International Journal of Mathematical, Engineering and Management Sciences
Vol. 9, No. 3, 472-498, 2024
https://doi.org/10.33889/IJMEMS.2024.9.3.025

Ram Arti Publishers

Study of Code Smells: A Review and Research Agenda

Stuti Tandon
Amity Institute of Information Technology,
Amity University, Noida, Uttar Pradesh, India.
E-mail: stuti.mahendra85@gmail.com

Vijay Kumar
Department of Mathematics,
Amity Institute of Applied Sciences, Amity University, Noida, Uttar Pradesh, India.
Corresponding author: vijay_parashar@yahoo.com

V. B. Singh
School of Computer and Systems Sciences,
Jawaharlal Nehru University, New Delhi, India.
E-mail: vbsingh@mail.jnu.ac.in

(Received on August 29, 2023; Revised on December 4, 2023 & February 1, 2024 & February 25, 2024;
Accepted on March 6, 2024)

Abstract
Code Smells have been detected, predicted and studied by researchers from several perspectives. This literature review is conducted to understand tools and algorithms used to detect and analyze code smells to summarize research agenda. 114 studies have been selected from 2009 to 2022 to conduct this review. The studies are deeply analyzed under the categorization of machine learning and non-machine learning, which are found to be 25 and 89 respectively. The studies are analyzed to gain insight into algorithms, tools and limitations of the techniques. *Long Method*, *Feature Envy*, and *Duplicate Code* are reported to be the most popular smells. 38% of the studies focused their research on the enhancement of tools and methods. *Random Forest* and *JRip* algorithms are found to give the best results under machine learning techniques. We extended the previous studies on code smell detection tools, reporting a total 87 tools during the review. Java is found to be the dominant programming language during the study of smells.

Keywords- Code smells, Machine learning, Non- machine learning, Datasets, Detection tool.

1. Introduction

Software system is modified several times as per the requirement. During this modification, defects are introduced which do not alter the working but depreciate its quality. These software defects are termed as ***code smells***. The modified software contains defects in the design of software which leads to technical debt (Cunningham, 1992). Code smells arise due to bad programming practices. They are resultants of poor designing of software (Tufano et al., 2015). The term "Code smell" is frequently used by agile programmers. This term was first coined by Kent Beck while writing the book on Refactoring (Fowler and Beck, 1997). Code smells are different from software bugs as they are resultant of violations during programming practices which do not impact the code directly. ***Anti patterns*** are the patterns which always lead to negative impact whereas code smells are the hint of some "problem" but not the pattern. These design defects decrease the maintainability of the system thereby increasing the cost. As per International Organization for Standardization (ISO) standard 8402-1986 software quality is described as "the totality of features and characteristics of a product or service that bears its ability to satisfy stated or implied needs" (Karapetrovic and Willborn, 1998). Code smells not only leads to structural instability but also shorten the life cycle of the software (Sharma et al., 2017).

Refactoring is the process of altering the source code without adding any new function in it. This alteration in the design of the software improves readability, complexity, maintainability and extensibility. (Fowler and Beck, 1997) identified code smells which could be solved by refactoring. Various studies have been conducted for the detection of code smell. A lot of effort has been made by the researchers to develop the tools for detection of code smell and its methodology. Studies throw light on the development of tools for prediction of bad smell in order to avoid the wastage of human effort. Tools such as *Find Bugs, PMD (Programming Mistake Detector), Check style, SonarQube* identify the bad smell automatically. Software smells are studied from different perspectives. Sharma et al. (2016) classifies smells in three categories on the basis of the scope of their impact: - 1) *Implementation Smell:* These smells have limited scope and are confined to a class or a file, they have less local impact, they require less effort to refractor. 2) *Design Smell:* These smells impact a set of classes e.g.- insufficient modularization like God class, multi-faceted abstraction like divergent change, broken hierarchy like refused bequest. 3) *Architectural smell:* These smells span multiple components and have a system level impact. Figure 1 depicts the categorization of smells on the basis of scope of impact. One of the important components which gained researchers' interest is the usage of machine learning technique in the detection of bad smell.

Table 1. List of 23 code smells and 7 anti patterns in Java code.

Code No.	Code smells
1.	Duplicated Code
2.	Long Method
3.	Large Class
4.	Long Parameter List
5.	Divergent Change
6.	Shotgun Surgery
7.	Feature Envy
8.	Data Clumps
9.	Primitive Obsession
10.	Switch Statements
11.	Parallel Inheritance Hierarchies
12.	Lazy Class
13.	Speculative Generality
14.	Temporary Field
15.	Message Chains'
16.	Middle Man
17.	Inappropriate Intimacy
18.	Alternative Classes with Different Interfaces
19.	Incomplete Library Class
20.	Data Class
21.	Refused Parent Bequest
22.	Comments
23.	Dead Code
24.	Brain Class
25.	Brain Method
26.	Blob
27.	Sphagetti Code
28.	God Class
29.	God Method
30.	Swiss Army Knife

In this study we include the works published in well recognized journals and proceedings available in Scopus database. The review spans from 2009 to 2022. Quality assessment of the 157 selected studies was done and 114 studies were filtered on the basis of the quality scores obtained on the several judgment criteria. Review Questions are framed on the basis of challenges and problems faced by researchers.

Table 1 tabulates the code smells and anti patterns found in any Java code. The rest of the paper is organized as follows: The paper is divided into five sections. Section 2 discloses the related research on code smells, its categorizations and tools used to study it. Section 3 lays the methodology to conduct this review. Section 4 presents the discussions of the Research Questions (RQs) considered in this SLR. Section 5 presents the conclusion and the limitations of this SLR.

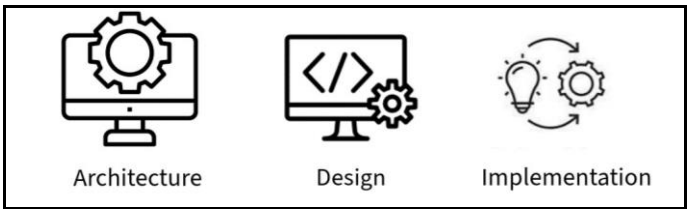

Figure 1. Smell classification based on scope of impact.

2. Related Research

Several studies have been conducted by the researchers to gain understanding on code smells detection and prediction (Fowler and Beck, 1997; Cunningham, 1992; Gupta et al., 2018; Aivaloglou and Hermans, 2016; Tandon et al., 2022; Kumar and Ram, 2021). Few other studies which draw our attention in retrieving answers to the research questions are: - Gupta et al. (2018) focused their study on the prediction of bad smell using mathematical models. Study by Aivaloglou and Hermans, (2016) was conducted on the block- based programming language Scratch. 250,000 projects were retrieved and code smells including large scripts and unmatched broadcast signals were found presented by authors. Gottschalk et al. (2012) proposes to detect and remove waste code using software re-engineering services.

Several review studies have been considered and analyzed from the perspectives of both machine learning and non – machine learning techniques. The review studies considered as the basis of the study conducted are: - the systematic literature review conducted by Zhang et al. (2011) on code smells covering papers from the span of 2000 to June 2009. Their SLR laid stress on: - a) The code smells attracting the most attention by researchers b) Why studies are carried on code smells? c) Methodologies used to study code smells d) Evidence indicating problems in the code. Azeem et al. (2019) conducted SLR considering papers published during 2005 -2017 and laid following points: - a) Usage of machine learning technique in detection of smells, b) Algorithms used by researchers for the detection of code smells, c) Performance of code smell prediction models. The literature review conducted by Rasool and Arshad (2015) focused their study on techniques and tools used for code smells from the source code of different software applications. A comparative study on the code mining techniques based on key characteristics was reported by Rasool and Arshad (2015). Most techniques were performed on different open-source systems and do not calculate the accuracy of the results due to absence of formal definition of code smells. The recommendations given by them are: - a) Detection techniques should be flexible enough to incorporate the integration of analysis methods and concepts. b) Metrics threshold values can either be based on expert knowledge or automatic. c) Detection techniques should be capable of detecting all 22 code smells identified by Fowler and Beck (1997). d) Standard benchmark desirable of evaluating results of code smells detection tools.

2.1 Contributions

This SLR has been broadly classified into two categories –

i) Studies deploying Machine Learning Techniques: Machine Learning techniques are being used currently and still have lot of scope for research due to - performance of classifiers over sample of total instances in the dataset, analysis of minimum training set to accurately detect smells and to analyze the number of smells detected by different classifiers over dataset (Fontana et al., 2016a).

ii) Studies deploying on-Machine Learning techniques: Several techniques under this category motivated the researchers to conduct their studies on code smells. Empirical studies, modeling, mathematical techniques, survey and several other apart from the machine learning techniques have been grouped under this category.

Gaps observed which motivated us to conduct this study adding value to the research. The contributions made by this study are as follows: -

- The difficulty in finding studies focusing its review both on Machine learning and non-machine learning techniques. This will not only reduce the effort but will also give comparative view for future research.
- This study reports the datasets used by the researchers in their study. This dataset will promote further research in this field.
- We have extended the study conducted by Fernandes et al. (2016) in identifying the tools for detecting code smells.
- Limitations of the strategies and performances of few studies considered for this review study have been reported in Table 2 to enhance further research.

Table 2. Comparative analysis of review studies.

	SCOPE	ADVANTAGES	LIMITATIONS
Proposed Study	2009-2022	• Aims and tools deployed for studying smells using ML and Non-ML techniques • Dataset of the studies are reported	• Metrics could be studied further deploying ML techniques
Zhang et al. (2011)	2000-2009	• Why study smell • Methodology • Evidences Indicating problems	• Generic code smell review
Azeem et al. (2019)	2005-2017	• Review used ML technique for detection of smell • Performance of models for smell detection	• Mainly focus on Machine learning techniques for detection of smells
Rasool and Arshad (2015)	1999-2015	• Techniques and tools used for smells	• Comparative study on code mining techniques

3. Research Methodology

To conduct literature review, its planning, processing and reporting are the key aspects which have been done during its study. The review conducted is focused on the studies published in all well recognized journals from the year 2009-2022. In this study the Theory, Methods and Context (TCM) framework proposed by Paul et al. (2023) is used. The steps considered in this study have been depicted pictorially in Figure 2. The first steps to frame the search strategy of the studies which are to be considered during this review. Then the selection criteria of the studies are framed to identify the relevant studies considered for this SLR. Step three focuses on the quality assessment of the studies. During this step a checklist is framed to evaluate the weight age of each study. Studies having a score less than 5 are rejected due to the lack of quality parameters which are considered for the review. In the data extraction step, several parameters are identified and tabulated. We have documented the important aspects as well as learning's related to smells. In data synthesis stage, asset of research questions is presented to analyze and conduct their view. TCM framework is considered as the review protocol to conduct unbiased and clear systematic

literature review. The subsequent subsections present the details of the review protocol used to conduct this SLR.

3.1 Search Strategy

The search string is formed by deriving the terms from the objectives of the study, their alternatives as well as synonyms are searched. A search strategy is formulated considering the processes laid in the textbooks by Mitchell (1997) and Witten and Frank (2002).

3.2 Study Selection

Selection of adequate and proper resources play a vital role in conducting a review. This literature review included the papers from the following searched electronic databases:

(i) SpringerLink (https://link.springer.com/).
(ii) IEEE Xplore (https://ieeexplore.ieee.org/Xplore/home.jsp).
(iii) ScienceDirect (https://www.sciencedirect.com/).
(iv) ACM Digital Library (https://dl.acm.org/).
(v) Wiley Online Library (https://onlinelibrary.wiley.com/).

