed to identify properties frequently used to analyze smells. While reviewing survey papers, we discovered a study [15] that was the foundation for creating an initial catalog of properties. They identified six distinct data sources useful for smell analysis. We made changes to the initial catalog taking into consideration the properties listed in other survey papers and our SLR. We describe these changes in the results section.

The result of the first step of our procedure was a catalog of five code properties, which we then mapped to 22 code smells specified by Fowler [1]. We used information from survey papers and the studies they cited to conduct the mapping. Using the existing literature as a reference, we identified specific properties that were examined when analyzing each smell.

Researchers do not give equal attention to each smell as they tend to focus on prevalent and impactful smells [16]. The literature we examined until this phase was insufficient to map certain smells and properties. We managed to map some of these smells to some of the properties in the previous step (most often *text* and *structural metrics*), but we needed more information for the remaining properties. We searched for additional studies that focused on analyzing these smells.

Despite extensive literature search, we were unable to find information for some of the less prioritized code smells and properties. Using the expert opinion research method [17], we completed the mapping of the properties to code smells rarely discussed in the literature. Relying on the expertise of two researchers who have been analyzing code smells for several years [18][19][20], we made predictions about whether properties could be useful for analyzing a specific smell.

The experts considered code smell definitions and property descriptions to make predictions. Each expert prediction was accompanied by an explanation of their reasoning, providing an opinion on the potential usefulness of the properties for code smell analysis. We provide a comprehensive description of these predictions and reasoning in the results of our research. Disagreements between experts were resolved in the discussion, and if there was no consensus, the opinion of a more experienced expert was considered more significant.

IV. RESULTS

This chapter presents the results of our research on code properties used for smell analysis by humans or automated systems.

A. Code properties

We used six distinct data sources useful for code smell analysis from [15] as a starting point for our catalog of code properties. These include *programmer's intuition and experience*, *metric values*, *abstract syntax tree (AST)*, *history of changes made in code*, *dynamic behavior of code*, and *existence of other smells.*

From these data sources, we retained *metric values*, referred to as **structural metrics** in our catalog of properties, **AST**, and *history of changes made in code* (referred to as **change history** in our catalog). The selection of these properties was based on data from survey papers on code smell analysis [6][11][12][13] and our SLR [14], which found that these properties were often used to analyze code smells.

Structural metrics provide a way to quantify the structural properties of the code, and most existing smell detection approaches are based only on structural metrics extracted from source code [7]. An extensive list of rules for automatic code smell detection based on the metrics and thresholds defined for each metric is presented in [23].

AST is a tree representation of the code structure where each node represents a construct in the code. AST is often used as an intermediate model for various model-based techniques for code smell analysis [23]. AST contains structural and semantic information that can be used for code smell analysis [8][24][25].

The code's **change history** provides insight into how the software evolved. This information can be used to identify trends and patterns in development, providing additional useful information for code smell analysis [26]. Several studies used change history to detect code smells [26][27][28], concluding that historical information can be helpful in the analysis of some code smells (e.g., *Divergent Change* and *Shotgun Surgery*).

We replaced *programmer's intuition and experience* with a code property named **text** (source code), bearing in mind that the manual smell analysis requires the source code where developers can rely on their intuition and experience. Analyzing the source code can reveal issues in the design or implementation that code smells imply. In [21] and [22], developers were asked how they analyze code smells. Both studies found that developers look at the source code to identify code smells and evaluate their severity.

Dynamic behavior of code as a data source is rarely discussed in the literature we reviewed while identifying properties [6][11][12][13][14]. Additionally, we did not analyze the co-existence of code smells because we focused primarily on the properties of the code being analyzed to determine the presence and impact of a code smell. Therefore, our catalog excludes *dynamic behavior of code* and *existence of other smells*.

Finally, we added the fifth code property to our catalog, representing **relationships** between system classes. The analysis of relationships is necessary to determine the presence of certain smells and has been used for code smell analysis [9][10][30][31]. Relationships can indicate highly coupled parts of the system, which can imply the presence of specific code smells (e.g., *Inappropriate Intimacy*) [29].

