DOI: https://doi.org/10.4038/jdrra.v3i2.93

The Journal of Desk Research Review and Analysis, Vol. 3, Issue 2, 2025, 134 - 157

CODE QUALITY ALARMS: A REVIEW OF TECHNIQUES, DATASETS, AND EMERGING TRENDS IN DETECTING SMELLS AND ANTI-PATTERNS

YVA Amarasinghe[1], PPGD Asanka[2], D Wickramaarachchi[3] and D Lorensuhewa[4]

Abstract

This systematic literature review examines research on code smell and anti-pattern detection in software engineering from 2020 to April 2025, aiming to synthesise detection methodologies, evaluate dataset practices, and identify critical research gaps and trends. A total of 49 peer-reviewed studies were selected from five major scholarly databases: Google Scholar, Scopus, ScienceDirect, IEEE Xplore, and the ACM Digital Library. Approximately 80% of the studies utilised custom-built datasets derived from open-source repositories like GitHub, with limited adoption of public datasets such as Qualitas Corpus, Fontana, and MLCQ. This dataset fragmentation undermines reproducibility and limits cross-study comparability. To reveal methodological trends, the reviewed techniques were categorised into static analysis, traditional machine learning, deep learning, and emerging paradigms such as large language models and human-in-the-loop approaches. This taxonomy highlights a shift toward hybrid frameworks that combine static analysis with AI-driven techniques, although standardised benchmarks remain scarce. Feature Envy was the most frequently studied code smell, underscoring its perceived impact on maintainability. Notably, 88% of studies focused exclusively on Java, revealing a strong language bias that restricts generalizability. The findings underscore the need for standardised, multilingual datasets, unified annotations, and broader adoption of advanced AI techniques to enhance scalability, practical relevance, and long-term software quality.

Keywords: Code Smells, Detection Techniques, Software Anti-Patterns, Software Maintenance, Software Quality

[1]Department of Industrial Management, University of Kelaniya, Sri Lanka.

Email: vishathakila2001@gmail.com https://orcid.org/0009-0006-6640-4413

[2]Department of Industrial Management, University of Kelaniya, Sri Lanka.

Email: dasanka@kln.ac.lk https://orcid.org/0000-0002-3433-5533

[3]Department of Industrial Management, University of Kelaniya, Sri Lanka.

Email: dilani@kln.ac.lk https://orcid.org/0000-0001-9042-6301

[4]Department of Industrial Management, University of Kelaniya, Sri Lanka.

Email: dlore241@kln.ac.lk https://orcid.org/0009-0006-5066-6777

Received date: 31.05.2025 Accepted date: 18.08.2025

Print Publishing Date: 31.10.2025 Web Publishing Date: 31.10.2025

Introduction

Background

In the domain of software engineering, maintaining high code quality is essential for ensuring the maintainability, scalability, and robustness of software systems. Poor design decisions and suboptimal practices often lead to the manifestation of structural (e.g., poor modularisation) and behavioural (e.g., inefficient logic flow) issues within the codebase, commonly referred to as code smells and anti-patterns. Code smells are indicators of potential problems in source code that suggest deeper design or implementation issues, often impairing maintainability and comprehension (Fowler, 2019). Anti-patterns refer to common but ineffective or counterproductive solutions to recurring design problems (Brown, 1998).

Such issues accumulate technical debt, making software harder to evolve and maintain, and often result in reduced developer productivity, increased defect rates, and degraded system performance (Z. Li et al., 2015). Familiar code smells include Long Method, where functions are excessively long and difficult to comprehend; Duplicate Code, which introduces redundancy and maintenance overhead; and Feature Envy, where one class inappropriately accesses data from another (Fowler, 2019). Similarly, typical anti-patterns include the God Object, which centralises excessive responsibilities; Spaghetti Code, characterised by tangled and unstructured control flow; and the Golden Hammer, where a single solution is applied indiscriminately, regardless of context (Brown, 1998).

Timely detection and mitigation of these issues are essential to managing software complexity and reducing long-term development costs (Mens & Tourwe, 2004).

Literature review

The detection of code smells and anti-patterns has been a longstanding concern in software quality assurance. Initial efforts predominantly utilised rule-based static analysis tools such as PMD, Checkstyle, and SonarQube. These tools rely on predefined heuristics and metric thresholds to identify structural code flaws (Lanza & Marinescu, 2011). While effective for detecting simple issues, they often lack contextual awareness and struggle to maintain precision and recall across large, evolving, or complex codebases.

To overcome these limitations, researchers have proposed various machine learning (ML) approaches that can learn from labelled datasets. Supervised learning models such as support vector machines (SVMs), decision trees, and random forests have been used to classify code smells based on static code metrics (Arcelli Fontana et al., 2012; Palomba et al., 2013). These approaches have demonstrated improved performance over static tools; however, their effectiveness remains constrained by dataset quality, limited generalizability across projects, and high dependence on manually engineered features.

The evolution of deep learning (DL) techniques further advanced the field. Models such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), long short-term memory (LSTM), and Transformers have been applied to code representations including token sequences, abstract syntax trees (ASTs), and control flow graphs (CFGs) (Dam et al., 2018; Martinez et al., 2013). These approaches enable automatic feature extraction and capture complex syntactic and semantic relationships in code. However, they require large-scale, high-quality, and diverse datasets, which are not always available, and their training is often computationally expensive.

Several public datasets have been used to support empirical evaluations, such as the MLCQ dataset (Madeyski & Lewowski, 2020) and Qualitas Corpus (Tempero et al., 2010). Nonetheless, these datasets

often lack cross-language coverage, consistent annotation standards, and real-world diversity, which hinders reproducibility and external validity in detection experiments.

More recently, large pre-trained models like CodeBERT and GraphCodeBERT, initially developed for natural language processing tasks, have shown potential in source code analysis, including smell and anti-pattern detection (Ahmad et al., 2021; Feng et al., 2020). These models exhibit strong semantic understanding but are still in early experimental stages for this domain. The lack of standard benchmarks, questions around interpretability, and difficulties in fine-tuning further limit their practical deployment in code quality tools.

Despite these technical advancements, the research landscape remains fragmented. Most studies target isolated smells or anti-patterns and fail to explore their interdependencies or composite effects (Ma et al., 2023; Yadav et al., 2024; M. Zhang & Jia, 2022). Many proposals focus narrowly on tool development or specific detection models without comprehensive comparative evaluation, and generalisation across paradigms, domains, and programming languages remains underexplored. In addition, comparative benchmarks with industry tools like SonarQube are infrequent, reducing the relevance and applicability of academic findings to practice.

Aim and Scope

This systematic literature review comprehensively analyses existing techniques for detecting code smells and anti-patterns. It provides a structured classification of detection methods, critically evaluates the datasets used, and highlights emerging trends and future research directions. The goal is to offer actionable insights for researchers and practitioners aiming to improve software quality through effective detection and mitigation strategies.

The review addresses the following objectives:

1. To identify code smells and anti-patterns studied in the literature, and identify the most frequently targeted issues.
2. To classify and evaluate detection techniques, highlighting their methodologies, strengths, and limitations.
3. To assess datasets used for training and evaluation, considering programming language support, annotation quality, and availability.
4. To identify challenges and opportunities for future work, focusing on recent advances such as large language models and multi-paradigm detection.

The research questions guiding this review are:

1. What types of code smells and anti-patterns have been studied in the existing literature?
2. What detection techniques have been proposed, and how are they categorised and compared?
3. What datasets have been used for training and evaluation, and what are their key characteristics?
4. What challenges and research gaps exist in current approaches to smell and anti-pattern detection?

By synthesising current knowledge and highlighting avenues for further research, this review aims to support the advancement of automated code quality assessment.

In summary, this review synthesises current knowledge on code smell and anti-pattern detection, highlighting methodological advancements, dataset challenges, and future research directions. By

providing actionable insights, it seeks to support both academic research and industrial practice in improving automated code quality assessment.

Methodology

This systematic literature review follows the PRISMA 2020 guidelines (Page et al., 2021) to ensure methodological rigour and transparency. The primary objective is to investigate existing approaches for detecting code smells and anti-patterns, assess the datasets used for evaluation, and explore emerging trends and research gaps.

Literature Search Strategy

To identify relevant publications, five digital libraries were selected based on their academic coverage, citation indexing, and relevance to software engineering and computer science:

- IEEE Xplore and ACM Digital Library: Chosen for their high-quality peer-reviewed conference and journal publications.
- ScienceDirect and Scopus: Included to broaden the search with interdisciplinary and indexed research.
- Google Scholar: Used to increase the comprehensiveness of the search by capturing grey literature, workshop papers (Haddaway et al., 2015).

The literature search covered the period from January 2020 to April 2025. Boolean queries were formulated to capture studies focused on both code smell detection and anti-pattern identification. The following generic Boolean string was adapted for each database's syntax:

("code smell" OR "code smells" OR "anti-pattern" OR "antipatterns")

AND ("detection" OR "identification" OR "recognition" OR "prediction")

AND ("static analysis" OR "heuristic" OR "machine learning" OR "deep learning" OR "rule-based" OR "graph-based" OR "large language model" OR "LLM" OR "NLP")

To maintain consistency and mitigate noise, the search in Google Scholar was limited to the first 20 pages of results, as relevance typically degrades beyond this threshold.