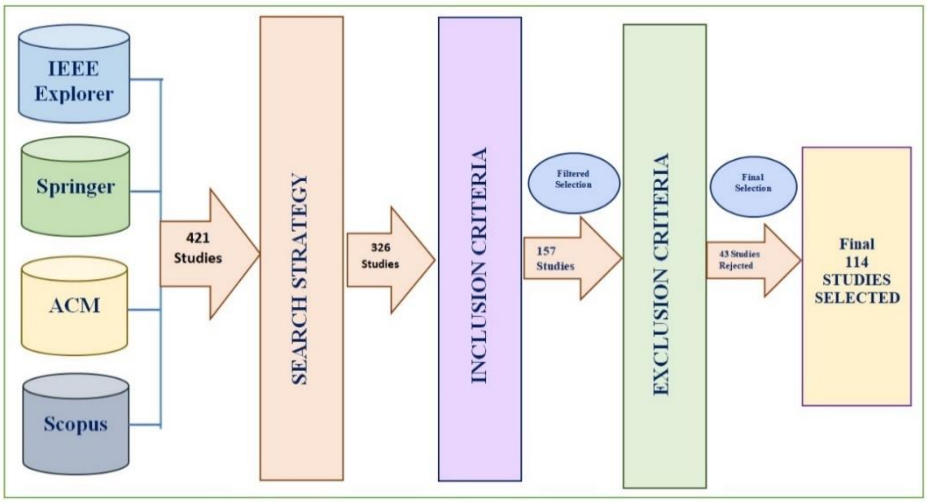

Figure 2. Selection process of studies.

After exhaustive searching of databases 421 studies were found, which were filtered to 157 studies on which inclusion criteria was applied. 43 studies were filtered out in exclusion criteria and finally 114 studies from the period of 2009-2022 were selected. Papers were identified using the following inclusion and exclusion criteria: -

Inclusion Criteria: We consider the following selection criteria for this literature review: -

- Papers including the following titles in their subject: - Code smell/ Bad smell, AI Techniques for Code smell detection, code smell detection, model for code smell detection, code smell using machine learning.
- Studies published in well-known journals, conferences, workshops and seminars in the timeline of 2009-2022.
- Papers indexed in web of science and Scopus are considered.

Exclusion Criteria: We excluded the papers which are: -

- Out of the scope of the above- mentioned knowledge areas.
- Not lying in the timeline from 2009- 2022.

Table 3. Studies published in journals.

JOURNALS	PAPER COUNT	IMPACT FACTOR (As per 2021)
Journal of software maintenance and evolution (Wiley)	3	1.864
Empirical Software Engineering	8	3.762
Applied Soft Computing	1	8.263
Information and software technology	4	3.862
Journal of systems and software	3	3.514
JOT (Journal of Object Technology)	1	1.21
Innovations system software engineering	1	1.44
Transactions on Software Engineering and methodology	10	3.685
Formal aspects of computing	1	1.170
Software Quality	1	1.813
ENTROPY	1	2.738
TOTAL	**34**	

Table 4. Studies published in conferences and workshops.

CONFERENCES and WORKSHOPS	PAPER COUNT
The International Conference on Evaluation and Assessment in Software Engineering (EASE)	4
The Mining Software Repositories (MSR)	1
Software Analysis, Evolution and Reengineering	2
Empirical software engineering and measurement (ESEM)	3
International Conference on Software Maintenance	2
WRT: Refactoring tools	3
Symposium on Search Based Software Engineering (SSBSE)	1
Quality of Information and Communications Technology, Proceedings of International Conference, QUATIC	2
Workshop on emerging trends in software metrics (WETSoM)	2
International Conference on Software Engineering (ICSE)	8
Managing Technical Debt (MTD)	3
Software architecture and metrics	1
Source code analysis and manipulation (SCAM)	4
International conference on software testing, verification and validation workshops	2
International Conference on Software Maintenance and Evolution	6
Proceedings of the XP2017 Scientific Workshops	1
International Conference on Mobile Software Engineering and Systems	1
International Conference on Program Comprehension	4
International Conference on Quality Software	1
Working Conference on Reverse Engineering	3
International conference on Aspect-oriented Software Development	3
International symposium on Software visualization	1
software engineering	2
International Conference on Automated Software Engineering (ASE)	1
data and software engineering	1
Working conference on reverse engineering (WCRE)	2
International conference on software analysis, evolution and reengineering (saner)	2
International Conference on Computer Science and Engineering (UBMK)	1
Computer Software and Applications Conference (COMPSAC)	1
TOTAL	**68**

The review focused on the studies published on code smells in leading software engineering journals and conferences to the best of our knowledge. Publication details of the studies selected for the review is tabulated in Table 3 (Studies published in Journals) and Table 4 (Studies published in conferences and workshops) respectively. Figure 3 depicts that 62% of the total papers are considered for this SLR and are

published in conferences and 38% of the total papers are published in journals.

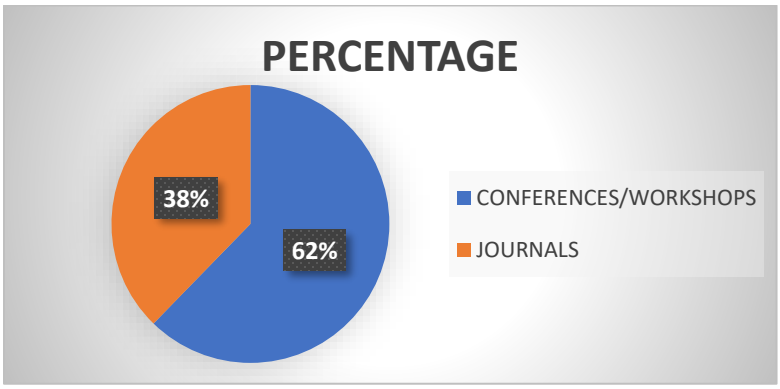

Figure 3. Percentage of sources of selected studies.

3.3 Identifying Search Terms

Relevant search terms are found by following the below listed five steps as recommended by Kitchenham et al. (2009):

I. The keywords in the titles and abstracts of the studies are searched.
II. To derive the search string, we find the synonyms of the extracted terms.
III. The Boolean operators OR operator is used in case of alternative spellings and synonyms; the AND operator is used for the concatenation of major terms.
IV. The obtained search string is integrated.
V. The research questions are used to derive the major terms.

Outcome of I: In the first step, we derived the keywords on the basis of the previous research on code smells.
Code smells, Technique, Software design defects, Machine learning algorithms, Tools, Programming language, Dataset.

Outcome of II: In this step the synonyms or the alternate spellings of all the major terms detected in step I are reported.

Code smell, Bad smell, Code Bad Smells, Smell, Software design defects, architectural smell, design smell, implementation smell; Techniques, learning, data mining, artificial intelligence, review, study.

Outcome of III: The Boolean operators OR and AND are used to concatenated the major terms retrieved during this process.

Smells (*code smells OR bad smells OR Code Bad Smells OR Software design Defects Or architectural smell OR Design Smell OR implementation smell) AND Study (prediction) AND Techniques (learning OR data mining OR artificial intelligence OR review OR study*).
Outcome of IV: The obtained search string is reported as:
"Smells" (*"code smells" OR "bad smells" OR "Code Bad Smells" OR "Software design Defects" Or "architectural smell" OR "Design Smell" OR "implementation smell") AND "Techniques" ("learning" OR "data mining" OR "artificial intelligence" OR "review" OR "study"*).

The resultant search string is considered and the studies undergone quality assessment.

3.4 Quality Assessment of Studies

Quality assessment is done to weigh each study to filter the relevant studies to conduct the review. A questionnaire is framed and tabulated in Table 5 to assess the relevance and quality of studies considered for this literature review. Every question in the questionnaire has three parameters "YES", "PARTLY", "NO". Here "YES" weighs 1 point, "NO" weighs 0 point and "PARTLY" carries 0.5 point. Every study is assessed for the quality and the score is summed. Studies having a quality score greater than 5(50% perfect score) are considered for this review to ensure that they contain maximum quality parameters. Studies with score less than 5 are thereby rejected. The obtained quality scores of the selected studies are presented in a table in Appendix A.

Table 5. Quality assessment questions.

QA#	QUESTIONS
QA1	Are the aims of research clearly specified?
QA2	Is the bad smell context described sufficiently?
QA3	Is the methodology for smells detection or prediction well defined?
QA4	Does the selected study contribute to the literature?
QA5	Are ML techniques specified in the selected study?
QA6	Are non-ML techniques specified in the selected study?
QA7	Any comparative analysis of ML techniques?
QA8	Any comparative analysis of non-ML techniques?
QA9	Are the results and findings clearly stated?
QA10	Are the limitations specified?

3.5 Data Extraction and Synthesis

The data extraction process finds the answers to the research questions considered for this SLR. The parameters mentioned in Table 6 are considered for the extraction process. The selected 114 research studies are considered for this process. The process not only extracts answers for the research questions but also finds limitations. The RQs are framed after brainstorming sessions by all the authors and professors.

Table 6. Parameters for paper extraction.

S. No.	PARAMETER
1.	Publication Year of the study
2.	Types of Code smell studied
3.	Aims of studies used by non-ML algorithms
4.	Algorithms used by ML Techniques
5.	Limitations of the study

3.6 Research Questions (RQs)

The aim of conducting this SLR is to find the answers to the questions which have been tabulated on the basis of the gaps and evidences obtained from the studies. Table 7 presents seven research questions which are discussed in this SLR. The relevance of code smells in software design is discussed as RQ1. The studies are analyzed to identify the most studied code smell in RQ2. RQ3 targets to find the most deployed technique to study code smells and its analysis. RQ4 focuses on identifying Machine Learning (ML) algorithms which gave the best results in the previous research works. Tools to detect code bad smells are found during the review of the available literature and reported under RQ5. RQ6 highlights the programming language which was preferred mostly by the researchers. RQ7 tabulates the datasets used by the researchers during their studies.

Table 7. Research questions.

S. No.	Research Question
RQ1	Relevance of Code smell in software design
RQ2	Identification of the most studied code smells
RQ3	To find the most deployed technique to study code smell and its analysis
RQ4	Recognizing the ML Algorithms deployed to give the best results
RQ5	Finding the studies on tools used in the detection of code smells
RQ6	Recognizing the programming languages used to study code smells
RQ7	To discern the datasets used by researchers during their study of smells

4. Discussion

This section discusses the research questions and its results considered for this systematic literature review. The seven research questions and their results are discussed hereby. The studies primarily used open-source software or public data sets. We provide directions for future research in the domain of software defects using the Theory, Methods and Context (TMC) framework.

4.1 Theory

Code smell has gained the most attention among researchers. There are possibilities in the future that the code smell which had least attention could be further investigated. The tools and the data with respect to the code smell which one wants to study can easily be accessible by going through the tabularized information. Table 8 presents the percentage and number of studies on each code smell and antipattern. 23 code smells and 7 anti-patterns are commonly found in any code of Java. Fontana et al. (2016b) proposed the taxonomies of smells, which are - *Bloaters* are those categories of smells which become unmanageable because of their size or volume; *Object Orientation Abusers* is a class of smells which deviate from actual practices of object-oriented principles; *Change Preventers* are the set of smells which make maintenance harder due to the nature of existing implementation. *Dispensable* are those smells which contain unnecessary code which should be removed. *Couplers* are the smells which contribute to excessive coupling between classes. *Encapsulators* are the smells which are increased at the cost of each other. Others is that category of smells which contain those units which are not fitted in the above-mentioned categories.

RQ1- Relevance of Code smell in software design

The potential breaches in the design of the code are termed smells. These technical debt affects the quality of the code adversely. Smells are easier and quicker to recognize than the ant patterns, which are more abstract and subtle. Causal analysis will help in studying the causes and effects of each category on the other. There may be cumulative effect of one or more categories which maybe more dangerous than the single category. After decades of research on this defect of software, researchers should now concentrate their studies on the cumulative effects of the categories of smells.

