Code smells and refactoring: a tertiary systematic literature review

Abhishilpa Nandini* and Randeep Singh

Department of Computer Science and Engineering,
IEC University,
Baddi Solan HP, 173205, India
Email: abhishilpa05@gmail.com
Email: randeeppoonia@gmail.com
*Corresponding author

Amit Rathee

Department of Computer Science,
Government College,
Barota, 131301, Sonipat (HR), India
Email: amit1983_rathee@rediffmail.com

Abstract: Software systems with code smells are difficult to maintain and evolve, and this impaired quality raises question marks on their future sustainability. Researchers have spent decades studying refactoring and code smells, which are key factors behind this problem. In lieu of the fact that the literature contains a huge collection of research publications that keeps evolving with time, dealing with code smell and refactoring activities is a challenge. Therefore, this paper targets a tertiary systematic literature survey. It aims at defining code smell and refactoring in general, identifying and analysing various tools and techniques available for code smell along with refactoring, identifying standard datasets available in the literature for the research community, and determining actively tackled code smells. This review paper considers 280 primary research publications collected from leading databases. The presented observations and recommendations are crucial for academic researchers as well as industry professionals.

Keywords: software quality; code smells; refactoring; refactoring tools; tertiary study; systematic literature survey; review; maintenance; quality decay; software sustainability.

Reference to this paper should be made as follows: Nandini, A., Singh, R. and Rathee, A. (2024) 'Code smells and refactoring: a tertiary systematic literature review', *Int. J. System of Systems Engineering*, Vol. 14, No. 1, pp.83–143.

Biographical notes: Abhishilpa Nandini is a Research Scholar in the Department of Computer Science and Engineering, IEC University Baddi Solan, HP, India. She received her BTech from Himachal Pradesh University (in the year 2014) and MTech from IEC University (in year 2018). She has four year working experience in teaching. Her current research interests are in software engineering.

Randeep Singh working as a Professor in the Department of Computer Science and Engineering at IEC University, Baddi, Solan, HP. He received his MTech from Kurukshetra University and PhD from Maharishi Markandeshwar (Deemed to be University), Mullana, NAAC Accredited highest Grade A++ University. He is in teaching profession for more than 15+ years. He has published about 30 research papers in international, national journals and refereed international conferences. He is having 7 patents and 2 books. His current research interests are in software engineering, IOT and data science.

Amit Rathee is currently an Assistant Professor at Government College, Barota, Haryana. He had a consistent and good academic record throughout his career. He did his PhD in the year 2020 from NIT Kurukshetra with 15 publications and 1 copyright out of his thesis in reputed international and national journals and conferences from IEEE, ACM, Elsevier, and Springer, etc., which are SCI and Scopus indexed. He has 10+ years of teaching and research experience. He is a Reviewer of Elsevier and Springer Journals. His areas of interest are software engineering, software security, soft computing, and artificial intelligence.

1 Introduction and motivation

In software development terminologies, software maintenance is an inevitable activity and generally incurs 50–80% of software cost. During this process time constraints, market pressure, negligence at the end of the developer, lack of knowledge about appropriate design principles, etc. are some of the factors that result in the degraded quality of the software and the reason behind the introduction of bad smells (termed as code smells) (Tufano, 2015; Singh et al., 2019, 2022; Michele et al., 2015; Kaur and Kaur, 2015). Fowler and Beck (2018) define code smells as the symptoms of design flaws (due to violation of design principles) that affect the architectural design of a software system negatively and they generally give rise to various problems such as technical debt, enhanced maintenance cost, evolution and understandability issues. They informally defined 22 types of code smell that denote design flaws at various levels and such symptoms must be controlled as soon as possible to curb future negative consequences. However, it is important to note here that efficient identification of code smell is much more challenging as opposed to what is advertised in the literature.