Inclusion and Exclusion Criteria

The following inclusion and exclusion criteria were applied:

Table 01: Inclusion and Exclusion Criteria

Criteria	Inclusion	Exclusion
Publication Year	2020 January - 2025 April	Before 2020
Language	English	Non-English
Source Type	Peer-reviewed journal articles, conference papers	Editorials, blog posts, slides, theses, Preprints
Full Text Access	Accessible full-text	Not accessible
Relevance to Topic	Focus on the detection of code smells/anti-patterns	Theoretical or review-only without empirical contributions
Methodological Content	Must describe or evaluate detection techniques, tools, or datasets	Lacks implementation details, datasets, or tool evaluation

Study Selection Process

The systematic search across five databases initially retrieved 7,781 records. After removing 1,203 duplicates, further automated filtering excluded publications before 2020, non-English articles, non-peer-reviewed works, and low-relevance Google Scholar entries, reducing the pool to 782 records. Title and abstract screening removed 387 irrelevant articles, and 104 full texts were inaccessible, leaving 291 articles for detailed eligibility assessment.

From these, 241 were excluded due to being reviews, theoretical works without practical evaluations, lacking specific detection techniques, or missing dataset/tool discussions. The final analysis included 49 primary studies providing empirical and methodological insights relevant to the research objectives. The selection process is summarised using a PRISMA flow diagram (Fig. 1), and metadata about each included study are available in Appendix A.

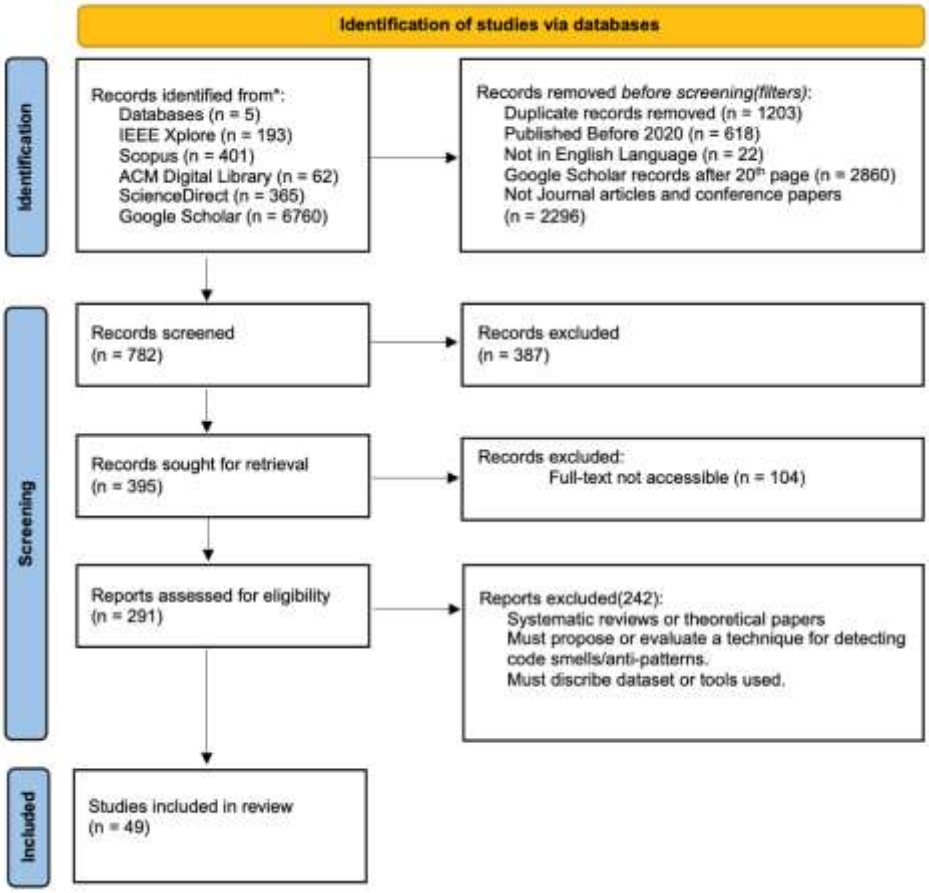

Figure 01: Study Selection Process Following PRISMA Flow Diagram

Data Extraction and Analysis

To enable structured synthesis, a data extraction form was designed, capturing the following dimensions:

- Metadata: Title, authors, venue, year.
- Detection Targets: Specific code smells or anti-patterns addressed.
- Detection Techniques: Methodologies used (e.g., static analysis, ML, DL, GNNs, LLMs).

- Datasets: Dataset name, language coverage, availability (public/private), and annotation approach.
- Evaluation Metrics: Precision, recall, F1-score, accuracy

A qualitative coding process was employed to classify detection techniques across studies. Key terms such as *AST*, *CNN*, *SVM*, *CodeBERT*, and *GPT* were extracted and mapped to one of seven methodological families based on shared computational principles. These include, Static and Rule-Based, relying on predefined rules and AST parsing (e.g., SonarQube); Traditional Machine Learning, using classical algorithms over engineered features (e.g., SVM, Random Forest); Deep Learning, which learns patterns from raw code (e.g., CNN, LSTM); Graph-Based Models, capturing code structure via graphs like ASTs or DFGs (e.g., GCN, GGNN); Hybrid Techniques, combining static and learning-based methods; NLP & Pretrained Models, leveraging transformers like CodeBERT; and Emerging Approaches, defined as novel paradigms, such as zero-shot, prompt-based LLMs, and human-in-the-loop systems that do not fit into conventional static, ML, or DL categories but are gaining traction in recent studies (Drori et al., 2024; Silva et al., 2024). This classification (Table 2) enabled structured thematic analysis. Additionally, we examined technique co-occurrence across studies to uncover hybridisation methods. For example, combining ASTs with CNNs or embeddings with transformers enhanced semantic representation (Barbez et al., 2020; Brdar et al., 2022), highlighting a growing shift toward multi-technique, scalable detection frameworks.

Results and Discussion

Overview of Study Selection and Publication Trends

The final selection comprised 49 primary studies, including 21 journal articles and 28 conference papers. As shown in Fig. 2, the annual distribution of publications from 2020 to early 2025 reveals relatively higher activity in 2021 and 2023, each contributing 11 studies. This pattern suggests periods of intensified scholarly engagement rather than a consistent trend. The lower count in 2025 likely reflects the incomplete publication cycle, as data collection concluded in April.

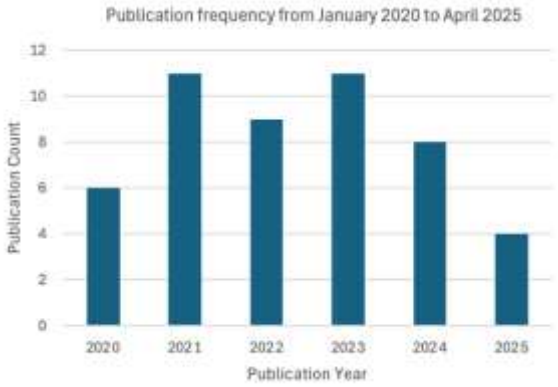

Figure 02: Annual Distribution of Selected Publications (2020–2025 April)

Geographical analysis of conference publications (Fig. 4) highlights strong representation from Asian countries, particularly China and India. Several studies also emerged from regions such as Europe, Australia, and North America, though with comparatively lower frequency. This distribution indicates a globally distributed but uneven research effort, with growing interest in code quality detection in emerging software development economies.

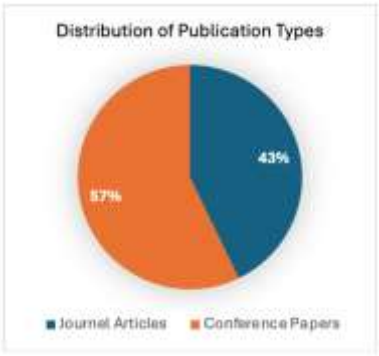

Figure 03: Distribution of Publication Types

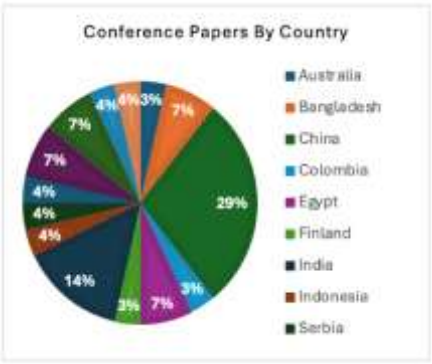

Figure 04: Distribution of Conference Papers by Country

Identified Code Smells and Anti-Patterns

The review identified 52 distinct code smells and anti-patterns that impact software maintainability, modularity, and readability. Fig. 5 illustrates their frequency distribution, with full details in Appendix B. A key issue observed was the inconsistent use of terminology across studies. For instance, "God Class" is labelled a code smell in some (Brdar et al., 2022) and an anti-pattern in others (Barbez et al., 2020). This terminological ambiguity complicates cross-study comparisons. To mitigate this, a unified analytical approach was adopted. The findings highlight the need for standardised taxonomies and more explicit conceptual definitions in future research to ensure consistency and comparability.

Feature Envy emerged as the most frequently addressed code smell, cited in 30 of the reviewed studies. This smell typically indicates excessive coupling between classes and a breach of object-oriented design principles, such as cohesion and encapsulation (Ma et al., 2023; Menshawy et al., 2023; H. Wang et al., 2020; M. Zhang & Jia, 2022). Its prevalence highlights a research focus on refactoring opportunities tied to method relocation and behavioural misplacement. Long Method, God Class, and Data Class followed closely, reaffirming concerns about low modularity and the violation of the Single Responsibility Principle (SRP) (Brdar et al., 2022). These findings show long-standing debates in software engineering regarding overburdened components and the challenge of decomposition (Alazba & Aljamaan, 2021; Nanadani et al., 2023).