Research Agenda: We not only derive the agenda to study the categories of code smells but would also recommend for the causal analysis of categories of smells- *Bloaters, Object Orientation Abusers, Dispensables, Encapsulators, Change Preventers and Couplers*.

RQ2- Identification of the most studied code smells

During the review of previous studies, it was found that *Bloaters* is the most studied category of smell constituting of about 61.76% of studies. Anti patterns are the second highest studied category with 55.88% of total studies followed by *Dispensables* studied in 50.98% of studies. *Couplers and Object-Oriented Abusers* are analyzed by 27.45% and 14.7% of studies respectively. *Encapsulators, Change Preventers* and other categories are the least to be studied with 12.74%, 11.76% and 1.9% respectively.

Table 8. Count of publications on code smells.

S. No.	Code Smells	Study Count	Category
1.	Duplicated Code	15	Dispensables
2.	Long Method	28	Bloater
3.	Large Class	14	Bloaters
4.	Long Parameter List	14	Bloaters
5.	Divergent Change	0	Change Preventers
6.	Shotgun Surgery	12	Change Preventers
7.	Feature Envy	24	Couplers
8.	Data Clumps	6	Bloaters
9.	Primitive Obsession	1	Bloaters
10.	Switch Statements	4	Object Oriented Abusers
11.	Parallel Inheritance Hierarchies	0	Object Oriented Abusers
12.	Lazy Class	9	Dispensables
13.	Speculative Generality	7	Dispensables
14.	Temporary Field	0	Object Oriented Abusers
15.	Message Chains'	9	Encapsulators
16.	Middle Man	4	Encapsulators
17.	Inappropriate Intimacy	4	Couplers
18.	Alternative Classes with Different Interfaces	0	Object Oriented Abusers
19.	Incomplete Library Class	2	Others
20.	Data Class	18	Dispensables
21.	Refused Parent Bequest	11	Object Oriented Abusers
22.	Comments	0	Others
23.	Dead Code	3	Dispensables
24.	Brain Class	7	Antipattern
25.	Brain Method	6	Antipattern
26.	Blob	8	Antipattern
27.	Sphagetti Code	8	Antipattern
28.	God Class	23	Antipattern
29.	God Method	3	Antipattern
30.	Swiss Army Knife	2	Antipattern

Research Agenda: The least investigated categories need further investigation with respect to correlation with other categories. There is a possibility of the occurrence of the least category of smell due to presence of any of the prevailed ones. The hypothesis needs to be justified by further investigation and analysis of these design defects.

4.2 Methods and Context

An unbiased review is conducted with a motive to find the techniques widely used to study code smells. In the selected study span of the chosen studies to conduct the review, 25 studies are found to investigate the machine learning technique. 89 papers either used heuristic or metric strategies or are review studies. The distribution of studies deploying ML and non-ML techniques has been shown by the pie chart in Figure 4. Few studies compared the performance of machine learning and non-machine learning methods for metric-based code smell detection. Pecorelli et al. (2019) reported DÉCOR tool to have better performance than machine learning approaches but the precision is low thus making its usage less and limited. Pecorelli et al. (2019) reported that code smell detection using machine learning techniques is still used very less and is a problem which needs more research. Karađuzović et al. (2018b) analyzed the performance of machine learning methods for detection of God class, feature envy, long method and retrieved that the performance of reduced feature set was considerably better than the one achieved by Fontana et al. (2013a).

Literature study was conducted with the data collected from the years 2009-2022 and was sorted and segregated. The data was analyzed and found that 72.80% of the studies used non-ML techniques to study

code smell. 21.92% of studies used ML techniques and 5.26% are reported to conduct the review of code smell. An increasing trend in the application of machine learning algorithms for the study of code smell is noted with a maximum of 32% reported in 2018.

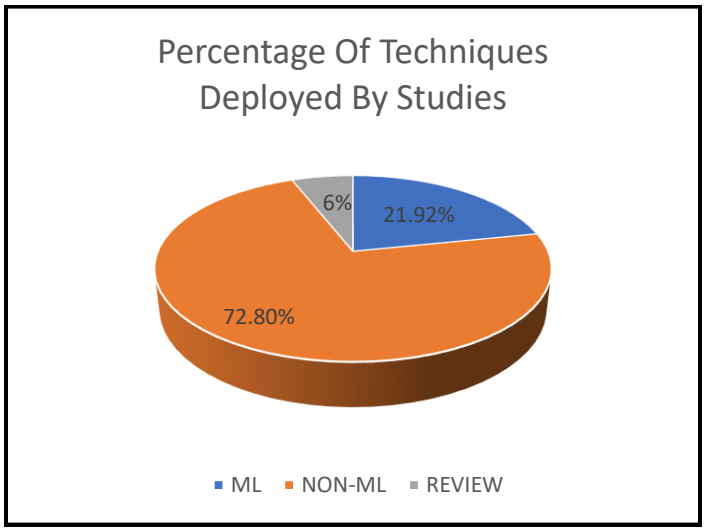

Figure 4. Percentage of techniques deployed by studies.

RQ3- To find the most deployed technique to study code smell and its analysis

It is pictorially observed that 72.80% of studies deployed non-ML techniques to study code smell either using heuristic and metric strategies in comparison to 21.92% of them using ML techniques. This shows there is scope for research in the detection of code smell using ML techniques considering its limitations. There is a reported rise in the trend of ML techniques used for code smell studies in recent years. Qualitative assessment using software metrics and further usage of algorithms of machine learning will be recommended for validated and precision of results. The focused aspects of code smell on which the researchers have worked were investigated. The analysis of studies which used non- AI techniques for their work is compiled and tabulated in Table 9 which depicts that:

- 38% of studies focused on tools and methods. Studies (Witten and Frank, 2002; Holschuh et al., 2009; Olbrich et al., 2009; Yamashita et al., 2009; Carneiro et al., 2010; Tempero et al., 2010; Fontana et al., 2011; Macia et al., 2011; Maneerat and Muenchaisri, 2011; Oliveto et al., 2011; Zhang et al., 2011; Fontana et al., 2012a; Peters and Zaidman, 2012; Fontana et al., 2013a; Palomba et al., 2013; Chatzigeorgiou and Manakos, 2014; Wohlin, 2014; Fontana et al., 2016b; Hermans and Aivaloglou, 2016; Palomba et al., 2016; Mansoor et al., 2017; Wang et al., 2018; Azeem et al., 2019; Guggulothu and Moiz, 2020; Pritam et al., 2019; Pritam et al., 2019; Gupta et al., 2021; Jain and Saha, 2021) are found to enhance tools and proposed methodology to identify code smells. This depicts that the primary goal during the study on code smell was the detection of smell and its methodology.
- 28% of studies are found to be either reports or case studies. Studies (Cunningham, 1992; Mitchell, 1997; Abebe et al., 2009; Yamashita et al., 2009; Chatzigeorgiou and Manakos, 2010; Counsell et al., 2010; Fontana et al., 2011; Meananeatra et al., 2011; Fontana et al., 2012b; Yamashita and Moonen, 2013c; Palomba et al., 2014a; Sahin et al., 2014; Fontana et al., 2015b; Rasool and Arshad, 2015; Sharma et al., 2016; Charalampidou et al., 2017; Palomba et al., 2018; Fontana et al., 2019) are found to improve the understanding on code smells.

- 21% of studies are grouped under empirical studies. Studies (Yamashita et al., 2009; Li and Thompson, 2010; Olbrich et al., 2010; Zazworka et al., 2011a; Hermans et al., 2012; Yamashita and Moonen, 2012; Hall et al., 2014; Fontana et al., 2015c; Szőke et al., 2015; Ahmed et al., 2017; Karađuzović-Hadžiabdić & Spahić, 2018b; Wang et al., 2018; Pecorelli et al., 2019; Kumar and Ram, 2021; Pigazzini et al., 2021) are found to analyze the data using existing techniques of code smell detection.
- 12% are found to target refactoring. Studies (Zhang et al., 2011; Yamashita and Moonen, 2012; Sjøberg et al., 2012; Fontana et al., 2013b; Hermans et al., 2016; Nagy and Cleve, 2017; Palomba et al., 2017; Di et al., 2018; Jain and Saha, 2021) enhanced the knowledge of refactoring.
- 1% of the study i.e. study by Azeem et al. (2019) is found to be a survey.

Research Agenda: By analyzing the above data, we can conclude that mostly the researchers aimed to enhance the tools and methodology for detecting code smells. Non-ML studies are found to be the most deployed technique by the researchers. In this technique 38% of studies enhance the tools and methods followed by 28% focusing on case studies and reports followed by 21% of empirical studies, 12% on refactoring and 1% on survey.

Table 9. Distribution of studies using non-ML techniques.

Aims of Studies	Count of Studies	Percentage
Case Study/Reports	23	28%
Tools/Methods/Experiments	32	38%
Refactor	10	12%
Empirical Study	18	21%
Survey	1	1%
Total	84	100%

RQ4- Recognizing the ML algorithms deployed to give best results

This RQ aims to report the algorithms which have given the best results in the previous studies. Maneerat and Muenchaisri (2011) compare the performances of the seven ML algorithms which used several statistical significance tests such as prediction accuracy, hypothesis test, sensitivity, specificity and predictive value of tests. Few algorithms are not suitable for the prediction of code smell because of their low prediction rate. The algorithms Naïve Bayes, VFI and J48 achieve less than 90% prediction average rate. Logistic regression, IB1, IBk and Random Forest achieves a higher overall prediction rate as they achieve 90% above average rate. Karađuzović-Hadžiabdić & Spahić (2018b) concluded that there is no best ML algorithm that accurately predicts all selected bad smells. ML algorithm should be considered based on the type of bad smell. Karađuzović-Hadžiabdić & Spahić (2018b) extended the work of Fontana et al. (2013a) by classifying code smells into 4 categories- no smell, non- severe smell, smell, severe smell and performed severity analysis by using six ML algorithms. The best result obtained for the data class by using the Random Forest ML method achieved 98.5% accuracy in performance. Results obtained for God class achieved 97.86% in performance using JRip. Fontana et al. (2016) report that the application of ML algorithms to detect code smells can provide high accuracy (>96%).

Research Agenda: It can be concluded that Random Forest and JRip are the most promising algorithms which gave better results in maximum studies than other algorithms. Decision Trees and Support Vector Machines are the most widely used algorithms for research. The performance of the reduced feature set is found to be better. Deep learning techniques are recommended to be used in studying this software defect to achieve more promising results.

RQ5- Finding the studies on tools used in the detection of code smells

Researchers find it difficult to choose which tools to use to conduct their study. This motivated us to

include this research question in the literature review. All the tools used by researchers in their previous studies on code smells are reported. A total of 87 tools are found which have been reported in Table 10. *JDeodorant* is found to be the most studied tool for detecting code smells followed by *iPlasma*, *infusion*, *DÉCOR*, *PMD*. These tools target to detect smells in the source code of software written in Java, C, C++, and C#. This SLR extended the study conducted by Fernandes et al. (2016) in identifying the tools for detecting code smell by reviewing the papers in the last decade.

Research Agenda: It is reported that a total of 87 tools for code smell detection have been reported during the literature review. Several detection tools for varied coding languages are reported. It is recommended to develop and use prediction-based tools to improve the prediction accuracy of the software. Prediction tools are recommended to avoid wastage of time and efforts on refactoring.

Table 10. List of tools to detect code smells.