Bad smells in code can be eliminated by the process termed refactoring (Fowler and Beck, 2018; Roberts et al., 1997; Singh et al., 2020). A refactoring procedure involves transforming the source code of a software system so as to maintain its observable behaviour while improving quality by mitigating smells. Identifying which refactoring to use and when to use it is a major challenge in software engineering research. The terms ‘code smell’ and ‘refactoring’ are first coined in 1999 (Fowler and Beck, 2018). Software engineers and researchers have explored various dimensions associated with the metaphor since its inception. Among these are identifying smells using various techniques, exploring the relationships between smells, exploring their causes, and exploring their effects. As a result of a large number of resources available, it is difficult for researchers and practitioners alike to understand the status quo when it comes to tools, methods, and techniques for determining software smells. It is possible, through the analysis and synthesis of available information, to not only improve the software engineering community's understanding of existing knowledge but also identify challenges in the present methods and opportunities for improvement.

In literature, few researchers carries out systematic literature surveys in the recent past (Lacerda et al., 2020; Agnihotri and Chug, 2020; Al-Shaaby et al., 2020; Sabir et al., 2019; Singh and Kaur, 2018; Baqais and Alshayeb, 2020; Kaur and Dhiman, 2019; Kaur, 2020; Kaur et al., 2021a, 2021b; AlOmar et al., 2021; Caram et al., 2019; Abid et al., 2020; de Paulo Sobrinho et al., 2018). However, such works either needs improvement and/or are not useful for academician and research professional because of many reasons, namely,

1 the small size of considered secondary studies for evaluation (Lacerda et al., 2020; Agnihotri and Chug, 2020; Al-Shaaby et al., 2020; Sabir et al., 2019)

2 lack of in-depth analysis of the inter-relationship between code smell and refactoring (Agnihotri and Chug, 2020)

3 inability to deal with code smell and refactoring together and exploring only single aspect such of code smell and refactoring opportunities (Singh et al., 2018; Baqais and Alshayeb, 2020; Kaur et al., 2021a, 2021b; AlOmar et al., 2021; Caram et al., 2019; Kaur and Dhiman, 2019; Kaur, 2020).

de Paulo Sobrinho et al.(2018) and Abid et al. (2020) carry out an in-depth systematic literature survey in the recent past, however, such analysis needs reinvestigation with time due to evolving nature of code smell and refactoring research field. Based on the study of existing literature, the following are the main motivation that guides carrying out a systematic literature survey in this paper:

1 A smell detection system can enhance software maintenance activities that are needed for quality assurance.

2 To facilitate software developers' understanding of code smells, which is one of the least known software issues.

3 Code smell and refactoring is an active research area in software engineering, so, there is a huge collection of literature available that keeps on evolving at regular intervals. A systematic literature survey in this case helps academicians as well as industry professionals by consolidating the vast literature in a single place. Thus, it is mandatory to carry out at regular intervals for an active research field.

4 To the best of the author's knowledge based on the current literature position, it is strongly believed that there is a need of carrying out a systematic literature survey that is based on a large-sized dataset in order to reduce the current research gap.

The rest of this paper is organised as follows: Section 2 mentions the background of current related works on the topic of systematic literature survey, Section 3 discusses the considered research methodology of this paper, Section 4 elaborates on obtained results and provides their interpretation, Section 5 summarises threats to validity of this research and finally, Section 6 provides the conclusion and future work remarks.

2 Background of related works

In literature, there is a huge amount of research on how refactoring and code smell affects the performance of a software system. In spite of this, only a few systematic literature reviews have been conducted in the field of code smell detection and refactoring. Arass et al. (2019) proposed a System of Systems (SoS) framework for efficiently handling big data by organising this data at different levels. However, in order to keep this paper focused and short, this section of the paper summarises only recent literature work related to systematic surveys instead of discussing each and every piece of paper related to code smell and refactoring. However, it is our strong belief that such knowledge can be easily gathered by studying the below-mentioned papers.