Other frequently discussed smells included Long Parameter List and Complex Method, which emphasise interface design complexity and cognitive load (Fawaz et al., 2023; Y. Li & Zhang, 2022; Sukkasem & Soomlek, 2023; Yahya & Taha, 2023). However, a substantial number of identified smells, such as Brain Controller, Fat Repository, Shotgun Surgery, Swiss Army Knife, and Functional Decomposition, were reported only once or twice, mostly in architectural or domain-specific contexts (Pecorelli et al., 2020). Despite their lower frequency, their presence suggests emerging areas of concern that remain underexplored, possibly due to the difficulty of automated detection or the lack of suitable benchmarks.

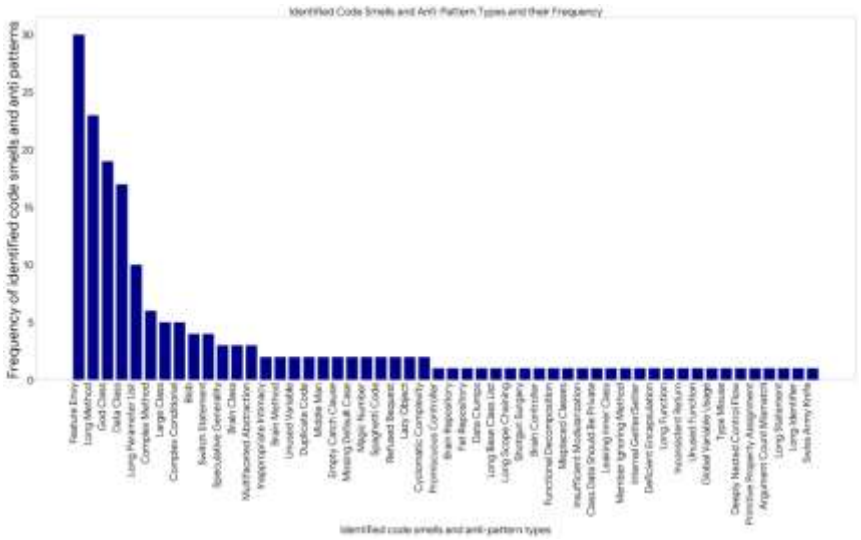

Figure 05: Frequency Distribution of Code Smells and Anti-Patterns Across Reviewed Studies

The analysis further reveals a concentration of research efforts on a small subset of well-known smells. This bias is likely driven by the availability of tool support and public datasets, reinforcing the detectability of these smells while neglecting others of equal or greater structural impact (Sukkasem & Soomlek, 2023). Fig. 6 illustrates that most studies addressed fewer than five smells (Brdar et al., 2022; Dewangan et al., 2022; Ho et al., 2023; Ma et al., 2023; Nanadani et al., 2023; Thakur et al., 2024; M. Zhang et al., 2025), suggesting a narrow empirical scope (Akhter et al., 2021; Saheb-Nassagh et al., 2022). While this allows for methodological rigour in focused contexts, it risks overlooking context-dependent or architectural smells that may significantly affect long-term software evolution.

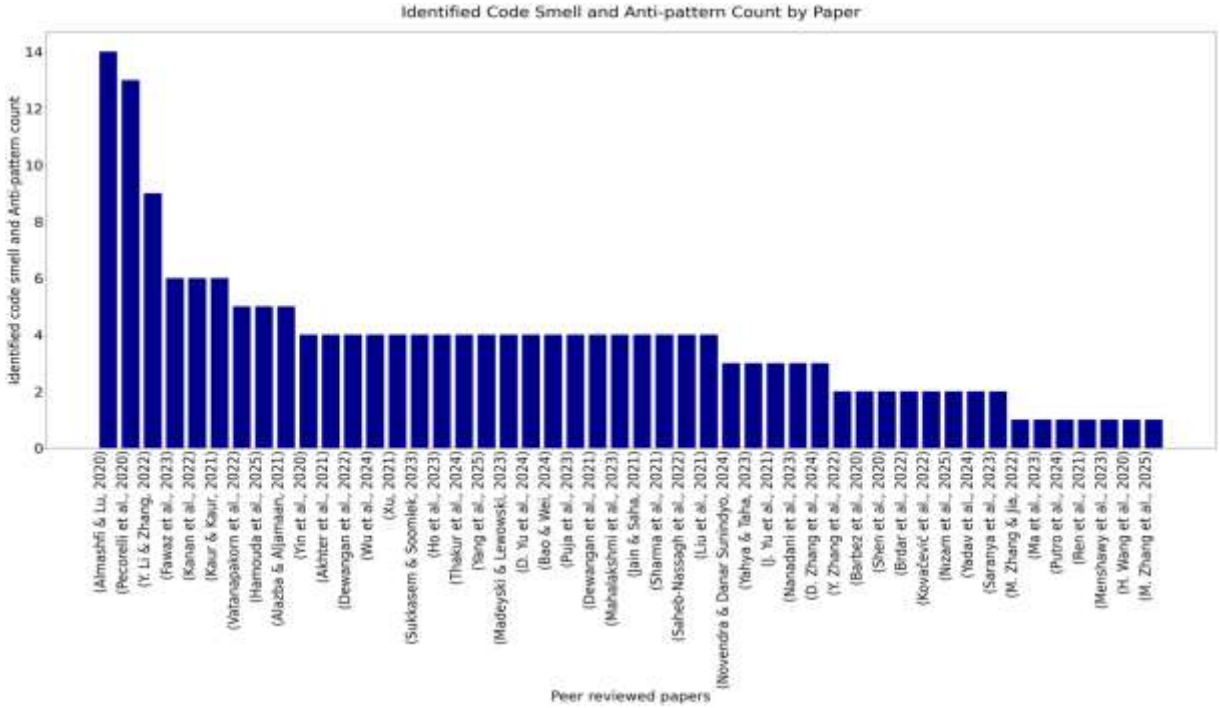

Figure 06: Frequency of Code Smells and Anti-Patterns Analysed in Reviewed Studies

To mitigate this imbalance, future research should broaden its scope beyond traditional smells, invest in building comprehensive datasets that include underrepresented patterns, and refine detection

methodologies capable of handling nuanced or higher-level design anomalies. This diversification is crucial for enhancing the ecological validity and practical relevance of automated code quality assessment tools.

Code Smell and Anti-Pattern Detection Techniques.

To systematically characterise the methodological approaches used for detecting code smells and anti-patterns, the reviewed studies were categorised into seven primary technique-based classes (Table 2). These categories reflect both traditional and emerging paradigms, encompassing static analysis tools, classical machine learning (Arcelli Fontana et al., 2012), deep learning architectures (Dam et al., 2018), and graph-based models (Allamanis et al., 2019). Recent advancements also include pretrained models like CodeBERT (Feng et al., 2020) and prompt-based LLM approaches. This taxonomy enables a nuanced understanding of the field's evolution, methodological diversity, and convergence across rule-based, statistical, and neural paradigms. Table 2 summarises these categories and the representative techniques identified across the literature.

Table 02: Code Smell and Anti-Pattern Detection Categories

Category	Definition / Approach	Key Characteristics	Representative Techniques / Models
Static and Rule-Based	Uses predefined rules, code metrics, or AST analysis without data-driven learning.	Deterministic, fast, explainable; does not generalise to unseen cases (Fowler, 2019; Tsantalis et al., 2018).	PMD, CheckStyle, DECOR, SonarQube
Traditional ML (SML)	Applies classical supervised learning on hand-engineered code features.	Requires labelled data and manual feature extraction (Azeem et al., 2019).	SVM, Random Forest, Decision Trees
Deep Learning (DL)	Learns hierarchical feature representations directly from code tokens or embeddings.	Automatic feature learning; capable of modelling complex patterns (Liu et al., 2021; S. Wang et al., 2016).	CNN, RNN, LSTM, GRU-based models
Graph-Based (GNNs)	Models code structure using graph representations like ASTs, CFGs, or DFGs.	Captures code semantics and relational context; handles structural dependencies (Allamanis et al., 2019).	GCN, GAT, GGNN
Hybrid Techniques	Combines static or rule-based logic with learning-based methods.	Balances interpretability with predictive performance (S. Wang et al., 2016).	AST + ML, Rules + DL
NLP & Pretrained Models	Uses language models pretrained on code to detect smells/anti-patterns.	Leverages transfer learning, code semantics, and token-level syntax (Feng et al., 2020).	CodeBERT, GraphCodeBERT, CodeT5

Emerging Approaches	Encompasses novel paradigms like zero-shot learning, prompt-based LLMs, human-in-the-loop, etc.	Reflects cutting-edge trends; often lacks standardised benchmarks but shows future directions (Fan et al., 2023).	GPT-4 (prompt-based), AlphaCode, Human-AI co-review

An analysis of the methodological landscape across the reviewed studies (Fig. 7) reveals a diverse but imbalanced use of detection techniques. Deep Learning and Traditional Supervised Machine Learning dominate the field, appearing in 22 and 21 studies respectively. This underscores a clear shift toward data-driven approaches capable of capturing complex code patterns (Menshawy et al. 2023, 2023; Vatanapakorn et al., 2022; Yin et al., 2020). Despite their scalability and performance, these methods often rely on extensive labelled datasets and require careful tuning.