S. No.	Tools	Paper Count	S. No.	Tools	Paper Count
1.	Absinthe	1	45.	Java Clone Detector	1
2.	Anti-pattern Scanner	1	46.	J Code Canine	1
3.	Refactoring Tools	5	47.	J Cosmo	2
4.	AutoMeD	1	48.	J deodorant	10
5.	Bad Smell Detection Tool (BSDT)	1	49.	J Smell	1
6.	Bad Smells Finder	1	50.	JS Nose	1
7.	bespoke tool	1	51.	J SpIRIT	2
8.	Bauhaus	1	52.	Kaleidoscope	1
9.	Bayesian Detection Expert (BDTEX)	1	53.	Kaur and Singh	1
10.	Borland together	1	54.	Keivanloo and Rilling	1
11.	Bug Forecast	1	55.	Komondoor and Horwitz	1
12.	CC Finder	1	56.	Matthew Munro	1
13.	Check style	4	57.	MOGP	1
14.	Clone Detector	1	58.	NiCad	2
15.	Clone Detective	1	59.	Nose Prints	1
16.	Clone Digger	1	60.	P-EA	1
17.	Coco Viz	1	61.	Paprika	1
18.	Code Nose	1	62.	PMD	6
19.	code smell detector	1	63.	PoSDef	1
20.	Code Smell Explorer	1	64.	PROblem DEtector O-O System (PRODEOOS)	1
21.	Code Vizard	3	65.	ProDeOOS	1
22.	Colligens	1	66.	Reclipse Tool Suite	1
23.	Concern ReCS	1	67.	Refactoring Browser	1
24.	Con QAT	1	68.	Ribeiro and Borba	1
25.	CP-Miner	1	69.	SCOOP	1
26.	Crespo	1	70.	Scorpio	1
27.	Crocodile	1	71.	SD Metrics	1
28.	DÉCOR	8	72.	Sextant	1
29.	DECKARD	1	73.	Smell checker	1
30.	DETEX	1	74.	Solid FX	1
31.	DuDe	1	75.	SonarQube	1
32.	Dup	1	76.	Stasys Peldzius	1
33.	Duploc	1	77.	Stench Blossom	4
34.	Evo Lens	1	78.	SVM Detect	1
35.	Excel Macros	1	79.	SY Make	2
36.	Fault Buster	1	80.	TACO	1
37.	Gendarme	1	81.	True Refactor	1
38.	HIST	3	82.	Understand	1
39.	In Code	1	83.	VCS-Analyzer	1
40.	Infusion	8	84.	Web Scent	1
41.	IntelliJ IDEA	1	85.	Weka Nose	1
42.	I Plasma	6	86.	Wrangler	2
43.	I SPARQL	1	87.	Xquery-based Analysis Framework (XAF)	1
44.	It's Your Code (IYC)	1			

RQ6-Recognizing the programming languages used to study code smells

By analyzing the 114 selected studies, we observe that about 73% of studies (83 in total) reported the programming language of the source code used to study smells. 31 studies either didn't declare the language of the source code considered or is a review study. 75 studies are found to use JAVA source code for studying code smells. 7 studies were considered C# to study smells. We found 6 studies using C++ source code and 4 studies using C code. 1 study each using R, Python and ABAP respectively is reported during the literature review. Figure 5 presents data of the distribution of programming languages considered by the researchers for the study of code smell. Detection tools are mainly proposed for the code of JAVA language thereby, giving it a rise to be the most dominant language to study code smells followed by C++ and C. Few researchers used the source code in C#, R, Python ABAP during their study. The ease of availability of the detection tool of any Java code has been reported by many studies. This certainly gives an opportunity to explore other programming languages. Charalampidou et al. (2017) stated the limitation of tools for identifying code smell as the reason to restrict their selection to Java projects.

Research Agenda: During this review, it is noted that few studies concentrated on multi language support for code smell detection. These findings report the opportunity of research in less explored programming languages.

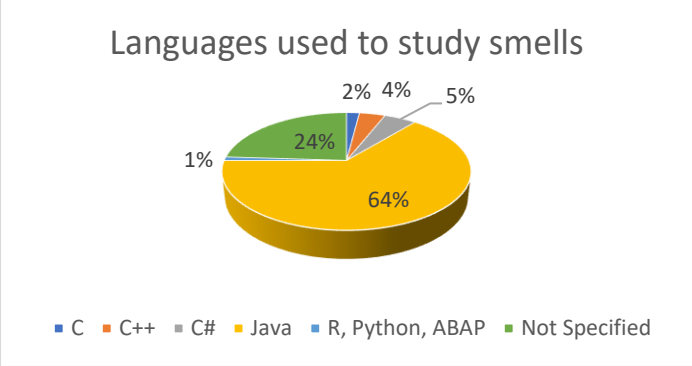

Figure 5. Languages used by researchers to study code smells.

RQ7-To discern the datasets used by researchers during their study of smells

After conducting review of 114 selected studies, we analyzed the datasets used by respective researcher. We believe this data tabulated in Table 11 will not only add novelty to the work but will also help future research with comprehensive information. Many researchers performed their experiments on Java based open-source systems or small internal projects. It was found that the dataset used by few researchers was either common or a part of the same data. This not only adds value to the subject data but also helps in setting a benchmark for evaluation of code smell detection tools and techniques. Fontana et al. (2011, 2013a, 2015a, 2015b, 2015c, 2019) Fontana and Spinelli (2011), Fontana and Zanoni (2011), Fontans et al. (2012) and Di Nucci et al. (2018) used the datasets of Tempero et al. (2010) in their works on code smells. These works by the authors are a great source of learning about code smells and were extended and used by other researchers. Karađuzović-Hadžiabdić and Spahić (2018a) and Szőke et al. (2015) conducted studies and used the dataset of Fontana et al. (2013b) in their study. Recent studies conducted by Jain and Saha (2021, 2022) are also reported to use the datasets of Fontana et al. (2013a). The study by Alazba and Aljamaan, (2021) used both the studies of Di Nucci et al. (2018) and Fontana et al.(2016a) to conduct their experiment. It is reported that the authors would like to replicate their study with new code smell datasets of different sizes. Eken et al. (2021) conducted their study by dividing the dataset into two

parts. The first part of the dataset is obtained by extending the dataset of Palomba et al. (2017) and the second part of datasetis obtained by including three additional open sources used in the study of Palomba et al. (2018). Sahin et al. (2014) collected their dataset using the open-source software in their study and also evaluated the proposal on the industrial system of Ford Motor Company.

Research Agenda: This article reports the list of tools which have been developed by several researchers. Prediction tools are not studied much and need to be explored as not much literature has been reported on it.

Table 11. Datasets used by researchers in their studies.

S. No.	STUDIES	DATASET/ SOFTWARE SYSTEM	REFERENCE
1.	Comparing and Experimenting Machine learning techniques for code smell detection	Dataset available at- http://essere.disco.unimib.it/reverse/MLCSD. html.	Fontana et al. (2016)
2.	Code smell detection as a bilevel problem	JFreeChart, GanttProject, ApacheAnt, Nutch, Log4J, Lucene, Xerces-J, and Rhino.	Sahin et al. (2014)
3.	Building Empirical Support for Automated code smell detection	Unnamed	Schumacher et al. (2010)
4.	Quantifying the effect of code smells on maintenance effort	Unnamed SYSTEM A, B, C, D	Sjøberg et al. (2012)
5.	Assessing the capability of code smells to explain maintenance problems: An empirical Study combining quantitative and qualitative data	Simula Research Laboratory's Software Engineering department web-based information system referred as System A, B, C, D	Yamashita (2014)
6.	Lexicon Bad Smells in Software	Alice, WinMerge	Abebe et al. (2009)
7.	Automatic detection of bad smells in code: An experimental assessment	6 versions of Gantt Project	Fontana et al. (2012b)
8.	Identifying Code Smells with Multiple Concern Views	5 versions of Open-source system Mobile Media	Carneiro et al. (2010)
9.	Assessing Code Smell Interest Probability: A Case Study	Spring, AndEngine open-source projects	Charalampidou et al. (2017)
10.	Investigating the Evolution of Bad Smells in Object-Oriented Code	Jflex, JFreeChart	Chatzigeorgiou and Manakos (2010)
11.	Investigating the evolution of code smells in object-oriented systems	Jflex, JFreeChart	Chatzigeorgiou and Manakos (2014)
12.	Is a Strategy for Code Smell Assessment Long Overdue?	Antlr, PDFBox, Velocity, Tyrant, HSQLDB	Counsell et al. (2010)
13.	Build Code Analysis with Symbolic Evaluation	SCST, LINN, GCC, MIN, LINS, FIRE, TS	Tamrawi et al. (2012)
14.	Exploring the Impact of Inter-smell Relations on Software Maintainability: An Empirical Study	Simula Research Laboratory's Software Engineering department web-based information system referred as System A, B, C, D	Yamashita and Counsell (2013)
15.	Investigating the Impact of Code Smells Debt on Quality Code Evaluation	Columba1.0, Drawswf1.2.9, Galleon2.3.0, C_jdbc 2.0.2, Heritrix 1.8.0, Struts 2.2.1, Ganttproject 2.0.9, Jhotdraw 7.5.1, Velocity 1.6.4, Antlr 3.2, Drjava 20100913-r5387, Pmd 4.2.5	Fontana et al. (2013a)
16.	Using bad smell-driven code refactorings in mobile applications to reduce battery usage	Fivestones, Sorter and Apps	Rodriguez et al. (2015)
17.	JSNOSE: Detecting JavaScript Code Smells	Webapplications: PeriodicTable , CollegeVis, ChessGame, Symbolistic, Tunnel, GhostBusters, TuduList, FractalViewer, PhotoGallery, TinySiteCMS, TinyMCE	Fard and Mesbah (2013)
18.	A Review-based Comparative Study of Bad Smell Detection Tools	Junit version 4, MobileMedia version 9	Fernandes et al. (2016)
19.	An experience report on using code smells detection tools	GanttProject v1.10.2	Fontana et al. (2011)
20.	Some Code Smells Have a Significant but Small Effect on Faults	API Recoder	Hall et al. (2014)
21.	An Empirical Study of the Performance Impacts of Android Code Smells	Open- source mobile apps: Sound Waves Podcast, Terminal Emulator	Hecht et al. (2016)