We use the term relationships for several relationships between system classes, such as inheritance. The important information is which classes are in the inheritance hierarchy and each class's direct descendants or ancestors. The next significant relationship represents method invocation to observe invoked methods, classes they belong to, and classes that invoked those methods. Similarly, it could be helpful to analyze the attributes that are accessed or modified, the classes to which they belong, and the classes that accessed or modified those attributes. One class could also have a member variable, local variable, method parameter, or return value of another class. These relationships also indicate highly coupled parts of the system and help analyze code smells. Some structural metrics provide insight into the relationships (e.g., CBO, DIT), but semantics are difficult to analyze through metrics.

B. Mapping code properties to code smells

After identifying the code properties, we map them to 22 code smells specified by Fowler [1], highlighting which properties are useful for analyzing each code smell. The resulting mapping is summarized in Table I where we mark "Yes" if the property was used to analyze a particular smell and list the references that discuss this analysis. If there are no references next to the value "Yes", our experts have reached these conclusions and we will describe their reasoning in more detail below. Code smells for which the value "No" is filled in Table I are those for which we found no evidence in the literature of specific properties being used for analysis, nor could we determine their usefulness through the expert opinion method [17].

Text and code smells: Several studies [21][22][32][33] conclude that developers perceive the existence and impact of some code smells by analyzing text (source code). For certain code smells (i.e., *Primitive Obsession, Alternative Classes with Different Interfaces, Divergent Change, Incomplete Library Class,* and *Comments*), we did not find similar evidence in the literature, so we used the expert opinion method to complete the mapping of the text code property. We concluded that all code smells could be analyzed through text as developers can look for the structures in the source code that suggest the possibility of refactoring (i.e., removal) of smells [1].

AST and code smells: AST has been used for code smell analysis [8][24][25][33][37][38]. *Alternative Classes with Different Interfaces* is one of the code smells not found in these studies. Since that code smell refers to classes with similar methods with different names [1][23], using the expert opinion method we concluded that similar patterns in classes could be observed through AST. Other smell not found in the literature is *Parallel Inheritance Hierarchies*, but we concluded that AST can also be used to observe hierarchies, providing information for this code smell.

Structural metrics and code smells: The metric-based approach is a widely used method among the various techniques for code smell detection [34]. Structural metrics can be used in combination with thresholds in detection rules [23], as well as in machine learning approaches [35][36].

Change history and code smells: Historical information has been used for code smell analysis in several studies [26][27][28][39][41].

TABLE I. CODE PROPERTIES MAPPED TO FOWLER'S CODE SMELLS, WHERE EACH VALUE (YES OR NO) INDICATES WHETHER THE SPECIFIC CODE PROPERTY IS USEFUL FOR ANALYZING AND DETECTING A PARTICULAR CODE SMELL

Code smell	**Code properties**				
	Text	***AST***	***Structural metrics***	***Change history***	***Relationships***
Long Method	Yes[21][22][32]	Yes[8][24][25]	Yes[23][34][35][36]	Yes[39]	No
Large Class	Yes[21][22][32]	Yes[8][25]	Yes[23][34][35][36]	Yes[26][27][39][41]	Yes[42][43]
Primitive Obsession	Yes	No	Yes[23]	Yes	No
Long Parameter List	Yes[21][22]	Yes[24][37]	Yes[23][34][35]	Yes	No
Data Clumps	Yes[33]	Yes[33][37]	Yes[34]	Yes	No
Switch Statements	Yes[32][33]	Yes[24][33][37]	Yes[23]	Yes	No
Temporary Field	Yes[32]	Yes[37]	Yes[23]	No	No
Refused Bequest	Yes[21][22][32]	Yes[24][37]	Yes[23][34][35]	Yes[39]	Yes
Alternative Classes with Different Interfaces	Yes	Yes	Yes[23]	No	No
Parallel Inheritance Hierarchies	Yes[32]	Yes	Yes[23]	Yes[26][27]	Yes
Divergent Change	Yes	No	Yes[23][34]	Yes[26][27][28]	Yes[31]
Shotgun Surgery	Yes[32]	No	Yes[23][34]	Yes[26][27][28][39]	Yes[42]
Lazy Class	Yes[21][22][32]	Yes[24][37]	Yes[23][34]	No	No
Data Class	Yes[32]	Yes[37]	Yes[23][36]	Yes[39][41]	Yes[43]
Duplicated Code	Yes[22][32]	Yes[38]	Yes[23][34]	Yes[28]	No
Speculative Generality	Yes[21][22][33]	Yes[33]	Yes[23][35]	No	No
Message Chains	Yes[32][33]	Yes[24][33]	Yes[23]	No	Yes
Middle Man	Yes[21][32][33]	Yes[33]	Yes[23][35]	No	Yes
Feature Envy	Yes[21][22][32]	Yes[8][24][25]	Yes[23][34][35][36]	Yes[26][27][39]	Yes
Inappropriate Intimacy	Yes[21][22][32]	No	Yes[35]	No	Yes[29]
Incomplete Library Class	Yes	No	No	No	No
Comments	Yes	No	Yes[23]	No	No