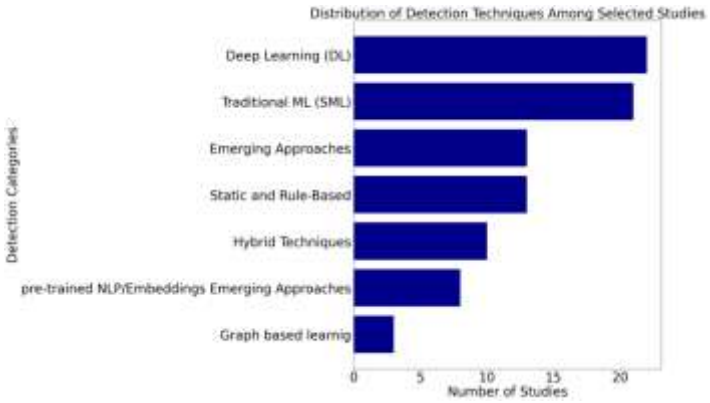

Figure 07. Frequency of Using Each Detection Technique

Static and rule-based techniques appeared in 13 studies, reflecting their continued relevance due to simplicity and interpretability, especially in environments where transparency is essential (Almashfi & Lu, 2020; Novendra & Danar Sunindyo, 2024). However, they struggle with adaptability and often fail to generalise across contexts. Emerging techniques also appeared in 13 studies, signalling a growing interest in integrating large-scale pretrained models and interactive refinement processes, although their real-world adoption remains nascent (Ozkaya, 2023; Wu et al., 2024).

Hybrid models and NLP/embedding-based techniques reflect attempts to combine symbolic and learned representations, enhancing robustness and semantic understanding (Thakur et al., 2024). Graph-based learning methods were rare, likely due to implementation complexity and lack of tool support, despite their potential to model code structure explicitly (Y. Li & Zhang, 2022; M. Zhang et al., 2025). Overall, the methodological spectrum is evolving, with conventional ML still dominant but newer paradigms gaining traction.

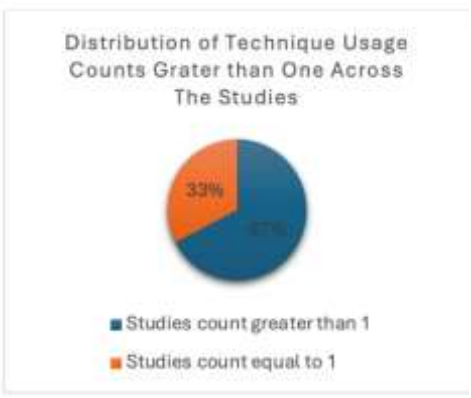

Figure 08. Count of Studies that Applied More than One Detection Method

To explore methodological synergy in code smell and anti-pattern detection, we performed a co-occurrence analysis (Fig. 9) based on the seven categories. This reveals interdependencies between detection methods and informs how the field is converging toward hybrid solutions. As shown in Fig. 8, 67.35% of the reviewed studies employed multi-technique strategies, highlighting a prevailing shift from isolated approaches to integrative frameworks.

Notably, deep learning frequently co-occurred with hybrid techniques and NLP/pretrained techniques, signalling a growing emphasis on leveraging semantic code understanding through neural embeddings and contextual representations (Bao & Wei, 2024; Kovačević et al., 2022; Liu et al., 2021; Ma et al., 2023; Nızam et al., 2025; Yang et al., 2025); D. (Zhang et al., 2024). Traditional machine learning showed flexible co-use with static/rule-based and emerging techniques, underscoring its enduring adaptability and role in structured pipelines.

In contrast, graph-based techniques, such as GNNs, exhibited minimal co-occurrence, possibly due to computational overhead and limited graph-structured datasets (M. Zhang et al., 2025). These patterns reflect a field in transition, where well-established pipelines are now enhanced by semantic-aware, data-driven models, ushering in a new phase of accuracy, interpretability, and automation in software quality assessment.

Integration Patterns in Code Smell and Anti-Pattern Detection Techniques

	Static and Rule-Based	Traditional ML (SML)	Deep Learing	Hybrid Techniques	Graph-Based (GNNs)	NLP & Pretrained Models	Emerging Approaches
Static and Rule-Based		5	5	3	0	1	2
Traditional ML (SML)			3	3	0	1	6
Deep Learing				8	2	6	7
Hybrid Techniques					1	1	4
Graph-Based (GNNs)						1	1
NLP & Pretrained Models							4
Emerging Approache							

Figure 09: Heatmap of Detection Method Co-Occurrence

A detailed analysis of detection techniques reveals that supervised machine learning and deep learning remain the most prevalent approaches in code smell and anti-pattern detection (Akhter et al., 2021; Dewangan et al., 2022; Vatanapakorn et al., 2022). Within supervised machine learning (Fig. 10), Random Forest classifiers are the most frequently employed due to their balance of interpretability, robustness, and firm performance on structured software metrics (Hamouda et al., 2025; Kovačević et al., 2022; Yadav et al., 2024). Decision Trees and Support Vector Machines (SVMs) also maintain widespread use, benefiting from simplicity and generalizability (Vatanapakorn et al., 2022; Yadav et al., 2024). Models such as Naive Bayes, Logistic Regression, and K-Nearest Neighbours often serve as baselines or components of ensembles to enhance predictive accuracy and comparative rigour (Jain & Saha, 2021; Madeyski & Lewowski, 2023). The rising adoption of ensemble methods, including Gradient Boosting, Extreme Gradient Boosting, AdaBoost, and Bagging, reflects an emphasis on mitigating noise and variance, though this comes at increased computational cost (Mahalakshmi et al., 2023). Less frequent but notable are rule-based learners like JRip, RuleFit, and Decision Tables, indicating exploratory efforts to improve interpretability within hybrid frameworks (Mahalakshmi et al., 2023; Yadav et al., 2024). Despite SML's dominance, challenges remain in feature engineering and scalability, while DL approaches offer automatic feature extraction but at the expense of transparency. Most studies rigorously evaluate multiple models per work, underscoring methodological diversity and comparative analysis (Akhter et al., 2021; Mahalakshmi et al., 2023; Menshawy et al., 2023; Shen et al., 2020).

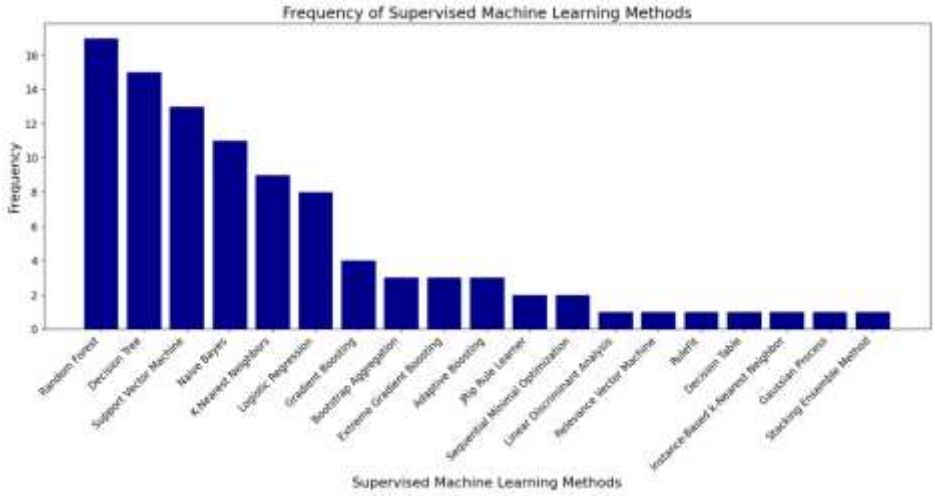

Figure 10: Frequency of Supervised Machine Learning Models Used as Detection Methods in Studies

In the domain of deep learning for code smell and anti-pattern detection (Fig. 11.), Long Short-Term Memory (LSTM) networks have emerged as the most prevalent, favored for effectively modeling sequential dependencies inherent in source code (Fawaz et al., 2023; Ho et al., 2023; Y. Li & Zhang, 2022; Nizam et al., 2023; Yahya & Taha, 2023). Convolutional Neural Networks and Gated Recurrent Units (GRUs) are also widely employed, frequently integrated into hybrid architectures that blend syntactic and semantic representations to capture multifaceted code features (Fawaz et al., 2023; Wu et al., 2024). More recently, Transformer-based models and self-attention mechanisms, exemplified by code-specific variants such as CodeBERT and CuBERT, signify a paradigm shift toward leveraging large-scale pretraining and transfer learning to enhance code understanding and representation (Bao & Wei, 2024; Kovačević et al., 2022; Nizam et al., 2023). The use of Bidirectional LSTMs further demonstrates the importance of modelling contextual information from both preceding and succeeding tokens in source code (H. Wang et al., 2020; Wu et al., 2024).

Despite this progress, advanced architectures like Graph Convolutional Networks (GCNs), Autoencoders, and Snapshot Ensembles remain underutilised, likely constrained by computational demands and a scarcity of standardised, graph-structured datasets (Y. Li & Zhang, 2022; Sharma et al., 2021). Notably, many studies employ multiple DL models alongside embedding techniques such as code2vec and code2seq to assess model robustness and generalizability across diverse smell types (Kovačević et al., 2022).