Table 11 continued…

22.	Detecting Code Smells in Spreadsheet Formulas	EUSES Corpus	Hermans et al. (2012)
23.	Detecting and refactoring code smells in spreadsheet formulas	EUSES Corpus	Hermans et al. (2016)
24.	A Cooperative Parallel Search-Based Software Engineering Approach for Code-Smells Detection	ApacheAnt , Xerces-J , GanttProject , Rhino , Log4J , Lucene , Nutch and JFreeChart	Kessentini et al. (2014)
25.	A Bayesian Approach for the Detection of Code and Design Smells	GanttProject v1.10.2, Xerces v2.7.0	Khomh et al. (2009a)
26.	An Exploratory Study of the Impact of Code Smells on Software Change-proneness	9 releases of Azureus and in 13 releases of Eclipse	Khomh et al. (2009b)
27.	Schedule of Bad Smell Detection and Resolution: A New Way to Save Effort	Java Source Metric and Thout Reader	Liu et al. (2011)
28.	Dynamic and Automatic Feedback-Based Threshold Adaptation for Code Smell Detection	AutoFlight, DirBuster, Java3DModeler, DavMail, PDF Split and Merge	Liu et al. (2015)
29.	Deep Learning Based Code Smell Detection	Junit, PMD, JExcelAPI, Areca, FreePlane, jEdit, Weka, AbdExtractor, Art of Illusion, Grinder	Liu et al. (2019)
30.	Are Automatically-Detected Code Anomalies Relevant to Architectural Modularity?	HealthWatcher,AspectualWatcher, MobileMedia, AspectualMedia, MIDAS	Macia et al. (2012)
31.	An Exploratory Study of Code Smells in Evolving Aspect-Oriented Systems	iBATIS, HealthWatcher,MobileMedia	Macia et al. (2011)
32.	From a domain analysis to the specification and detection of code and design smells	Xerces v2.7.0 and GanttProject v1.10.2	Moha et al. (2010)
33.	DECOR: A Method for the Specification and Detection of Code and Design Smells	ARGOUML, AZUREUS, GANTTPROJECT, LOG4J, LUCENE, NUTCH, PMD, QUICKUML, two versions of XERCES, and ECLIPSE.	Moha et al. (2009)
34.	Code Anomalies Flock Together	HealthWatcher, MobileMedia, Apache OODT, S1, S2, S3, S4(due to intellectual property constraints not specified)	Oizumi et al. (2016)
35.	The Evolution and Impact of Code Smells: A Case Study of Two Open-Source Systems	Apache Lucene and Apache Xerces 2 J	Olbrich et al. (2009)
36.	Are all Code Smells Harmful? A Study of God Classes and Brain Classes in the Evolution of three Open-Source Systems	Log4j, Apache Lucene, Apache Xerces	Olbrich et al. (2010)
37.	Improving Multi-Objective Code-Smells Correction Using Development History	Xerces-J, JFreeChart, GanttProject, ArtOfIllusion, JHotDraw	Ouni et al. (2015)
38.	Detecting Bad Smells in Source Code using Change History Information	Apache Ant3, Apache Tomcat4, jEdit5 and five projects belonging to the Android APIs	Palomba et al. (2013)
39.	Do they Really Smell Bad? A Study on Developers' Perception of Bad Code Smells	ArgoUML, Eclipse, jEdit	Palomba et al. (2014a)
40.	Mining Version Histories for Detecting Code Smells	Apache Ant, Apache Tomcat, jEdit, Android API, Apache Commons Lang, Apache Cassandra, Apache Commons Codec, Apache Derby, Eclipse Core, Apache James Mime4j, Google Guava, Aardvark, And Engine, Apache Commons IO, Apache Commons Logging, Mongo DB	Palomba et al. (2014b)
41.	A Textual-based Technique for Smell Detection	Apache Ant 1.8.0, aTunes 2.0.0, Eclipse Core 3.6.1, Apache Hive 0.9, Apache Ivy 2.1.0, Apache Lucene 3.6, JVLT 1.3.2, Apache Pig 0.8, Apache Qpid 0.18, Apache Xerces 2.3.0	Palomba et al. (2016)
42.	Evaluating the Lifespan of Code Smells using Software Repository Mining	CalDAV4j, Evolution Chamber, JDiveLog, jGnash, Saros, VLCJ, Vrapper (Base), Vrapper (Core branch)	Peters and Zaidman (2012)
43.	Clones: what is that smell?	APACHE httpd, NAUTILUS, EVOLUTION, GIMP	Rahman et al. (2012)
44.	A Proposal of Software Maintainability Model using Code Smell Measurement	BSIS 0.7.8, UniCenta POS 3.81, Traccar 3.1, JHotDraw7.0.6, JChart2D 3.2.2, Columba1.4	Wagey et al. (2015)
45.	Investigating the Impact of Design Debt on Software Quality	unnamed projects	Zazworka et al. (2011b)

Table 11 continued…

46.	Investigating Code Smell Co-occurrences using Association Rule Learning: A Replicated Study	ArgoUML, Ant, aTunes, Cassandra, Derby, Eclipse Core, Elastic Search, FreeMind, Hadoop, HSQLDB, Hbase, Hibernate, Hive, Incubating, Ivy, Lucene, Jedit, JHotDraw, JFreeChart, Jboss, JVlt, jSL, Karaf, Nutch, Pig, Qpid, Sax, Struts, Wicket, Xerces	Palomba et al. (2017)
47.	On the diffuseness and the impact on maintainability of code smells: a large-scale empirical investigation	ArgoUML, Ant, aTunes, Cassandra, Derby, Eclipse Core, Elastic Search, FreeMind, Hadoop, HSQLDB, Hbase, Hibernate, Hive, Incubating, Ivy, Lucene, JEdit, JHotDraw, JFreeChart, JBoss, JVlt, jSL, Karaf, Nutch, Pig, Qpid, Sax, Struts, Wicket, Xerces	Palomba et al. (2018)
48.	Code Smell Detection using Multilabel Classification Approach	datasets at: http://essere. disco.unimib.it/reverse/MLCSD.html	Guggulothu and Moiz (2020)
49.	Assessment of Code Smell for Predicting Class Change Proneness using Machine Learning	ACI, CheckStyle, FreePlane, Jkiwi, Joda, JStock, JText, LWJGL, ModBus, OpenGTS, OpenRocket, Quartz, Spring, SubSonic	Pritam et al. (2019)
50.	Comparing Heuristic and Machine Learning Approaches for Metric-Based Code Smell Detection	Ant, ArgoUML, Cassandra, Derby, Eclipse, Elastic Search, Hadoop, HSQLDB, Incubating, Nutch, Qpid, Wicket, Xerces	Pecorelli et al. (2019)
51.	Using Code Evolution Information to Improve the Quality of Labels in Code Smell Datasets	Tomcat, Jruby, netty	Wang et al. (2018)
52.	Multi-objective code-smells detection using good and bad design examples	ArgoUML v0.26, ArgoUML v0.3, Xerces v2.7, Ant-Apache v1.5, Ant-Apache v1.7.0, Gantt v1.10.2, Azureus v2.3.0.6	Mansoor et al. (2017)
53.	On investigating code smells correlations	GanttProject	Fontana and. Zanano (2011)
54.	Predicting Source Code Quality with Static Analysis and Machine Learning	PROMISE1: It is dataset from PROMISE workshop website named jm1, PROMISE2 jm1 with discrete values, STUDENT: dataset having student hand-ins basic programming course	Barstad et al. (2014)
55.	Experience Report: Evaluating the Effectiveness of Decision Trees for Detecting Code Smells	Eclipse 3.3.1, Mylyn 3.1.1, ArgoUML 0.26 and Rhino 1.6	Amorim et al. (2015)
56.	Software Code Smell Prediction Model Using Shannon, Rényi and Tsallis Entropies	Apache Abdera	Gupta et al. (2018)
57.	A Novel Four-Way Approach Designed with Ensemble Feature Selection for Code Smell Detection	Dr Java, EMMA, and FindBugs	Kaur and Kaur (2021)
58.	Tracing Bad Code Smells Behavior Using Machine Learning with Software Metrics	Android-Universal-ImageLoader-master, bigbluebutton-master, Bukkit-master, clojure-master, dropwizard-master, jfreechart-1.0.19, JHotDraw5.3-master, junit4-master, libgdx-master, metrics-4.1-development, netty-4.1, nokogiri-master, okhttp-master, presto-master	Gupta et al. (2021)
59.	Code smell detection and identification in imbalanced environments	GanttProject, ArgoUML, Xerces-J, JFreeChart, Ant-Apache, Azureus	Boutaib et al. (2021)
60.	Code smell detection by deep direct-learning and transfer-learning	C# and JAVA repositories	Sharma et al. (2021)
61.	A study on correlations between architectural smells and design patterns	Axion, cayenne, db-derby, hsqldb, squirrel-sql, hibernate, batik, displaytag, drawswf, itext, jasperreports, jext, marauroa, megamek, checkstyle, colt, drjava, eclipse SDK, jpf, nakedobjects, trove, informa, jena, jspwiki, jtopen, openjms, oscache, picocontainer, xmojo, quartz, QuickServer, sunflow, tapestry, ant, antlr, apache-maven, javacc, jparse, nekohtml, xalan, xerces, cobertura, emma, findbugs, fitjava, jmeter, junit, log4j, pmd, freecs, heritrix, james, jfreechart, jgraph, jgraphpad, jmoney, jsXe, pooka, proguard, webmail	Pigazzini et al. (2021)
62.	Are Multi-Language Design Smells Fault-Prone? An Empirical Study	Rocksdb, VLC-android, Realm,Conscrypt, Pljava, Javacpp, Zstd-jni, Jpype, Java-smt	Abidi et al. (2021)

5. Conclusion

The paper presents the literature review of code smells using both machine learning and non- machine learning techniques. The scope of this paper is limited to the research publications on code smells. The study span considered for this paper is 2009-2022. A total of 114 studies are selected to conduct this SLR. Our results indicate that Long Method (27%), Feature Envy (23%) and God Class (22%) are the most

popular smells among researchers. This may either mean that they are easy to study or are the most important ones. During our study we found 72.80% of the studies deployed non-ML techniques to study code smell and 21.92% of them used ML techniques. This reports non-ML techniques (heuristic, metric etc.) are largely studied by the researchers indicating that ML technique has scope for research. This review studies the non-ML techniques and reports 38% of the studies are conducted on the tools and methodologies followed by 28% on case studies and reports. It is found that *Random Forest* and *JRip* are the most promising algorithms reported and *Decision Trees* and *Support Vector Machines* are the most widely used algorithms for research. Java is found to be the dominant language used for studying code smells. After exhaustive study by several researchers, the datasets used in their study are reported and tabulated. Limitations of the studies inspires the researchers for future work. External Validity: This study targets research studies conducted during 2009-2022. The construct validity relative to the category choice of smell is present in the study due to the non-specific category of smell in this domain. A well-labelled dataset is obtained after collecting several studies which is filtered and it may suffer from internal validity despite of cross validating the filtered research papers. Lastly, conclusion validity may arise even though the databases are exhaustively searched, still there is a possibility that a qualifying study may be left. Random Forest and JRip are the most promising algorithms. This work can be further extended by evaluating the performance measures of all algorithms of machine learning. Causal analysis of *Bloaters*, *Object Orientation Abusers*, *Dispensables*, *Encapsulators*, *Change Preventers and Couplers* can be done to further enhance the work.

Conflict of Interest

The authors declare no conflict of interest.

Acknowledgments

This research did not receive any specific grant from funding agencies in the public, commercial or not-for-profit sectors. The authors would like to thank the editor- in- chief and anonymous reviewers for their comments that help improve the quality of this work.

Appendix A. Quality scores of the selected studies used in this SLR

S.No.	STUDY	QA1	QA2	QA3	QA4	QA5	QA6	QA7	QA8	QA9	Q A10	TOTAL
1.	Fontana et al. (2015c)	1	1	1	1	0	0.5	0	0	1	1	6.5
2.	Karađuzović-Hadžiabdić & Spahić (2018a)	1	1	1	1	0	1	0	1	1	1	8
3.	Szőke et al. (2015)	1	1	0.5	1	0	0.5	0	0.5	1	1	6.5
4.	Wang et al. (2018)	1	1	1	1	0	1	0	1	1	1	8
5.	Aivaloglou and Hermans (2016)	1	1	1	1	0	1	0	1	1	1	8
6.	Abebe et al. (2009)	1	1	1	1	0	1	0	1	1	1	8
7.	Fontana et al. (2016b)	1	1	1	1	0	1	0	1	1	1	8
8.	Sahin et al. (2014)	1	1	1	1	0	1	0	1	1	1	8
9.	Schumacher et al. (2010)	1	1	1	1	0	1	0	1	1	1	8
10.	Sjøberg et al. (2012)	1	1	1	1	0	0.5	0	1	1	1	7.5
11.	Yamashita (2014)	1	1	0	1	0	0.5	0	0	1	1	5.5
12.	Fontana et al. (2012a)	1	1	0.5	1	0	0.5	0	1	1	1	7
13.	Carneiro et al. (2010)	1	1	1	1	0	1	0	1	1	1	8
14.	Charalampidou et al. (2017)	1	1	1	1	0	0.5	0	1	1	1	7.5
15.	Chatzigeorgiou and Manakos (2010)	1	1	1	1	0	1	0	1	1	1	8
16.	Chatzigeorgiou and Manakos (2014)	1	1	1	1	0	1	0	1	1	1	8
17.	Counsell et al. (2010)	1	1	0.5	1	0	0.5	0	1	1	1	7
18.	Tamrawi et al. (2012)	1	1	0.5	1	0	1	0	0	1	1	6.5
19	Boussaa et al. (2013)	1	1	1	1	0	1	0	0.5	1	1	7.5