For some code smells (i.e., *Primitive Obsession, Long Parameter List,* and *Data Clumps*), we did not find similar research in the literature. It is pointed out in [40] that these smells seem likely to grow over time. Taking this into account and using expert opinion, we concluded that change history could provide useful information for analyzing these code smells.

We also used expert opinion on *Switch Statements* code smell, relying on the definition from [1] where it is said that the problem is essentially a duplication of code. Since *Duplicated Code* can be analyzed through change history [28], we concluded that *Switch Statements* could too.

Relationships and code smells: The visualization or information of relationships was used as an aid in the code smell analysis by the authors of several papers [29][31][42][43].

For smells not included in the mentioned papers (*Refused Bequest, Parallel Inheritance Hierarchies, Message Chains, Middle Man,* and *Feature Envy*), we used the expert opinion method to finish the mapping of relationships code property. The inheritance relationship [5] and related classes/methods should be considered when analyzing *Refused Bequest*. Examining the hierarchy in which the observed instance is located, we can determine if the hierarchy is incorrectly designed and whether the observed class is appropriately placed in it.

Several inheritance-related metrics are used in [23] for *Parallel Inheritance Hierarchies* code smell. These metrics provide numerical values that indicate certain characteristics of the hierarchy in which the observed class is located. Analyzing the relationships and related classes is essential to gain a deeper understanding of this code smell.

Message Chains code smell is reflected in a series of calls to other objects [5]. These relationships between various objects should be analyzed when detecting *Message Chains* and determining how to remove this smell. Class or method infected by *Middle Man* code smell has references to other classes as it is calling their methods [23][33]. We concluded that developers should pay attention to these relationships between system's classes when analyzing *Middle Man,* to determine whether the observed class just delegates its work to other classes or if it has a meaningful responsibility.

Methods containing *Feature Envy* code smell frequently access data from other classes [5][10]. These dependency relationships should be analyzed when assessing whether *Feature Envy* is present in code, which class is being accessed the most and how this smell should be removed.

V. DISCUSSION

This chapter will discuss the results, limitations and threats to the validity of our research. These limitations can also indicate areas for improvement and further research.

For five identified properties and 22 Fowler's code smells (a total of 110 combinations for mapping), we based 60% of the mapping on the existing literature. We completed the remaining 40% of the mapping based on the conclusions of two experts. To map text property and code smells, we used the expert opinion method in 22.7% of cases (five out of 22 smells had to be evaluated using this method). In the case of AST, 36.4% of mapping (eight out of 22 smells) was done using the expert opinion method, whereby the experts concluded that 27.3% of smells could not be analyzed by AST (six out of 22 smells). In the case of structural metrics, an expert opinion

was required for only one smell, and it was concluded that the metrics are not suitable for the analysis of that specific smell. From 59.1% of smells (13 out of 22), the experts concluded that 41% (nine out of 22) could not be analyzed through change history. Finally, the relationships property was the least represented in the literature, whereby experts made conclusions for 77.3% of smells (17 out of 22), and for as many as 54.5% (12 out of 22), it was concluded that this property would not be useful for the analysis.