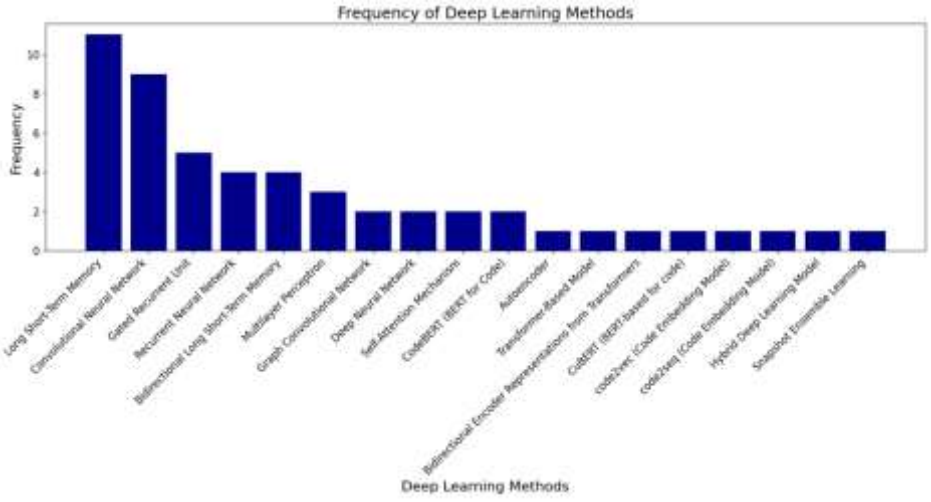

Figure 11: Frequency of Deep Learning Models Used as Detection Methods in Studies

While these trends highlight methodological sophistication, the concentration on a limited set of architectures suggests a degree of research saturation. Future investigations should prioritise exploring underexploited models, particularly graph-based and large language models, which are promising for capturing higher-level architectural smells. However, significant challenges remain due to heterogeneous datasets, annotation inconsistency, varying evaluation metrics, and positive publication bias, which hinder cross-study comparability. Addressing these through standardised benchmarks, rigorous dataset curation, and open reporting of negative results is critical for advancing detection robustness and generalizability.

Datasets Used In The Selected Studies

An analysis of the reviewed studies reveals a dominant reliance on custom-built datasets, with approximately 80% of code samples sourced from GitHub repositories (Fig. 13). While custom datasets enable researchers to tailor data to specific detection objectives, they introduce challenges related to reproducibility, comparability, and standardisation (Zakeri-Nasrabadi et al., 2023). Variations in labelling strategies, code selection criteria, and preprocessing pipelines result in methodological fragmentation, making it difficult to benchmark detection techniques across studies.

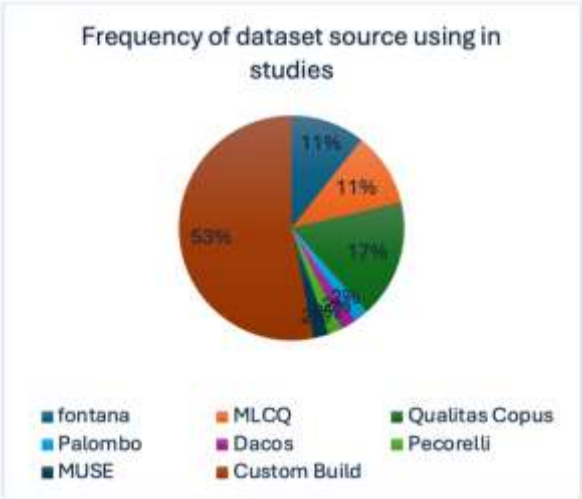

Figure 12: Frequency of Dataset Source Used in Studies

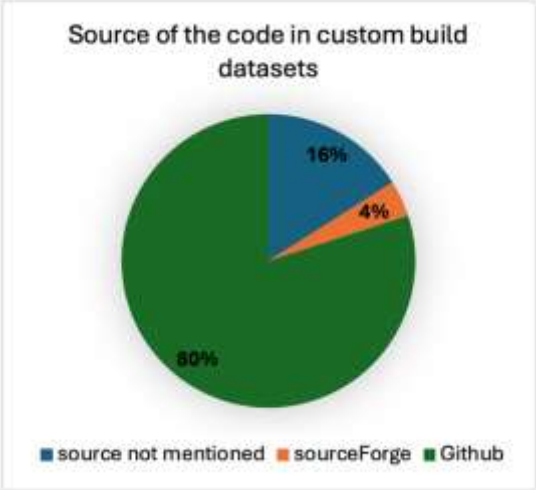

Figure 13: Source for the Codebase used in Custom-Build Datasets.

Among publicly available datasets, the Qualitas Corpus (Tempero et al., 2010) remains the most widely adopted, appreciated for its curated structure and compatibility with static analysis tools. Others, such as Fontana (Arcelli Fontana et al., 2016), MLCQ (Madeyski & Lewowski, 2020), Palomba (Palomba et al., 2018), and MUSE (Sasaki et al., 2012), are used less frequently, reflecting a lack of consensus around benchmark datasets. This underscores the need for standardised, high-quality corpora in the field.

As shown in Fig. 15, 37 of the 49 studies analysed rely on open-source project data, which, although accessible and diverse, may introduce inconsistencies in code quality, documentation, and structure. Only 9 of the 25 custom datasets were manually annotated (Fig. 14), indicating manual labelling improves accuracy. However, it is slow and depends on human judgment, which means it does not scale well and can lead to inconsistent or biased results.

A significant Java-centric bias was observed. 43 out of 49 studies focused on Java, with minimal representation of other languages such as Python, JavaScript, or Kotlin (Fig. 16). While Kotlin shares the JVM with Java, it introduces distinct features, such as null safety, coroutines, and more expressive syntax that alter smell patterns and design idioms (Putranto et al., 2020). Therefore, Java and Kotlin should be treated as separate languages in empirical studies. Language-specific semantics, tooling ecosystems, and idiomatic usage significantly influence the manifestation and detection of code smells, limiting the generalizability of Java-centric findings.

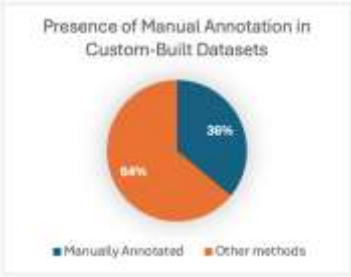

Figure 14: Presence of Manual Annotation in Custom-Built Datasets

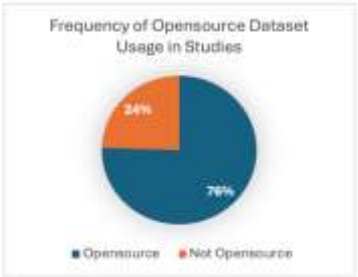

Figure 15: Frequency of Open-Source Dataset Usage in Studies

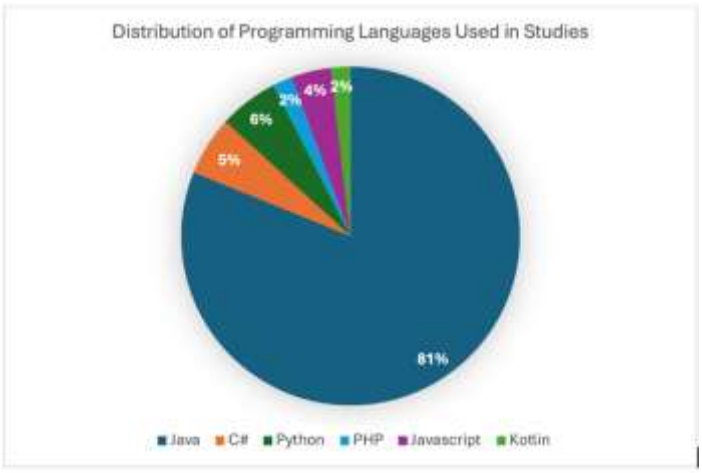

Figure 16: Distribution of Programming Languages Used in Studies

Limitations

This review reveals several limitations that affect the interpretability, generalizability, and reproducibility of code smell and anti-pattern detection research.

1. The search and selection bias remains a critical concern. Despite a systematic search strategy, inconsistencies in terminology (e.g., interchanging "code smells" and "anti-patterns") and limited indexing in specific databases may have led to the exclusion of relevant studies.
2. There is significant terminological and classification inconsistency, with studies often failing to clearly distinguish between code smells and anti-patterns, making meta-analysis challenging.
3. Methodological heterogeneity and language bias reduce the comparability of results due to variations in dataset construction, detection techniques, and evaluation metrics. Over 70% of studies focused exclusively on Java, likely due to mature tools like JDeodorant and Checkstyle (D. Zhang et al., 2024). This bias limits generalizability because languages like Python, JavaScript, and Kotlin have different syntax, semantics, and idiomatic patterns that affect how smells and anti-patterns manifest and are detected (Palomba et al., 2014).
4. The prevalence of custom-built datasets with limited public availability and varying annotation quality introduces subjectivity, reducing reproducibility and cross-study validation. The lack of standardised benchmarks and evaluation frameworks further compounds this issue.
5. Emerging AI-based methods like Graph Neural Networks and Large Language Models remain underexplored, despite their ability to capture rich semantic and structural representations of code (Feng et al., 2020).

Addressing these limitations is critical to advancing robust, scalable, and widely applicable detection approaches.

Conclusion

This systematic review of 49 studies from 2020 to April 2025 shows that traditional static analysis and machine learning remain prevalent in code smell and anti-pattern detection. Emerging techniques such as deep learning, graph-based models, and large language models are increasingly explored but are not yet widespread.

Many studies combine multiple methods to enhance results, but a lack of standardised evaluation and limited public availability of tools and datasets impede fair comparison and reproducibility.

The intense focus on Java (88% of studies) raises concerns about generalizability. Programming languages differ significantly in syntax, semantics, and idioms, influencing the types and detectability of smells and anti-patterns. Consequently, techniques validated on Java may not perform similarly on languages like Kotlin, Python, or JavaScript.

While progress is evident, significant challenges remain regarding evaluation consistency, dataset accessibility, and language diversity. Furthermore, advanced AI techniques such as Large Language Models and Graph Neural Networks offer promising capabilities to better model code's semantic and structural features. Systematic exploration and validation of these methods are recommended to advance detection performance and applicability.

Recommendations

Several key recommendations are proposed to enhance the effectiveness and impact of code smell and anti-pattern detection research.