Appendix A continued…

20.	Rodriguez et al. (2015)	1	1	1	1	0	0.5	0	0	1	1	6.5
21.	Fard and Mesbah (2013)	1	1	1	1	0	1	0	0	1	1	7
22.	Fernandes et al. (2016)	1	1	1	1	0	1	0	0	1	1	7
23.	Fontana et al. (2011a)	1	1	1	1	0	1	0	0	1	1	7
24.	Hall et al. (2014)	1	1	1	1	0	1	0	0	1	1	7
25.	Hecht et al. (2016)	1	1	1	1	0	0.5	0	0	1	1	6.5
26.	Hermans et al. (2012)	1	1	1	1	0	0.5	0	0	1	1	6.5
27.	Hermans and Aivaloglou (2016)	1	1	1	1	0	1	0	0	1	1	7
28.	Kessentini et al. (2014)	1	1	1	1	0	1	0	0	1	1	7
29.	Khomh et al. (2009a)	1	1	1	1	0	1	0	1	1	1	8
30.	Liu et al. (2011)	1	1	1	1	0	1	0	0	1	1	7
31.	Khomh et al. (2009b)	1	1	1	1	0	1	0	0	1	1	7
32.	Liu et al. (2015)	1	1	1	1	0	1	0	0	1	0.5	6.5
33.	Liu et al. (2019)	1	1	1	1	0	1	0	1	1	1	8
34.	Macia et al. (2012)	1	1	1	1	0	1	0	1	1	1	8
35.	Macia et al. (2011)	1	1	1	1	0	1	0	1	1	1	8
36.	Moha et al. (2010)	1	1	1	1	0	1	0	0.5	1	1	7.5
37.	Moha et al. (2009)	1	1	1	1	1	1	1	0	1	1	9
38.	Oizumi et al. (2016)	1	1	1	1	0	1	0	0	1	1	7
39.	Olbrich et al. (2009)	1	1	1	1	0	1	0	0	1	1	7
40.	Olbrich et al. (2010)	1	1	1	1	0	1	0	1	1	1	8
41.	Ouni et al. (2015)	1	1	0	1	0	1	0	0	1	1	6
42.	Palomba et al. (2013)	1	1	1	1	1	0	1	0	1	1	8
43.	Palomba et al. (2014a)	1	1	1	1	0	1	0	1	1	1	8
44.	Palomba et al. (2016)	1	1	1	1	0	1	0	0.5	1	1	7.5
45.	Peters and Zaidman (2012)	1	1	1	1	0	1	0	0.5	1	1	7.5
46.	Rahman et al. (2012)	1	1	0.5	1	0	1	0	0.5	1	1	7
47.	Wagey et al. (2015)	1	1	0.5	1	0	1	0	1	1	1	7.5
48.	Mansoor et al. (2017)	1	1	1	1	0	1	0	1	1	1	8
49.	Fontana and Zanoni (2011c)	1	1	1	1	0	1	0	1	1	1	8
50.	Barstad et al. (2014)	1	1	1	1	0	1	0	1	1	1	8
51.	Amorim et al. (2015)	1	1	1	1	0	1	0	0	1	1	7
52.	Gupta et al. (2018)	1	1	0.5	1	0	1	0	0	1	1	6.5
53.	Kaur and Kaur (2021)	1	1	0.5	1	0	0.5	0	0	1	1	6
54.	Gupta et al. (2021)	1	1	1	1	1	0	1	0	1	1	8
55.	Boutaib et al. (2021)	1	1	1	1	1	0	0.5	0	1	1	7.5
56.	Sharma et al. (2021)	1	1	1	1	0	1	0	0	1	1	7
57.	Pigazzini et al. (2021)	1	1	1	1	0	1	0	0	1	1	7
58.	Abidi et al. (2021)	1	1	1	1	0	0.5	0	0	1	1	6.5
59.	Nagy and Cleve (2017)	1	1	1	1	0	1	0	1	1	1	8
60.	Moha et al. (2010)	1	1	1	1	0	1	0	0	1	1	7
61.	Pereira et al. (2022)	1	1	1	1	0	1	0	0	1	1	7
62.	Murphy-Hill and Black (2010)	1	1	1	1	0	1	0	1	1	1	8
63.	Mitchell (1997)	1	1	0	1	0	1	0	0	1	1	6
64.	Oizumi et al. (2016)	1	1	1	1	0	1	0	0	1	1	7
65.	Mansoor et al. (2017)	1	1	1	1	0	1	0	0	1	1	7
66.	Meananeatra et al. (2011)	1	1	1	1	0	1	0	1	1	1	8
67.	Kessentini et al. (2014)	1	1	1	1	0	1	0	0	1	1	7
68.	Moha et al. (2009)	1	1	1	1	0	1	0	0	1	1	7
69.	Li et al. (2010)	1	1	0.5	1	0	1	0	0	1	1	6.5
70.	Paul et al. (2023)	1	1	1	1	0	1	0	0	1	1	7
71.	Karapetrovic and Willborn (1998)	1	1	1	1	0	1	0	0	1	1	7
72.	Pigazzini et al. (2021)	1	1	0	1	0	1	0	0	1	1	6
73.	Karađuzović-Hadžiabdić and Spahić (2018a)	1	1	0	1	0	0	0	1	1	1	6
74.	Peters and Zaidman (2012)	1	1	0	1	0	0	0	1	1	1	6
75.	Jain and Saha (2022)	1	1	1	1	0	1	0	0	1	1	7
76.	Sharma et al. (2017)	1	1	0.5	1	0	1	0	0	1	1	6.5
77.	Olbrich et al. (2010)	1	1	1	1	0	1	0	0	1	1	7
78.	Jain and Saha. (2021)	1	1	0.5	1	0	1	0	0	1	1	6.5

Appendix A continued…

79.	Sharma et al. (2021)	1	1	1	1	0	1	0	0	1	1	7
80.	Holschuh et al. (2009)	1	1	0.5	1	0	1	0	0	1	1	6.5
81.	Pritam et al. (2019)	1	1	0	1	0	1	0	0	1	1	6
82.	Schumacher et al. (2010)	1	1	1	1	1	0	0	0	1	1	7
83.	Bryton et al. (2010)	1	1	1	1	0	0	0	0	1	1	6
84.	Maneerat and Muenchaisri (2011)	1	1	1	1	1	0	1	0	1	1	8
85.	Arcoverde et al. (2011)	1	1	0.5	1	0	0	0	0	1	1	5.5
86.	Sjøberg et al. (2012)	1	1	1	1	1	0	0	0	1	1	7
87.	Pecorelli et al. (2019)	1	1	1	1	1	0	1	0	1	1	8
88.	Avgeriou et al. (2016)	1	1	1	1	1	0	1	0	1	1	8
89.	Rahman et al. (2012)	1	1	1	1	1	1	1	1	1	1	10
90.	Hecht et al. (2016)	1	1	1	1	1	0	1	0	1	1	8
91.	Rasool and Arshad (2015)	1	1	1	1	1	0	1	0	1	1	8
92.	Hall et al. (2014)	1	1	1	1	1	0	0.5	0	1	1	7.5
93.	Sahin et al. (2014)	1	1	1	1	1	0	1	0	1	1	8
94.	Haendler et al. (2017)	1	1	1	1	1	0	0.5	0	1	1	7.5
95.	Palomba et al. (2014b)	1	1	1	1	0	0.5	0	0	1	1	6.5
96.	Guo et al. (2010)	1	1	1	1	0.5	0.5	0	0	1	1	7
97.	Rodriguez et al. (2015)	1	1	1	1	0	1	0	1	1	1	8
98.	Guggulothu and Moiz (2020)	1	1	0.5	1	0	0.5	0	0.5	1	1	6.5
99.	Oliveto et al. (2011)	1	1	0.5	1	0	0.5	0	0.5	1	1	6.5
100.	Gottschalk et al. (2012)	1	1	1	1	1	0	1	0	1	1	8
101.	Ouni et al. (2015)	1	1	1	1	1	0	1	0	1	1	8
102.	Fontana et al. (2015b)	1	1	1	1	0	1	0	1	1	1	8
103.	Sharma et al. (2016)	1	1	1	1	1	0.5	0.5	1	1	1	8
104.	Tandon et al. (2022)	1	1	1	1	1	1	0.5	1	1	0.5	9
105.	Szőke et al. (2015)	1	1	1	1	1	1	1	0.5	1	0.5	9
106.	Taibi et al. (2017)	1	1	1	1	1	1	0.5	1	1	0.5	9
107.	Tamrawi et al. (2012)	1	1	1	1	1	1	0.5	0.5	0.5	0.5	8
108.	Tempero et al. (2010)	1	1	1	1	1	1	0.5	0.5	0	0	7
109.	Vidal et al. (2016)	1	1	0.5	1	0	0.5	0	0.5	1	1	6.5
110.	Wagey et al. (2015)	1	1	1	1	1	1	0	0.5	0	0	6.5
111.	Wen at al. (2012)	1	1	1	1	1	1	0	0	0	1	7
112.	Witten et al. (2002)	1	1	1	1	1	1	0.5	0.5	0	0	7
113.	Zazworka et al. (2011a)	1	1	0.5	1	0	0.5	0	0.5	1	1	6.5
114.	Zhang et al. (2011)	1	1	1	1	1	1	0	0	0	1	7

References

Abebe, S.L., Haiduc, S., Tonella, P., & Marcus, A. (2009). Lexicon bad smells in software. In *2009 16th Working Conference on Reverse Engineering* (pp. 95-99). IEEE. Lille, France.

Abidi, M., Rahman, M.S., Openja, M., & Khomh, F. (2021). Are multi-language design smells fault-prone? An empirical study. *ACM Transactions on Software Engineering and Methodology*, *30*(3), 1-56.

Ahmed, I., Brindescu, C., Mannan, U.A., Jensen, C., & Sarma, A. (2017). An empirical examination of the relationship between code smells and merge conflicts. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 58-67). IEEE. Toronto, Canada.

Aivaloglou, E., & Hermans, F. (2016). How kids code and how we know: An exploratory study on the Scratch repository. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (pp. 53-61). Association for Computing Machinery. Melbourne, Australia.

Alazba, A., & Aljamaan, H. (2021). Code smell detection using feature selection and stacking ensemble: An empirical investigation. *Information and Software Technology*, *138*, 106648.

Amorim, L., Costa, E., Antunes, N., Fonseca, B., & Ribeiro, M. (2015). Experience report: Evaluating the effectiveness of decision trees for detecting code smells. In *2015 IEEE 26th International Symposium on Software Reliability Engineering* (pp. 261-269). IEEE. Gaithersbury, MD, USA.

Arcoverde, R., Garcia, A., & Figueiredo, E. (2011). Understanding the longevity of code smells: preliminary results of an explanatory survey. In *Proceedings of the 4th Workshop on Refactoring Tools* (pp. 33-36). Waikiki, Honolulu, USA.

Avgeriou, P., Kruchten, P., Ozkaya, I., & Seaman, C. (2016). Managing technical debt in software engineering (dagstuhl seminar 16162). In *Dagstuhl Reports* (Vol. 6, No. 4, pp. 110-138). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Azeem, M.I., Palomba, F., Shi, L., & Wang, Q. (2019). Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. *Information and Software Technology*, *108*, 115-138.

Barstad, V., Goodwin, M., & Gjøsæter, T. (2014). Predicting source code quality with static analysis and machine learning. In *Norsk IKT-konferanse for Forskning og Utdanning*. Fredrikstad, Norway.

Boussaa, M., Kessentini, W., Kessentini, M., Bechikh, S., & Ben Chikha, S. (2013). Competitive coevolutionary code-smells detection. In *Search Based Software Engineering: 5th International Symposium, SSBSE 2013, St. Petersburg, Russia, 2013. Proceedings 5* (pp. 50-65). Springer Berlin, Heidelberg.