Lacerda et al. (2020) carry out a tertiary systematic literature survey to identify observations and challenges in the field of code smell and refactoring. They carries out an investigation on only 40 primary studies selected during the period 1992 to, 2018. The investigation is carried around five research questions related to code smell and refactoring definition, code smell detection and refactoring approaches, and the most commonly used refactoring techniques and tools.

Agnihotri and Chug (2020) carry out a survey on the issues related to software metrics, code smell, and refactoring by selecting a total of 68 publications between, 2001 and, 2019. The investigation is based on three criteria namely types of code smells identified, the type of refactoring action used, and the relationship between their impacts on software metrics.

Singh et al. (2018) carry out a systematic literature survey by selecting 238 research papers up to 2015. They carry out an in-depth general investigation of code smells and the role of antipatterns in reference to refactoring. However, the study mainly focuses on refactoring with respect to code smells belonging to object-oriented software systems only. The paper is helpful in enhancing the attentiveness of the readers related to code smells and antipatterns.

Kaur and Dhiman (2019) carry out a survey to investigate search-based approaches used to identify code smells from object-oriented software systems. The authors conclude that many of the code smells are not properly formally defined, most of the used techniques are not publically available to reproduce obtained results, commercial/industry standard projects should be used for evaluation purposes, and threshold values used are subjective to the expert's knowledge.

Menshawy et al. (2021) carry out an investigation to identify different challenges related to code smells, detection, and refactoring techniques and tools. The main challenge identified by the authors relates to the fact that different tools need calibration using the same benchmarked datasets along with the fact that threshold values used are subjective in nature and often arise inconsistencies in obtained results.

Kaur et al. (2021a) carry out a literature survey on the issue of prioritising different code smells belonging to an object-oriented software system. The survey is based on 23 papers collected till May 2020. They conclude that the literature missed out on sufficient automated tool support for automatically prioritising code smells and literature focuses only on a small subset of code smells.

Baqais and Alshayeb (2020) carry out a systematic literature survey by selecting 41 papers obtained after various rigorous analysis steps and snowballing techniques.

The aim is to determine the current status and possibilities in the field of automated refactoring. They conclude that only a few research papers discuss the automatic process of refactoring and search-based refactoring is gaining popularity among researchers due to reduced time and effort at the end of developers.

AlOmar et al. (2021) carry out a systematic literature survey to determine the current situation of the behavioural preservation approach adopted during the process of refactoring. They conclude that behavioural preservation during refactoring is an active open research area and many of the refactoring techniques are still under-researched in reference to behaviour preservation.

Al-Shaaby et al. (2020) carry out a systematic literature survey to identify the feasibility of machine learning algorithms in the field of code smell detection. They concluded that a total of 27 different code smells were targeted using 16 different machine-learning algorithms in the literature.

Mumtaz et al. (2019) carry out a systematic literature survey to identify various bad smells detection techniques related to the UML model. They also propose a framework for evaluating and comparing such bad smell detection approaches. The proposed framework works in two phases. In the first phase, different techniques are evaluated based on factors such as investigated UML model, used detection mechanism, and set of identified bad smells. The second phase deals with exploring experimental designs adopted by different researchers. They conclude that class diagrams are the most explored and validated UML models in the literature.

Caram et al. (2019) carry out another in-depth systematic literature survey to determine the role of machine learning techniques in respect of identifying different code smells. The study identifies:

1 various code smells that are targeted using machine learning approaches

2 a set of machine learning techniques suitable for code smell detection

3 the most suitable machine learning approach for enhancing accuracy during code smell detection.

They conclude that different machine learning techniques used for code smell detection are difficult to compare with ease because of heterogeneity in used datasets and presented results. The authors further recommend empirical investigation on standard datasets in order to improve the reliability and replicability of the studies. Similarly, the authors in (Azeem et al., 2019) give an overview and provides possible usage of machine learning techniques for code smell identification by carrying out a systematic literature survey on 15 papers selected from 2000 to 2017 duration. They conclude that machine learning techniques require ample performance improvements when applied to code smell detection.