Given the lack of information in the literature for mapping all properties and smells, we relied on two experts to complete the mapping. A greater number of experts in this mapping stage may have led to discovery of additional relevant insights.

Apart from the five properties we have identified, several other properties for code smell analysis can be found in the literature (i.e., semantic metrics [44], dynamic code behavior [15][45], and concern metrics [46]). Semantic metrics have been discussed more in the literature in the light of refactoring and various aspects of code quality. Still, we have not found relevant papers that analyze semantic metrics for the analysis of Fowler's code smells. Dynamic code behavior was not included because no recent studies used this property, while we found this property used only for *Data Class* [15] and *Refused Bequest* [45]. We excluded concern metrics because only several code smells were analyzed using those metrics (*Divergent Change, Shotgun Surgery*, *Large Class*) [46].

Furthermore, the existing tools for smell analysis often use properties that are easier to calculate and use (i.e., structural metrics are easier to calculate than semantic or concern metrics which require a deeper understanding of the system). Considering the literature and existing tools, it would be too challenging and time-consuming to identify all properties and map them to Fowler's code smells as part of this paper. It remains an open question of which less frequent code properties would be useful for smell analysis.

While we have mapped five properties to 22 Fowler's code smells, there are additional code smells in the literature that we have not covered [47][48][49][50][51]. With such a comprehensive list of code smells, we chose to concentrate on those specified by Fowler, which have a strong presence in the literature. Future researchers in this field may consider mapping code properties to other code smells to assess their usefulness for analysis.

We should point out that we did not conduct a SLR to identify the code properties for code smell analysis. A SLR would represent a more thorough approach for gathering information, but it would also be highly time-consuming. As indicated in the description of our methodology, we mainly relied on the data we read in recently published review papers and the papers they reference. However, we did rely on our SLR [14] of the existing machine learning approaches for code smell detection. As part of that review, we collected the data sources used in detection approaches, which we found useful for this study.

We believe that researchers should go beyond the traditional structural metrics when analyzing code smells and examine other code properties to achieve a more comprehensive understanding, and improved analysis of code smells. The resulting mapping of identified properties and code smells can guide software engineers and code maintainability researchers to build automated code smell detectors that examine properties beyond the traditional structural metrics. Additionally, our results can help researchers manually analyze code smells by examining properties highlighted for specific smells and investigate the impact of other properties sufficiently presented in the literature.

VI. CONCLUSION

The first objective of our study was to identify code properties used for code smell analysis by both human and automatic detectors. The second objective was to map the identified properties to code smells specified by Fowler.

Based on our previous work, the existing literature, and expert opinion, we have defined five code properties that are used for code smell analysis: text, AST, structural metrics, change history, and relationships. Through the mapping of these properties and 22 code smells, we emphasized the usefulness of properties for specific smell analysis. Resulting mapping could guide software engineers and researchers in their attempts to analyze code smells and build automated code smells detectors.

In addition to the five identified code properties, other properties (e.g., semantic metrics, concern metrics) have received less attention in the literature. They could be examined in the future to assess their impact and relevance for smell analysis. It should be noted that there are also other code smells besides Fowler's. These code properties and code smells have yet to be explored in the literature, and the research community should strive to identify all aspects essential for analyzing smells.

ACKNOWLEDGMENT

This research is supported by the Science Fund of the Republic of Serbia, Grant No 6521051, AI-Clean CaDET.

REFERENCES

[1] M. Fowler, et al., Refactoring: improving the design of existing code. Addison-Wesley, 1999.

[2] T. Sharma and D. Spinellis, "A survey on software smells," Journal of Systems and Software, 138, pp.158-173, 2018.

[3] A. Kaur, "A systematic literature review on empirical analysis of the relationship between code smells and software quality attributes," Archives of Computational Methods in Engineering, 27, pp.1267-1296, 2020.