Boutaib, S., Bechikh, S., Palomba, F., Elarbi, M., Makhlouf, M., & Said, L.B. (2021). Code smell detection and identification in imbalanced environments. *Expert Systems with Applications*, *166*, 114076.

Bryton, S., Abreu, F.B., & Monteiro, M. (2010). Reducing subjectivity in code smells detection: Experimenting with the long method. In *2010 Seventh International Conference on the Quality of Information and Communications Technology* (pp. 337-342). IEEE. Porto, Portugal.

Carneiro, G.D.F., Silva, M., Mara, L., Figueiredo, E., Sant'Anna, C., Garcia, A., & Mendonça, M. (2010). Identifying code smells with multiple concern views. In *2010 Brazilian Symposium on Software Engineering* (pp. 128-137). IEEE. Salvador, Brazil.

Charalampidou, S., Ampatzoglou, A., Chatzigeorgiou, A., & Avgeriou, P. (2017). Assessing code smell interest probability: a case study. In *Proceedings of the XP2017 Scientific Workshops* (pp. 1-8). Association for Computing Machinery. Cologne, Germany.

Chatzigeorgiou, A., & Manakos, A. (2010). Investigating the evolution of bad smells in object-oriented code. In *2010 Seventh International Conference on the Quality of Information and Communications Technology* (pp. 106-115). IEEE. Porto, Portugal.

Chatzigeorgiou, A., & Manakos, A. (2014). Investigating the evolution of code smells in object-oriented systems. *Innovations in Systems and Software Engineering*, *10*, 3-18.

Counsell, S., Hierons, R.M., Hamza, H., Black, S., & Durrand, M. (2010). Is a strategy for code smell assessment long overdue?. In *Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics* (pp. 32-38). Association for Computing Machinery. Cape Town, South Africa.

Cunningham, W. (1992). The WyCash portfolio management system. *ACM Sigplan Oops Messenger*, *4*(2), 29-30.

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)* (pp. 612-621). IEEE. Campobasso, Italy.

Eken, B., Palma, F., Ayşe, B., & Ayşe, T. (2021). An empirical study on the effect of community smells on bug prediction. *Software Quality Journal*, *29*, 159-194.

Fard, A.M., & Mesbah, A. (2013). Jsnose: Detecting JavaScript code smells. In *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation* (pp. 116-125). IEEE. Eindhoven, Netherlands.

Fernandes, E., Oliveira, J., Vale, G., Paiva, T., & Figueiredo, E. (2016). A review-based comparative study of bad smell detection tools. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering* (pp. 1-12). Association for Computing Machinery. Limerick, Ireland.

Fontana, F.A., Ferme, V., Zanoni, M., & Yamashita, A. (2015c). Automatic metric thresholds derivation for code smell detection. In *2015 IEEE/ACM 6th International Workshop on Emerging Trends in Software Metrics* (pp. 44-53). IEEE. Florence, Italy.

Fontana, F.A., & Spinelli, S. (2011). Impact of refactoring on quality code evaluation. In *Proceedings of the 4th Workshop on Refactoring Tools* (pp. 37-40). Association for Computing Machinery. Waikiki, USA.

Fontana, F.A., & Zanoni, M. (2011). On investigating code smells correlations. In *2011 IEEE fourth International Conference on Software testing, Verification and Validation Workshops* (pp. 474-475). IEEE. Berlin, Germany.

Fontana, F.A., Braione, P., & Zanoni, M. (2012b). Automatic detection of bad smells in code: An experimental assessment. *Journal of Object Technology*, *11*(2), 5-1.

Fontana, F.A., Dietrich, J., Walter, B., Yamashita, A., & Zanoni, M. (2016a). Antipattern and code smell false positives: Preliminary conceptualization and classification. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering* (Vol. 1, pp. 609-613). IEEE. Osaka, Japan.

Fontana, F.A., Mäntylä, M.V., Zanoni, M., & Marino, A. (2016b). Comparing and experimenting machine learning techniques for code smell detection. *Empirical Software Engineering*, *21*, 1143-1191.

Fontana, F.A., Ferme, V., & Spinelli, S. (2012a). Investigating the impact of code smells debt on quality code evaluation. In *2012 Third International Workshop on Managing Technical Debt* (pp. 15-22). IEEE. Zurich, Switzerland.

Fontana, F.A., Ferme, V., & Zanoni, M. (2015a). Towards assessing software architecture quality by exploiting code smell relations. In *2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics* (pp. 1-7). IEEE. Florence, Italy.

Fontana, F.A., Ferme, V., Marino, A., Walter, B., & Martenka, P. (2013b). Investigating the impact of code smells on system's quality: An empirical study on systems of different application domains. In *2013 IEEE International Conference on Software Maintenance* (pp. 260-269). IEEE. Eindhoven, Netherlands.

Fontana, F.A., Ferme, V., Zanoni, M., & Roveda, R. (2015b). Towards a prioritization of code debt: A code smell intensity index. In *2015 IEEE 7th International Workshop on Managing Technical Debt* (pp. 16-24). IEEE. Bremen, Germany.

Fontana, F.A., Lenarduzzi, V., Roveda, R., & Taibi, D. (2019). Are architectural smells independent from code smells? An empirical study. *Journal of Systems and Software*, *154*, 139-156.

Fontana, F.A., Mariani, E., Mornioli, A., Sormani, R., & Tonello, A. (2011). An experience report on using code smells detection tools. In *2011 IEEE fourth International Conference on Software Testing, Verification and Validation workshops* (pp. 450-457). IEEE. Berlin, Germany.

Fontana, F.A., Zanoni, M., Marino, A., & Mäntylä, M.V. (2013a). Code smell detection: Towards a machine learning-based approach. In *2013 IEEE International Conference on Software Maintenance* (pp. 396-399). IEEE. Eindhoven, Netherlands.

Fowler, M., & Beck, K. (1997). Refactoring: Improving the design of existing code. In *11th European Conference*. Jyväskylä, Finland.

Gottschalk, M., Josefiok, M., Jelschen, J., & Winter, A. (2012). Removing energy code smells with reengineering services. *INFORMATIK 2012*. Braunschweig, Germany.

Guggulothu, T., & Moiz, S.A. (2020). Code smell detection using multi-label classification approach. *Software Quality Journal*, *28*, 1063-1086.

Guo, Y., Seaman, C., Zazworka, N., & Shull, F. (2010). Domain-specific tailoring of code smells: An empirical study. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2* (pp. 167-170). Association for Computing Machinery. Cape Town, South Africa.

Gupta, A., Suri, B., & Lamba, L. (2021). Tracing bad code smells behavior using machine learning with software metrics. *Smart and Sustainable Intelligent Systems*, 245-257. https://doi.org/10.1002/9781119752134.ch18.

Gupta, A., Suri, B., Kumar, V., Misra, S., Blažauskas, T., & Damaševičius, R. (2018). Software code smell prediction model using Shannon, Rényi and Tsallis entropies. *Entropy*, *20*(5), 372.

Haendler, T., Sobernig, S., & Strembeck, M. (2017). Towards triaging code-smell candidates via runtime scenarios and method-call dependencies. In *Proceedings of the XP2017 Scientific Workshops* (pp. 1-9). Association for Computing Machinery. Cologne, Germany.

Hall, T., Zhang, M., Bowes, D., & Sun, Y. (2014). Some code smells have a significant but small effect on faults. *ACM Transactions on Software Engineering and Methodology*, *23*(4), 1-39.

Hecht, G., Moha, N., & Rouvoy, R. (2016). An empirical study of the performance impacts of android code smells. In *Proceedings of the International Conference on Mobile Software Engineering and Systems* (pp. 59-69). Association for Computing Machinery. Austin, USA.

Hermans, F., & Aivaloglou, E. (2016). Do code smells hamper novice programming? A controlled experiment on Scratch programs. In *2016 IEEE 24th International Conference on Program Comprehension* (pp. 1-10). IEEE. Austin, USA.

Hermans, F., Pinzger, M., & Van Deursen, A. (2012). Detecting code smells in spreadsheet formulas. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)* (pp. 409-418). IEEE. Trento, Italy.

Holschuh, T., Pauser, M., Herzig, K., Zimmermann, T., Premraj, R., & Zeller, A. (2009). Predicting defects in SAP Java code: An experience report. In *2009 31st International Conference on Software Engineering-Companion Volume* (pp. 172-181). IEEE. Vancouver, BC, Canada.

Jain, S., & Saha, A. (2021). Improving Performance by Genetically Optimizing Support Vector Machine to Detect Code Smells. In *Proceedings of the International Conference on Smart Data Intelligence*. Tamil Nadu, India.

Jain, S., & Saha, A. (2022). Rank-based univariate feature selection methods on machine learning classifiers for code smell detection. *Evolutionary Intelligence*, *15*(1), 609-638.

Karaduzović-Hadžiabdić, K., & Spahić, R. (2018a). Comparison of machine learning methods for code smell detection using reduced features. In *2018 3rd International Conference on Computer Science and Engineering* (pp. 670-672). IEEE. Sarajevo, Bosnia and Herzegovina.

Karaduzovic-Hadziabdic, K., & Spahic, R. (2018b). Class level code smell detection using machine learning methods. *Computational Methods and Telecommunication in Electrical Engineering and Finance* (pp. 74-99). Bosnia & Herzegovina, Balkans.

Karapetrovic, S., & Willborn, W. (1998). The system's view for clarification of quality vocabulary. *International Journal of Quality & Reliability Management*, *15*(1), 99-120.

Kaur, I., & Kaur, A. (2021). A novel four-way approach designed with ensemble feature selection for code smell detection. *IEEE Access*, *9*, 8695-8707.

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.

Khomh, F., Di Penta, M., & Gueheneuc, Y.G. (2009a). An exploratory study of the impact of code smells on software change-proneness. In *2009 16th Working Conference on Reverse Engineering* (pp. 75-84). IEEE. Lille, France.

Khomh, F., Vaucher, S., Guéhéneuc, Y.G., & Sahraoui, H. (2009b). A bayesian approach for the detection of code and design smells. In *2009 Ninth International Conference on Quality Software* (pp. 305-314). IEEE. Jeju, Korea (South).

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 and Software Technology*, *51*(1), 7-15.

Kumar, V., & Ram, M. (2021). *Predictive analytics: Modeling and optimization*. CRC Press, Taylor & Francis Group, Boca Raton, USA.

Li, H., & Thompson, S. (2010). Similar code detection and elimination for Erlang programs. In *Practical Aspects of Declarative Languages: 12th International Symposium, PADL 2010, Madrid, Spain, January 18-19, 2010. Proceedings 12* (pp. 104-118). Springer Berlin Heidelberg.

Liu, H., Jin, J., Xu, Z., Zou, Y., Bu, Y., & Zhang, L. (2019). Deep learning based code smell detection. *IEEE Transactions on Software Engineering*, *47*(9), 1811-1837.

Liu, H., Liu, Q., Niu, Z., & Liu, Y. (2015). Dynamic and automatic feedback-based threshold adaptation for code smell detection. *IEEE Transactions on Software Engineering*, *42*(6), 544-558.

Liu, H., Ma, Z., Shao, W., & Niu, Z. (2011). Schedule of bad smell detection and resolution: A new way to save effort. *IEEE Transactions on Software Engineering*, *38*(1), 220-235.

Macia Bertran, I., Garcia, A., & von Staa, A. (2011). An exploratory study of code smells in evolving aspect-oriented systems. In *Proceedings of the tenth International Conference on Aspect-oriented Software Development* (pp. 203-214). Association for Computing Machinery. Porto de Galinhas, Brazil.

Macia, I., Garcia, J., Popescu, D., Garcia, A., Medvidovic, N., & von Staa, A. (2012). Are automatically-detected code anomalies relevant to architectural modularity? An exploratory analysis of evolving systems. In *Proceedings of the 11th Annual International Conference on Aspect-oriented Software Development* (pp. 167-178). Association for Computing Machinery. Potsdam, Germany.