The authors in (Sobrinho, 2018) carry out an extensive in-depth extensive systematic literature survey on bad smells from 1990 to 2017 period. They carryout investigation on five aspects (5 W's), namely

1 *which-* of the bad smells are studied more than others and the nature of inter-relatedness between them (if any)

2 *when- a* perspective of different researchers towards various bad smells with reference to time

3 *what*- techniques and experimental setups used in literature for bad smells

4 *who*- the list of researchers who actively and regularly worked on the problem of code bad smells

5 *where*-deals with the geographical location of the researcher and/or community engaged with bad smell.

Pereira et al. (2021) carryout a systematic literature survey on 102 publications selected during 2002–2019 that aims twofold. Firstly, they identified the main tools and techniques presented for code smell. Secondly, visual support to handle code smells is analysed. They conclude that literature has diversity in terms of detected code smells and used programming languages for evaluation; subjectivity exists for code smells in terms of their definition and detection approaches, and lack of visual techniques for validation and oracles to facilitate replication of the studies.

AbuHassan et al. (2021) carry out another systematic literature survey on 145 primary studies. This study aims at analysing existing code smell detection techniques in terms of used metrics, their implementation style, and used validation approaches.

Mariani and Vergilio (2017) carry out a systematic literature survey using 71 primary studies aiming at presenting search-based refactoring approaches proposed in literature along with identifying common characteristics and research trends.

Dwivedi and Satapathy (2020) utilised software metrics to recover reusable documents using neural network models and mining pattern retrieval approaches.

Kaur and Sikka (2022) proposed an approach to create enriched MDG (Module Dependency Graph) by using various weighted code dependencies.

Sehgal et al. (2022) carried out an investigation on 20 projects taken from a public repository (GitHub) to study refactoring using JDeodorant. They conclude that applying one kind of refactoring sometimes results in the introduction of another kind of code smell.

3 Research methodology

The systematic literature review process consists of various key steps that are carried out sequentially, namely constructing goals and identifying Research Questions (RQs), defining which databases will be used during searching, collecting data along with information used for the inclusion and exclusion of the data, and analysing the data along with providing conclusion of the study.

Figure 1 diagrammatically represents these steps carried out during the systematic literature review process. This systematic literature review seeks to identify the gaps left by prior studies. This section of the paper gives details about these key steps. The methodology adopted in this paper is inspired by an evidence-based systematic literature review reporting approach known as Preferred Reporting Items for Systematic Reviews and Meta-Analysis (PRISMA) (Liberati, 2009). PRISMA consists of 27 items checklist that helps in planning carefully and consists of four main phases as shown in Figure 2 that together ensures transparent and complete reporting of systematic literature review. Research papers that fulfil the inclusion criteria of PRISMA are only considered in this systematic literature review. The total number of research papers at different stages of analysis is also depicted in Figure 2. Further, using systematic and explicit methods, this systematic review inspects clearly defined questions, selecting, critically evaluating, and collecting data from the studies included in their analysis.