1. Broaden Language Coverage: Future work must evaluate tools across multiple programming languages, with separate treatment for Kotlin and Java to account for semantic and syntactic differences.
2. Standardise Datasets: Establish open, annotated, multi-language datasets with consistent labelling for both smells and anti-patterns, including architectural issues.
3. Explore Emerging Techniques: Conduct systematic evaluations of LLM-based, prompt-driven, and zero-shot detection methods, as well as graph neural networks for structural analysis.
4. Human-Centric Approaches: Encourage integration of human-in-the-loop and explainable AI techniques to improve tool usability in real-world software engineering workflows.

By addressing these areas, future research can move toward more generalizable, reliable, and actionable detection frameworks.

References

Ahmad, W., Chakraborty, S., Ray, B., & Chang, K.-W. (2021). Unified Pre-training for Program Understanding and Generation. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Online. https://doi.org/10.18653/v1/2021.naacl-main.211

Akhter, N., Rahman, S., & Taher, K. A. (2021). An Anti-Pattern Detection Technique Using Machine Learning to Improve Code Quality. *2021 International Conference on Information and Communication Technology for Sustainable Development (ICICT4SD)*, 356–360. https://doi.org/10.1109/ICICT4SD50815.2021.9396937

Alazba, A., & Aljamaan, H. (2021). Code smell detection using feature selection and stacking ensemble: An empirical investigation. *Information and Software Technology*, *138*, 106648. https://doi.org/10.1016/j.infsof.2021.106648

Allamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2019). A Survey of Machine Learning for Big Code and Naturalness. *ACM Computing Surveys*, *51*(4), 1–37. https://doi.org/10.1145/3212695

Almashfi, N., & Lu, L. (2020). Code Smell Detection Tool for Java Script Programs. *2020 5th International Conference on Computer and Communication Systems (ICCCS)*, 172–176. https://doi.org/10.1109/ICCCS49078.2020.9118465

Arcelli Fontana, F., Braione, P., & Zanoni, M. (2012). Automatic detection of bad smells in code: An experimental assessment. *The Journal of Object Technology*, *11*(2), 5:1. https://doi.org/10.5381/jot.2012.11.2.a5

Arcelli Fontana, F., Mäntylä, M. V., Zanoni, M., & Marino, A. (2016). Comparing and experimenting machine learning techniques for code smell detection. *Empirical Software Engineering*, *21*(3), 1143–1191. https://doi.org/10.1007/s10664-015-9378-4

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. https://doi.org/10.1016/j.infsof.2018.12.009

Bao, H., & Wei, L. (2024). Multi-label code smell Detection Method based on CodeBERT. *2024 3rd International Conference on Electronics and Information Technology (EIT)*, 1–7. https://doi.org/10.1109/eit63098.2024.10762501

Barbez, A., Khomh, F., & Guéhéneuc, Y.-G. (2020). A machine-learning-based ensemble method for anti-patterns detection. *Journal of Systems and Software*, *161*, 110486.

Brdar, I., Vlajkov, J., Slivka, J., Grujic, K.-G., & Kovacevic, A. (2022). Semi-supervised detection of Long Method and God Class code smells. *2022 IEEE 20th Jubilee International Symposium on Intelligent Systems and Informatics (SISY)*, 403–408. https://doi.org/10.1109/SISY56759.2022.10036248

Brown, W. J. (Ed.). (1998). *AntiPatterns: Refactoring software, architectures, and projects in crisis*. Wiley.

Dam, H. K., Pham, T., Ng, S. W., Tran, T., Grundy, J., Ghose, A., Kim, T., & Kim, C.-J. (2018). *A deep tree-based model for software defect prediction* (Version 1). arXiv. https://doi.org/10.48550/ARXIV.1802.00921

Dewangan, S., Rao, R. S., Mishra, A., & Gupta, M. (2021). A Novel Approach for Code Smell Detection: An Empirical Study. *IEEE Access*, *9*, 162869–162883. https://doi.org/10.1109/ACCESS.2021.3133810

Dewangan, S., Rao, R. S., & Yadav, P. S. (2022). Dimensionally Reduction based Machine Learning Approaches for Code smells Detection. *2022 International Conference on Intelligent Controller and Computing for Smart Power (ICICCSP)*, 1–4. https://doi.org/10.1109/ICICCSP53532.2022.9862030

Drori, I., Boston University / Columbia University, Te'eni, D., & Tel Aviv University. (2024). Human-in-the-Loop AI Reviewing: Feasibility, Opportunities, and Risks. *Journal of the Association for Information Systems*, *25*(1), 98–109. https://doi.org/10.17705/1jais.00867

Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., & Zhang, J. M. (2023). Large Language Models for Software Engineering: Survey and Open Problems. *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, 31–53. https://doi.org/10.1109/ICSE-FoSE59343.2023.00008

Fawaz, O., Amaan, M., Sahu, S., Adnan, M., & Gupta, A. (2023). Experimentation of Code Smells Using Deep Learning Techniques. *Proc. Int. Conf. Contemp. Comput. Informatics, IC3I*, 369–373. Scopus. https://doi.org/10.1109/IC3I59117.2023.10397793

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *Findings of the Association for Computational Linguistics: EMNLP 2020*. Findings of the Association for Computational Linguistics: EMNLP 2020, Online. https://doi.org/10.18653/v1/2020.findings-emnlp.139

Fowler, M. (with Beck, K.). (2019). *Refactoring: Improving the design of existing code* (Second edition). Addison-Wesley.

Haddaway, N. R., Collins, A. M., Coughlin, D., & Kirk, S. (2015). The Role of Google Scholar in Evidence Reviews and Its Applicability to Grey Literature Searching. *PLOS ONE*, *10*(9), e0138237. https://doi.org/10.1371/journal.pone.0138237

Hamouda, E., El-Korany, A., & Makady, S. (2025). Smell-ML: A Machine Learning Framework for Detecting Rarely Studied Code Smells. *IEEE Access*, *13*, 12966–12980. https://doi.org/10.1109/ACCESS.2025.3530927

Ho, A., Bui, A. M. T., Nguyen, P. T., & Di Salle, A. (2023). Fusion of deep convolutional and LSTM recurrent neural networks for automated detection of code smells. *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, 229–234. https://doi.org/10.1145/3593434.3593476

Jain, S., & Saha, A. (2021). Improving performance with hybrid feature selection and ensemble machine learning techniques for code smell detection. *Science of Computer Programming*, *212*. Scopus. https://doi.org/10.1016/j.scico.2021.102713

Kanan, A., Sherief, N., & Abdelmoez, W. (2022). An Intelligent Framework based on CNN and Neutrosophic Technique for Detecting Code Smells and Ranking Code. *2022 32nd International Conference on Computer Theory and Applications (ICCTA)*, 31–37. https://doi.org/10.1109/ICCTA58027.2022.10206227

Kaur, I., & Kaur, A. (2021). A Novel Four-Way Approach Designed With Ensemble Feature Selection for Code Smell Detection. *IEEE Access*, *9*, 8695–8707. https://doi.org/10.1109/access.2021.3049823

Kovačević, A., Slivka, J., Vidaković, D., Grujić, K.-G., Luburić, N., Prokić, S., & Sladić, G. (2022). Automatic detection of Long Method and God Class code smells through neural source code embeddings. *Expert Systems with Applications*, *204*, 117607. https://doi.org/10.1016/j.eswa.2022.117607

Lanza, M., & Marinescu, R. (2011). *Object-oriented metrics in practice: Using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer.

Li, Y., & Zhang, X. (2022). *Multi-Label Code Smell Detection with Hybrid Model based on Deep Learning*. 42–47. https://doi.org/10.18293/SEKE2022-077

Li, Z., Avgeriou, P., & Liang, P. (2015). A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, *101*, 193–220. https://doi.org/10.1016/j.jss.2014.12.027

Liu, H., Jin, J., Xu, Z., Bu, Y., Zou, Y., & Zhang, L. (2021). Deep Learning Based Code Smell Detection. *IEEE Transactions on Software Engineering*, 1–1. https://doi.org/10.1109/TSE.2019.2936376

Ma, W., Yu, Y., Ruan, X., & Cai, B. (2023). Pre-trained Model-Based Feature Envy Detection. *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, 430–440. https://doi.org/10.1109/MSR59073.2023.00065

Madeyski, L., & Lewowski, T. (2020). MLCQ: Industry-Relevant Code Smell Data Set. *Proceedings of the Evaluation and Assessment in Software Engineering*, 342–347. https://doi.org/10.1145/3383219.3383264

Madeyski, L., & Lewowski, T. (2023). Detecting code smells using industry-relevant data. *Information and Software Technology*, *155*, 107112. https://doi.org/10.1016/j.infsof.2022.107112

Mahalakshmi, D., Kasinathan, P., Elangovan, D., Bhat, C. R., Balamurugan, M., & Sivakumar, S. (2023). Code Smell Detection using Hybrid Machine Learning Algorithms. *2023 5th International Conference on Inventive Research in Computing Applications (ICIRCA)*, 633–638. https://doi.org/10.1109/icirca57980.2023.10220911

Martinez, M., Duchien, L., & Monperrus, M. (2013, September). Automatically Extracting Instances of Code Change Patterns with AST Analysis. *2013 IEEE International Conference on Software Maintenance*. 2013 IEEE International Conference on Software Maintenance (ICSM), Eindhoven, Netherlands. https://doi.org/10.1109/icsm.2013.54

Mens, T., & Tourwe, T. (2004). A survey of software refactoring. *IEEE Transactions on Software Engineering*, *30*(2), 126–139. https://doi.org/10.1109/tse.2004.1265817

Menshawy, R. S., Yousef, A. H., & Salem, A. (2023). Comparing the Effectiveness of Machine Learning and Deep Learning Techniques for Feature Envy Detection in Software Systems. *2023 Intelligent Methods, Systems, and Applications (IMSA)*, 470–475. https://doi.org/10.1109/IMSA58542.2023.10217458