[4] M. Fowler, Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 2018.

[5] M. Hozano, et al., "Are you smelling it? Investigating how similar developers detect code smells," Information and Software Technology, 93, pp.130-146, 2018.

[6] M. I. Azeem, et al., "Machine learning techniques for code smell detection: A systematic literature review and meta-analysis," Information and Software Technology, 108, pp.115-138, 2019.

[7] F. Palomba, et al., "Detecting bad smells in source code using change history information," 28th IEEE/ACM International Conference on Automated Software Engineering, pp.268-278, 2013.

[8] M. Hadj-Kacem and N. Bouassida, "Improving the Identification of Code Smells by Combining Structural and Semantic

Information," Neural Information Processing: 26th International Conference, pp.296-304, 2019.

[9] E. Murphy-Hill, et al., "Interactive ambient visualizations for soft advice," Information Visualization 12(2), pp.107-132, 2013.

[10] R. Oliveira, et al., "Collaborative identification of code smells: A multi-case study," IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track, pp.33-42, 2017.

[11] M. S. Haque, et al., "Causes, impacts, and detection approaches of code smell: a survey," Proceedings of the ACMSE 2018 Conference, pp.1-8, 2018.

[12] A. Kaur, et al., "A review on machine-learning based code smell detection techniques in object-oriented software system(s)," Recent Advances in Electrical & Electronic Engineering, 14(3), pp.290-303, 2021.

[13] G. Rasool and Z. Arshad, "A review of code smell mining techniques," Journal of Software: Evolution and Process 27(11), pp.867-895, 2015.

[14] K. G. Grujic, et al., "Machine Learning Approaches for Code Smell Detection: A Systematic Literature Review," Available at SSRN: https://ssrn.com/abstract=4299859, 2022. Unpublished.

[15] B. Walter and B. Pietrzak, "Multi-criteria Detection of Bad Smells in Code with UTA Method," Extreme Programming and Agile Processes in Software Engineering: 6th International Conference, Proceedings 6, pp.154-161, 2005.

[16] N. Sae-Lim, et al., "An Investigative Study on How Developers Filter and Prioritize Code Smell," IEICE TRANSACTIONS on Information and Systems, 101(7), pp.1733-1742, 2018.

[17] R. Wieringa, "Empirical research methods for technology validation: Scaling up to practice," Journal of systems and software, 95, pp.19-31, 2014.

[18] A. Kovacevic, et al., "Automatic detection of Long Method and God Class code smells through neural source code embeddings," Expert Systems with Applications, 204, p.117607, 2022.

[19] N. Luburic, et al., "Towards a systematic approach to manual annotation of code smells," Preprint available at TechRxiv: https://doi.org/10.36227/techrxiv.14159183.v3,2021.Unpublished.

[20] A. Kovacevic, et al., "Automatic detection of code smells using metrics and CodeT5embeddings: a case study in C#," Preprint available at TechRxiv: https://doi.org/10.36227/techrxiv.19682754.v2,2022.Unpublished.

[21] F. Palomba, et al., "Do they really smell bad? A study on developers' perception of bad code smells," IEEE International Conference on Software Maintenance and Evolution, pp.101-110, 2014.

[22] D. Taibi, et al., "How developers perceive smells in source code: A replicated study," Information and Software Technology, 92, pp.223-235, 2017.

[23] B. Bafandeh Mayvan, et al., "Bad smell detection using quality metrics and refactoring opportunities," Journal of Software: Evolution and Process, 32(8), p.e2255, 2020.

[24] S. Slinger, "Code smell detection in Eclipse," Delft University of Technology, 2005.

[25] M. Hadj-Kacem and N. Bouassida, "Deep representation learning for code smells detection using variational auto-encoder," International Joint Conference on Neural Networks, pp.1-8, 2019.

[26] F. Palomba, et al., "Detecting bad smells in source code using change history information," 28th IEEE/ACM International Conference on Automated Software Engineering, pp.268-278., 2013.