Maneerat, N., & Muenchaisri, P. (2011). Bad-smell prediction from software design model using machine learning techniques. In *2011 Eighth International Joint Conference on Computer Science and Software Engineering* (pp. 331-336). IEEE. Nakhonpathom, Thailand.

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*, 529-552.

Meananeatra, P., Rongviriyapanish, S., & Apiwattanapong, T. (2011). Using software metrics to select refactoring for long method bad smell. In *The 8th Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology Association of Thailand-Conference 2011* (pp. 492-495). IEEE. Khon Kaen, Thailand.

Mitchell, T.M. (1997). Does machine learning really work?. *AI magazine*, *18*(3), 11. https://doi.org/10.1609/aimag.v18i3.1303.

Moha, N., Guéhéneuc, Y.G., Duchien, L., & Le Meur, A.F. (2009). Decor: A method for the specification and detection of code and design smells. *IEEE Transactions on Software Engineering*, *36*(1), 20-36.

Moha, N., Guéhéneuc, Y.G., Meur, A.F.L., Duchien, L., & Tiberghien, A. (2010). From a domain analysis to the specification and detection of code and design smells. *Formal Aspects of Computing*, *22*, 345-361.

Murphy-Hill, E., & Black, A.P. (2010). An interactive ambient visualization for code smells. In *Proceedings of the 5th International Symposium on Software Visualization* (pp. 5-14). Association for Computing Machinery. Salt Lake City Utah, USA.

Nagy, C., & Cleve, A. (2017). A static code smell detector for SQL queries embedded in Java code. In *2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)* (pp. 147-152). IEEE. Shanghai, China.

Oizumi, W., Garcia, A., da Silva Sousa, L., Cafeo, B., & Zhao, Y. (2016). Code anomalies flock together: Exploring code anomaly agglomerations for locating design problems. In *Proceedings of the 38th International Conference on Software Engineering* (pp. 440-451). Association for Computing Machinery. Austin, Texas.

Olbrich, S., Cruzes, D.S., Basili, V., & Zazworka, N. (2009). The evolution and impact of code smells: A case study of two open source systems. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement* (pp. 390-400). IEEE. Lake Buena Vista, USA.

Olbrich, S.M., Cruzes, D.S., & Sjøberg, D.I. (2010). Are all code smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems. In *2010 IEEE International Conference on Software Maintenance* (pp. 1-10). IEEE. Timisoara, Romania.

Oliveto, R., Gethers, M., Bavota, G., Poshyvanyk, D., & De Lucia, A. (2011). Identifying method friendships to remove the feature envy bad smell (nier track). In *Proceedings of the 33rd International Conference on Software Engineering* (pp. 820-823). Association for Computing Machinery. Waikiki, USA.

Ouni, A., Kessentini, M., Sahraoui, H., Inoue, K., & Hamdi, M.S. (2015). Improving multi-objective code-smells correction using development history. *Journal of Systems and Software*, *105*, 18-39.

Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R., & De Lucia, A. (2018). On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. In *Proceedings of the 40th International Conference on Software Engineering* (pp. 482-482). Association for Computing Machinery. Gothenburg, Sweden.

Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., & De Lucia, A. (2014a). Do they really smell bad? a study on developers' perception of bad code smells. In *2014 IEEE International Conference on Software Maintenance and Evolution* (pp. 101-110). IEEE. Victoria, BC, Canada.

Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A., & Poshyvanyk, D. (2013). Detecting bad smells in source code using change history information. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 268-278). IEEE. Silicon Valley, CA, USA

Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., Poshyvanyk, D., & De Lucia, A. (2014b). Mining version histories for detecting code smells. *IEEE Transactions on Software Engineering*, *41*(5), 462-489.

Palomba, F., Oliveto, R., & De Lucia, A. (2017). Investigating code smell co-occurrences using association rule learning: A replicated study. In *2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)* (pp. 8-13). IEEE. Klagenfurt, Austria.

Palomba, F., Panichella, A., De Lucia, A., Oliveto, R., & Zaidman, A. (2016). A textual-based technique for smell detection. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)* (pp. 1-10). IEEE. Austin, TX, USA.

Paul, J., Khatri, P., & Kaur Duggal, H. (2023). Frameworks for developing impactful systematic literature reviews and theory building: What, Why and How?. *Journal of Decision Systems*, 1-14.

Pecorelli, F., Palomba, F., Di Nucci, D., & De Lucia, A. (2019). Comparing heuristic and machine learning approaches for metric-based code smell detection. In *2019 IEEE/ACM 27th International Conference on Program Comprehension* (pp. 93-104). IEEE. Montreal, QC, Canada.

Pereira dos Reis, J., Brito e Abreu, F., de Figueiredo Carneiro, G., & Anslow, C. (2022). Code smells detection and visualization: a systematic literature review. *Archives of Computational Methods in Engineering*, *29*(1), 47-94.

Peters, R., & Zaidman, A. (2012). Evaluating the lifespan of code smells using software repository mining. In *2012 16th European Conference on Software Maintenance and Reengineering* (pp. 411-416). IEEE. Szeged, Hungary.

Pigazzini, I., Fontana, F.A., & Walter, B. (2021). A study on correlations between architectural smells and design patterns. *Journal of Systems and Software*, *178*, 110984.

Pritam, N., Khari, M., Kumar, R., Jha, S., Priyadarshini, I., Abdel-Basset, M., & Long, H.V. (2019). Assessment of code smell for predicting class change proneness using machine learning. *IEEE Access*, *7*, 37414-37425.

Rahman, F., Bird, C., & Devanbu, P. (2012). Clones: What is that smell?. *Empirical Software Engineering*, *17*, 503-530.

Rasool, G., & Arshad, Z. (2015). A review of code smell mining techniques. *Journal of Software: Evolution and Process*, *27*(11), 867-895.

Rodriguez, A., Longo, M., & Zunino, A. (2015). Using bad smell-driven code refactorings in mobile applications to reduce battery usage. In *Simposio Argentino de Ingeniería de Software (ASSE 2015)-JAIIO 44*, Rosario.

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.

Schumacher, J., Zazworka, N., Shull, F., Seaman, C., & Shaw, M. (2010). Building empirical support for automated code smell detection. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 1-10). Association for Computing Machinery. Bolzano-Bozen, Italy.

Sharma, T., Efstathiou, V., Louridas, P., & Spinellis, D. (2021). Code smell detection by deep direct-learning and transfer-learning. *Journal of Systems and Software*, *176*, 110936.

Sharma, T., Fragkoulis, M., & Spinellis, D. (2016). Does your configuration code smell?. In *Proceedings of the 13th International Conference on Mining Software Repositories* (pp. 189-200). Association for Computing Machinery. Austin, Texas.

Sharma, T., Fragkoulis, M., & Spinellis, D. (2017). House of cards: code smells in open-source c# repositories. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 424-429). IEEE. Toronto, ON, Canada.

Sjøberg, D.I., Yamashita, A., Anda, B.C., Mockus, A., & Dybå, T. (2012). Quantifying the effect of code smells on maintenance effort. *IEEE Transactions on Software Engineering*, *39*(8), 1144-1156.

Szőke, G., Nagy, C., Fülöp, L.J., Ferenc, R., & Gyimóthy, T. (2015). FaultBuster: An automatic code smell refactoring toolset. In *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)* (pp. 253-258). IEEE. Bremen, Germany.

Taibi, D., Janes, A., & Lenarduzzi, V. (2017). How developers perceive smells in source code: A replicated study. *Information and Software Technology*, *92*, 223-235.

Tamrawi, A., Nguyen, H.A., Nguyen, H.V., & Nguyen, T.N. (2012). Build code analysis with symbolic evaluation. In *2012 34th International Conference on Software Engineering (ICSE)* (pp. 650-660). IEEE. Zurich, Switzerland.

Tandon, S., Kumar, V., & Singh, V.B. (2022). Empirical evaluation of code smells in open-source software (OSS) using Best Worst Method (BWM) and TOPSIS approach. *International Journal of Quality & Reliability Management*, *39*(3), 815-835.

Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton,H., & Noble, J. (2010). The qualitas corpus: A curated collection of java code for empirical studies. In *2010 Asia Pacific Software Engineering Conference* (pp. 336-345). IEEE. Sydney, NSW, Australia.

Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., & Poshyvanyk, D. (2015). When and why your code starts to smell bad. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* (Vol. 1, pp. 403-414). IEEE. Florence, Italy.

Vidal, S.A., Marcos, C., & Díaz-Pace, J.A. (2016). An approach to prioritize code smells for refactoring. *Automated Software Engineering*, *23*, 501-532.

Wagey, B.C., Hendradjaya, B., & Mardiyanto, M.S. (2015). A proposal of software maintainability model using code smell measurement. In *2015 International Conference on Data and Software Engineering* (pp. 25-30). IEEE. Yogyakarta, Indonesia.

Wang, Y., Hu, S., Yin, L., & Zhou, X. (2018). Using code evolution information to improve the quality of labels in code smell datasets. In *2018 IEEE 42nd Annual Computer Software and Applications Conference* (Vol. 1, pp. 48-53). IEEE. Tokyo, Japan.

Wen, J., Li, S., Lin, Z., Hu, Y., & Huang, C. (2012). Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, *54*(1), 41-59.

Witten, I.H., & Frank, E. (2002). Data mining: practical machine learning tools and techniques with Java implementations. *Acm Sigmod Record*, *31*(1), 76-77.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* (pp. 1-10). Association for Computing Machinery. London, England, United Kingdom.

Yamashita, A. (2014). Assessing the capability of code smells to explain maintenance problems: an empirical study combining quantitative and qualitative data. *Empirical Software Engineering*, *19*, 1111-1143.

Yamashita, A., & Counsell, S. (2013). Code smells as system-level indicators of maintainability: An empirical study. *Journal of Systems and Software*, *86*(10), 2639-2653.

Yamashita, A., & Moonen, L. (2012). Do code smells reflect important maintainability aspects?. In *2012 28th IEEE International Conference on Software Maintenance* (pp. 306-315). IEEE. Trento, Italy.

Yamashita, A., & Moonen, L. (2013c). Do developers care about code smells? An exploratory survey. In *2013 20th Working Conference on Reverse Engineering* (pp. 242-251). IEEE. Koblenz, Germany.

Yamashita, A., & Moonen, L. (2013b). Exploring the impact of inter-smell relations on software maintainability: An empirical study. In *2013 35th International Conference on Software Engineering (ICSE)* (pp. 682-691). IEEE. San Francisco, CA, USA.

Yamashita, A., & Moonen, L. (2013a). To what extent can maintenance problems be predicted by code smell detection?–An empirical study. *Information and Software Technology*, *55*(12), 2223-2242.

Yamashita, A.F., Benestad, H.C., Anda, B., Arnstad, P.E., Sjoberg, D.I., & Moonen, L. (2009). Using concept mapping for maintainability assessments. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement* (pp. 378-389). IEEE. Lake Buena Vista, FL, USA.

Zazworka, N., Seaman, C., & Shull, F. (2011a). Prioritizing design debt investment opportunities. In *Proceedings of the 2nd Workshop on Managing Technical Debt* (pp. 39-42). Association for Computing Machinery. Waikiki, USA.

Zazworka, N., Shaw, M.A., Shull, F., & Seaman, C. (2011b). Investigating the impact of design debt on software quality. In *Proceedings of the 2nd Workshop on Managing Technical Debt* (pp. 17-23). Association for Computing Machinery. Waikiki, USA.

Zhang, M., Hall, T., & Baddoo, N. (2011). Code bad smells: A review of current knowledge. *Journal of Software Maintenance and Evolution: Research and Practice*, *23*(3), 179-202.