Nanadani, H., Saad, M., & Sharma, T. (2023). Calibrating Deep Learning-based Code Smell Detection using Human Feedback. In Moonen L., Newman C., & Gorla A. (Eds.), *Proc. - IEEE Int. Working Conf. Source Code Anal. Manip., SCAM* (pp. 37–48). Institute of Electrical and Electronics Engineers Inc.; Scopus. https://doi.org/10.1109/SCAM59687.2023.00015

Nizam, A., Avar, M. Y., Adaş, Ö. K., & Yanık, A. (2023). Detecting Code Smell with a Deep Learning System. *2023 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 1–5. https://doi.org/10.1109/ASYU58738.2023.10296577

Nızam, A., İslamoğlu, E., Kerem Adali, Ö., & Aydin, M. (2025). Optimizing Pre-Trained Code Embeddings With Triplet Loss for Code Smell Detection. *IEEE Access*, *13*, 31335–31350. https://doi.org/10.1109/access.2025.3542566

Novendra, R. D., & Danar Sunindyo, W. (2024). Emerging Trends in Code Quality: Introducing Kotlin-Specific Bad Smell Detection Tool for Android Apps. *IEEE Access*, *12*, 63895–63903. https://doi.org/10.1109/ACCESS.2024.3397055

Ozkaya, I. (2023). Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications. *IEEE Software*, *40*(3), 4–8. https://doi.org/10.1109/MS.2023.3248401

Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., Shamseer, L., Tetzlaff, J. M., Akl, E. A., Brennan, S. E., Chou, R., Glanville, J., Grimshaw, J. M., Hróbjartsson, A., Lalu, M. M., Li, T., Loder, E. W., Mayo-Wilson, E., McDonald, S., … Moher, D. (2021). The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ*, n71. https://doi.org/10.1136/bmj.n71

Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., & De Lucia, A. (2014). Do They Really Smell Bad? A Study on Developers' Perception of Bad Code Smells. *2014 IEEE International Conference on Software Maintenance and Evolution*, 101–110. https://doi.org/10.1109/ICSME.2014.32

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. *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 268–278. https://doi.org/10.1109/ase.2013.6693086

Palomba, F., Bavota, G., Penta, M. D., Fasano, F., Oliveto, R., & Lucia, A. D. (2018). On the diffuseness and the impact on maintainability of code smells: A large scale empirical investigation. *Empirical Software Engineering*, *23*(3), 1188–1221. https://doi.org/10.1007/s10664-017-9535-z

Pecorelli, F., Di Nucci, D., De Roover, C., & De Lucia, A. (2020). A large empirical assessment of the role of data balancing in machine-learning-based code smell detection. *Journal of Systems and Software*, *169*, 110693. https://doi.org/10.1016/j.jss.2020.110693

Puja, R. S., Fatema, T., Akhter, N., & Khatun, A. (2023). Prediction of Code Smell from Source Code: A Hybrid Approach. *2023 International Conference on Information and Communication Technology for Sustainable Development (ICICT4SD)*, 315–319. https://doi.org/10.1109/ICICT4SD59951.2023.10303449

Putranto, B. P. D., Saptoto, R., Jakaria, O. C., & Andriyani, W. (2020). A Comparative Study of Java and Kotlin for Android Mobile Application Development. *2020 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, 383–388. https://doi.org/10.1109/ISRITI51436.2020.9315483

Putro, H. P., Yuhana, U. L., Yuniarno, E. M., & Purnomo, M. H. (2024). Relevance Vector Machine for Code Smell Detection. *2024 IEEE 22nd International Conference on Industrial Informatics (INDIN)*, 1–7. https://doi.org/10.1109/INDIN58382.2024.10774521

Ren, S., Shi, C., & Zhao, S. (2021). Exploiting Multi-aspect Interactions for God Class Detection with Dataset Fine-tuning. *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, 864–873. https://doi.org/10.1109/compsac51774.2021.00119

Saheb-Nassagh, R., Ashtiani, M., & Minaei-Bidgoli, B. (2022). A probabilistic-based approach for automatic identification and refactoring of software code smells. *Applied Soft Computing*, *130*, 109658. https://doi.org/10.1016/j.asoc.2022.109658

Saranya, G., Mishra, D., Srikar, V., C, A., & Dooda, S. (2023). Code Smell Detection Using a Weighted Cockroach Swarm Optimization Algorithm. *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, 1–8. https://doi.org/10.1109/icccnt56998.2023.10306683

Sasaki, Y., Ishihara, T., Hotta, K., Hata, H., Higo, Y., Igaki, H., & Kusumoto, S. (2012). Preprocessing of Metrics Measurement Based on Simplifying Program Structures. *2012 19th Asia-Pacific Software Engineering Conference*, 120–127. https://doi.org/10.1109/APSEC.2012.59

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. https://doi.org/10.1016/j.jss.2021.110936

Shen, L., Liu, W., Chen, X., Gu, Q., & Liu, X. (2020). Improving Machine Learning-Based Code Smell Detection via Hyper-Parameter Optimization. *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*, 276–285. https://doi.org/10.1109/APSEC51365.2020.00036

Silva, L. L., Janio Silva, Montandon, J. E., & Valente, M. T. (2024). *Detecting Code Smells using ChatGPT: Initial Insights*. https://doi.org/10.13140/RG.2.2.36634.04802

Sukkasem, P., & Soomlek, C. (2023). Enhanced Machine Learning-Based Code Smell Detection Through Hyper-Parameter Optimization. *2023 20th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 297–302. https://doi.org/10.1109/JCSSE58229.2023.10202124

Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., & Noble, J. (2010, November). The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. *2010 Asia Pacific Software Engineering Conference*. 2010 17th Asia Pacific Software Engineering Conference (APSEC), Sydney, Australia. https://doi.org/10.1109/apsec.2010.46

Thakur, P. S., Jadeja, M., & Chouhan, S. S. (2024). CBReT: A Cluster-Based Resampling Technique for dealing with imbalanced data in code smell prediction. *Knowledge-Based Systems*, *286*, 111390. https://doi.org/10.1016/j.knosys.2024.111390

Tsantalis, N., Mansouri, M., Eshkevari, L. M., Mazinanian, D., & Dig, D. (2018). Accurate and efficient refactoring detection in commit history. *Proceedings of the 40th International Conference on Software Engineering*, 483–494. https://doi.org/10.1145/3180155.3180206

Vatanapakorn, N., Soomlek, C., & Seresangtakul, P. (2022). Python Code Smell Detection Using Machine Learning. *2022 26th International Computer Science and Engineering Conference (ICSEC)*, 128–133. https://doi.org/10.1109/ICSEC56337.2022.10049330

Wang, H., Liu, J., Kang, J., Yin, W., Sun, H., & Wang, H. (2020). Feature Envy Detection based on Bi-LSTM with Self-Attention Mechanism. *2020 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, 448–457. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom51426.2020.00082

Wang, S., Liu, T., & Tan, L. (2016). Automatically learning semantic features for defect prediction. *Proceedings of the 38th International Conference on Software Engineering*, 297–308. https://doi.org/10.1145/2884781.2884804

Wu, D., Mu, F., Shi, L., Guo, Z., Liu, K., Zhuang, W., Zhong, Y., & Zhang, L. (2024). iSMELL: Assembling LLMs with Expert Toolsets for Code Smell Detection and Refactoring. *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 1345–1357. https://doi.org/10.1145/3691620.3695508

Xu, W. (2021). *Multi-Granularity Code Smell Detection using Deep Learning Method based on Abstract Syntax Tree*. 503–509. https://doi.org/10.18293/SEKE2021-014

Yadav, P. S., Rao, R. S., & Mishra, A. (2024). An Evaluation of Multi-Label Classification Approaches for Method-Level Code Smells Detection. *IEEE Access*, *12*, 53664–53676. Scopus. https://doi.org/10.1109/ACCESS.2024.3387856

Yahya, H. M., & Taha, D. B. (2023). Detection Bad Code Smells By Using Deep Machine Learning Approaches. *2023 1st International Conference on Advanced Engineering and Technologies (ICONNIC)*, 281–286. https://doi.org/10.1109/ICONNIC59854.2023.10467672

Yang, Q., Yu, D., Wang, S., Xu, Y., Chen, X., Chen, J., & Hu, B. (2025). Enhancing structural knowledge in code smell identification: A fusion learning framework combining AST-based metrics with semantic embeddings. *Expert Systems with Applications*, *263*, 125725. https://doi.org/10.1016/j.eswa.2024.125725

Yin, Y., Su, Q., & Liu, L. (2020). Software smell detection based on machine learning and its empirical study. In T. Wang, T. Chai, H. Fan, & Q. Yu (Eds.), *Second Target Recognition and Artificial Intelligence Summit Forum* (p. 27). SPIE. https://doi.org/10.1117/12.2550500

Yu, D., Yang, Q., Chen, X., Chen, J., Wang, S., & Xu, Y. (2024). Actionable code smell identification with fusion learning of metrics and semantics. *Science of Computer Programming*, *236*, 103110. https://doi.org/10.1016/j.scico.2024.103110

Yu, J., Mao, C., & Ye, X. (2021). A Novel Tree-based Neural Network for Android Code Smells Detection. *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, 738–748. https://doi.org/10.1109/QRS54544.2021.00083

Zakeri-Nasrabadi, M., Parsa, S., Esmaili, E., & Palomba, F. (2023). A Systematic Literature Review on the Code Smells Datasets and Validation Mechanisms. *ACM Computing Surveys*, *55*(13s), 1–48. https://doi.org/10.1145/3596908