[27] F. Palomba, et al., "Mining Version Histories for Detecting Code Smells," IEEE Transactions on Software Engineering, 41(5), pp.462-489, 2015.

[28] S. Fu and B. Shen, "Code Bad Smell Detection through Evolutionary Data Mining," ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp.1-9, 2015.

[29] A. Jermakovics, et al., "Visualizing software evolution with lagrein," Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications, pp.749-750. 2008.

[30] K. Dhambri, et al., "Visual Detection of Design Anomalies," 12th European Conference on Software Maintenance and Reengineering, pp.279-283, 2008.

[31] D. Jiang, et al., "Distance metric based divergent change bad smell detection and refactoring scheme analysis," International Journal of Innovative Computing, Information and Control, 10(4), pp.1519-1531, 2014.

[32] A. Tahir, et al., "Can you tell me if it smells? A study on how developers discuss code smells and anti-patterns in Stack Overflow," Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018, pp.68-78. 2018.

[33] T. Hall, et al., "Some Code Smells Have a Significant but Small Effect on Faults," ACM Transactions on Software Engineering and Methodology, 23(4), pp.1-39, 2014.

[34] G. Lacerda, et al., "Code smells and refactoring: A tertiary systematic review of challenges and observations," Journal of Systems and Software, 167, p.110610, 2020.

[35] F. Pecorelli, et al., "A large empirical assessment of the role of data balancing in machine-learning-based code smell detection," Journal of Systems and Software, 169, p.110693, 2020.

[36] M. Y. Mhawish and M. Gupta, "Predicting Code Smells and Analysis of Predictions: Using Machine Learning Techniques and Software Metrics," Journal of Computer Science and Technology, 35, pp.1428-1445, 2020.

[37] S. Mekruksavanich, "Design Flaws Detection in Object-Oriented Software with Analytical Learning Method," International Journal of e-Education, e-Business, e-Management and e-Learning, 1(3), p.210, 2011.

[38] I. D. Baxter, et al., "Clone detection using abstract syntax trees," Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272), pp.368-377, 1998.

[39] S. A. Vidal, et al., "An approach to prioritize code smells for refactoring," Automated Software Engineering, 23, pp.501-532, 2016.

[40] M. V. Mäntylä and C. Lassenius, "Subjective evaluation of software evolvability using code smells: An empirical study," Empirical Software Engineering, 11, pp.395-431, 2006.

[41] D. Rapu, et al., "Using history information to improve design flaws detection," Eighth European Conference on Software Maintenance and Reengineering, pp.223-232, 2004.

[42] F. A. Fontana, et al., "Towards Assessing Software Architecture Quality by Exploiting Code Smell Relations," 2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics, pp.1-7, 2015.

[43] R. Wettel and M. Lanza, "Visually localizing design problems with disharmony maps," Proceedings of the 4th ACM Symposium on Software Visualization, pp.155-164. 2008.

[44] J. Pantiuchina, et al., "Why developers refactor source code: A mining-based study," ACM Transactions on Software Engineering and Methodology, 29(4), pp.1-30, 2020.

[45] E. Ligu, et al., "Identification of Refused Bequest Code Smells," 2013 IEEE International Conference on Software Maintenance, pp.392-395, 2013.

[46] J. Padilha, et al., "On the Effectiveness of Concern Metrics to Detect Code Smells: An Empirical Study," Advanced Information Systems Engineering: 26th International Conference, 26, pp.656-671, 2014.

[47] G. Suryanarayana, et al., "Refactoring for software design smells," ACM SIGSOFT Software Engineering Notes, 40, 2015.

[48] W. H. Brown, et al., AntiPatterns: refactoring software, architectures, and projects in crisis. John Wiley & Sons, 1998.

[49] A. J. Riel, Object-Oriented Design Heuristics. Addison-Wesley, 1996.

[50] M. Lippert and S. Roock, Refactoring in large software projects: performing complex restructurings successfully. John Wiley & Sons, 2006.

[51] T. Sharma, et al., "House of Cards: Code Smells in Open-Source C# Repositories," 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2017.