Zhang, D., Song, S., Zhang, Y., Liu, H., & Shen, G. (2024). Code Smell Detection Research Based on Pre-training and Stacking Models. *IEEE Latin America Transactions*, *22*(1), 22–30. https://doi.org/10.1109/TLA.2024.10375735

Zhang, M., & Jia, J. (2022). Feature Envy Detection with Deep Learning and Snapshot Ensemble. *2022 9th International Conference on Dependable Systems and Their Applications (DSA)*, 215–223. https://doi.org/10.1109/DSA56465.2022.00037

Zhang, M., Jia, J., Capretz, L. F., Hou, X., & Tan, H. (2025). Graph neural network-based long method and blob code smell detection. *Science of Computer Programming*, *243*, 103284. https://doi.org/10.1016/j.scico.2025.103284

Zhang, Y., Ge, C., Hong, S., Tian, R., Dong, C., & Liu, J. (2022). DeleSmell: Code smell detection based on deep learning and latent semantic analysis. *Knowledge-Based Systems*, *255*, 109737. https://doi.org/10.1016/j.knosys.2022.109737

Appendix A
Data Extraction Framework

Field	Description
DOI	Provides a unique identifier for precise referencing of each study.
Title	Essential for identifying and reporting individual papers.
Publication Year	Enables analysis of research trends over time.
Code Smells and Anti-Patterns	Specifies targeted defects to understand detection focus(RQ1).
Detection Technique	Categorizes methods used to address research question 2 (RQ2).
ML Algorithm	Details machine learning models applied, informing RQ2 insights.
LLM Model	Identifies large language models used, contributing to RQ2 analysis.
Dataset Used	Indicates data sources, supporting frequency and usage analysis.
Dataset Type	Shows dataset origin, impacting reproducibility of results.
Dataset Language	Highlights programming languages involved, assessing domain scope.
Tool Benchmarked	Identifies evaluation platforms, relevant to RQ2 comparative analysis.
Metrics Used	Reflects evaluation rigour through performance measures.
Results Summary	Provides a snapshot of model effectiveness for comparison.
Threats to Validity	Summarises quality assurance and study limitations.
Contribution to Research	Assesses novelty and impact of the study (RQ4).
Gaps Identified	Highlights open challenges, guiding future research focus (RQ4).
Future Work Proposed	Suggests directions for extending research efforts.
Reproducibility	Indicates availability of code/data, ensuring research credibility.

Appendix B
Table 03: Identified Code Smells and Anti-Patterns with Related Studies

Code Smell/Anti-Pattern	Frequency	Related Studies
Data Class	17	(Alazba & Aljamaan, 2021; Dewangan et al., 2021, 2022; Fawaz et al., 2023; Jain & Saha, 2021; Kaur & Kaur, 2021; Madeyski & Lewowski, 2023; Mahalakshmi et al., 2023b; Saheb-Nassagh et al., 2022; Saranya et al., 2023; Shen et al., 2020; Sukkasem & Soomlek, 2023; Thakur et al., 2024; Yang et al., 2025; Yin et al., 2020; D. Yu et al., 2024; D. Zhang et al., 2024)
God Class	19	(Alazba & Aljamaan, 2021; Barbez et al., 2020; Brdar et al., 2022; Dewangan et al., 2021, 2022; Jain & Saha, 2021; Kanan et al., 2022; Kaur & Kaur, 2021; Kovačević et al., 2022; Mahalakshmi et al., 2023; Novendra & Danar Sunindyo, 2024; Pecorelli et al., 2020; Ren et al., 2021; Sukkasem & Soomlek, 2023; Thakur et al., 2024; Wu et al., 2024; Yahya & Taha, 2023; Yang et al., 2025; D. Zhang et al., 2024)
Long Method	23	(Alazba & Aljamaan, 2021; Bao & Wei, 2024; Brdar et al., 2022; Dewangan et al., 2021; Fawaz et al., 2023; Jain & Saha, 2021; Kanan et al., 2022; Kaur & Kaur, 2021; Kovačević et al., 2022; Y. Li & Zhang, 2022; Liu et al., 2021; Madeyski & Lewowski, 2023; Mahalakshmi et al., 2023; Pecorelli et al., 2020; Sukkasem & Soomlek, 2023; Vatanapakorn et al., 2022; Wu et al., 2024; Yadav et al., 2024; Yahya & Taha, 2023; Yang et al., 2025; Yin et al., 2020; D. Yu et al., 2024; M. Zhang et al., 2025)

Switch Statement	4	(Alazba & Aljamaan, 2021; Dewangan et al., 2022; Fawaz et al., 2023; Kaur & Kaur, 2021)
Long Parameter List	10	(Alazba & Aljamaan, 2021; Almashfi & Lu, 2020; Dewangan et al., 2022; Fawaz et al., 2023; Kanan et al., 2022; Y. Li & Zhang, 2022; Nanadani et al., 2023; Pecorelli et al., 2020; Thakur et al., 2024; Vatanapakorn et al., 2022)
Complex Method	6	(Bao & Wei, 2024; Ho et al., 2023; Y. Li & Zhang, 2022; Nanadani et al., 2023; Saheb-Nassagh et al., 2022; Sharma et al., 2021)
Complex Conditional	5	(Bao & Wei, 2024; Ho et al., 2023; Y. Li & Zhang, 2022; Pecorelli et al., 2020; Sharma et al., 2021)
Feature Envy	30	(Akhter et al., 2021; Bao & Wei, 2024; Barbez et al., 2020; Dewangan et al., 2021; Fawaz et al., 2023; Ho et al., 2023; Jain & Saha, 2021; Kanan et al., 2022; Kaur & Kaur, 2021; Liu et al., 2021; Ma et al., 2023; Madeyski & Lewowski, 2023; Mahalakshmi et al., 2023; Menshawy et al., 2023; Pecorelli et al., 2020; Saheb-Nassagh et al., 2022; Saranya et al., 2023; Sharma et al., 2021; Shen et al., 2020; Sukkasem & Soomlek, 2023; Thakur et al., 2024; H. Wang et al., 2020; Wu et al., 2024; Xu, 2021; Yadav et al., 2024; Yahya & Taha, 2023; Yang et al., 2025; Yin et al., 2020; D. Yu et al., 2024; M. Zhang & Jia, 2022)
Multifaceted Abstraction	3	(Ho et al., 2023; Nanadani et al., 2023; Sharma et al., 2021)
Magic Number	2	(Y. Li & Zhang, 2022; Putro et al., 2024)
Long Identifier	1	(Y. Li & Zhang, 2022)
Long Statement	1	(Y. Li & Zhang, 2022)
Missing Default Case	2	(Almashfi & Lu, 2020; Y. Li & Zhang, 2022)
Empty Catch Clause	2	(Y. Li & Zhang, 2022; Xu, 2021)
Lazy Object	2	(Akhter et al., 2021; Almashfi & Lu, 2020)
Cyclomatic Complexity	2	(Almashfi & Lu, 2020; Nızam et al., 2025)
Argument Count Mismatch	1	(Almashfi & Lu, 2020)
Primitive Property Assignment	1	(Almashfi & Lu, 2020)
Unused variable	2	(Almashfi & Lu, 2020; Nızam et al., 2025)
Deeply Nested Control flow	1	(Almashfi & Lu, 2020)
Duplicate Code	2	(Almashfi & Lu, 2020; Wu et al., 2024)
Type Misuse	1	(Almashfi & Lu, 2020)
Global Variable Usage	1	(Almashfi & Lu, 2020)
Unused Function	1	(Almashfi & Lu, 2020)
Inconsistent Return	1	(Almashfi & Lu, 2020)
Long Function	1	(Almashfi & Lu, 2020)
Insufficient Modularization	1	(Xu, 2021)
Deficient Encapsulation	1	(Xu, 2021)

Internal Getter/Setter	1	(J. Yu et al., 2021)
Member Ignoring Method	1	(J. Yu et al., 2021)
Leaking Inner Class	1	(J. Yu et al., 2021)
Class Data Should be provate	1	(Hamouda et al., 2025)
Spectulative Generality	3	(Hamouda et al., 2025; Pecorelli et al., 2020; Saheb-Nassagh et al., 2022)
Inappropriate Intimacy	2	(Hamouda et al., 2025; Pecorelli et al., 2020)
Refused Bequest	2	(Hamouda et al., 2025; Pecorelli et al., 2020)
Misplaced Classes	1	(Liu et al., 2021)
Brain Class	3	(Novendra & Danar Sunindyo, 2024; D. Zhang et al., 2024; Y. Zhang et al., 2022)
Brain Method	2	(Novendra & Danar Sunindyo, 2024; Y. Zhang et al., 2022)
Shotgun Surgery	1	(Kaur & Kaur, 2021)
Long Scope Chaining	1	(Vatanapakorn et al., 2022)
Long Base Class List	1	(Vatanapakorn et al., 2022)
Fat Repository	1	(Pecorelli et al., 2020)
Brain Repository	1	(Pecorelli et al., 2020)
Promiscuous Controller	1	(Pecorelli et al., 2020)
Brain Controller	1	(Pecorelli et al., 2020)
Large Class	5	(Fawaz et al., 2023; Kanan et al., 2022; Liu et al., 2021; Vatanapakorn et al., 2022; Yin et al., 2020)
Data Clumps	1	(Kanan et al., 2022)
Middle Man	2	(Hamouda et al., 2025; Pecorelli et al., 2020)
Blob	4	(Akhter et al., 2021; Madeyski & Lewowski, 2023; Puja et al., 2023; D. Yu et al., 2024)
Spaghetti Code	2	(Akhter et al., 2021; Puja et al., 2023)
Functional Decomposition	1	(Puja et al., 2023)
Swiss Army Knife	1	(Puja et al., 2023)