Figure 1 Systematic literature review steps

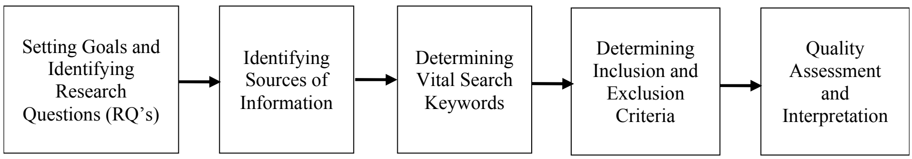
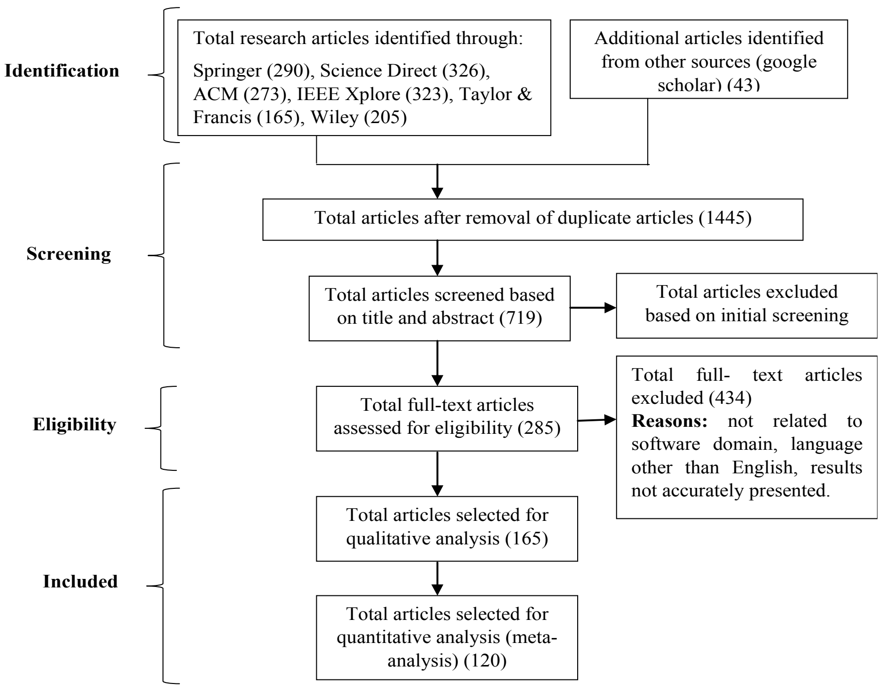


Figure 2 Information flow through different phases of PRISMA for the current study



3.1 Setting goals and identifying research questions

The systematic literature review carried out in this paper related to the topics of code smell and refactoring is designed to uncover the existing vast literature already available on various aspects, namely refactoring, code smells, datasets used and/or available, object-oriented design and refactoring, detection of code smells or antipatterns, as well as analysing different techniques that are used to detect code smells. To conduct a systematic review in this paper, the following research questions are framed and answered in this paper in order to elaborate literature analysis:

RQ1: *What are the different techniques adopted by researchers for identifying code smells?*

The goal is to present a comprehensive list of the main code smell detection techniques. This list enables researchers and practitioners to select the one that is most appropriate for their daily activities while highlighting those that need to be investigated further in the future.

RQ2: *What are different refactoring techniques used by researchers to mitigate code smells?*

The goal is to present a comprehensive list of the main refactoring techniques. This list enables researchers and practitioners to select the one that is most appropriate for their daily activities while highlighting those that need to be investigated further in the future.

RQ3: *What are various tools proposed to handle code smell and refactoring support?*

The goal is to identify semi-/fully- automated tools and/or frameworks that can provide support during code smell and refactoring.

RQ4: *What types of code smells are mainly tackled in literature?*

The aim is to compare different techniques and to identify bad smells that affect most during the degradation of software quality.

RQ5: *What are different standard code smell datasets available?*

The aim is to identify any standard dataset that can promote reproducibility and/or validation in future research.

RQ6: *Which software systems are mainly used during the empirical evaluation?*

The aim is to identify a set of software systems that are mostly used by the majority of researchers for the empirical evaluation of the proposed approach. This will help the researchers in standardising their future techniques related to code smell and refactoring.

3.2 Identifying sources of information

Systematic reviews must have a broader perspective in order to be implemented. To begin the systematic literature review, suitable databases must be selected that can produce appropriate results based on pertinent keywords. The considered databases in this paper are Springer,[1] ScienceDirect,[2] ACM Digital Library,[3] IEEE Xplore,[4] Wiley Online Library,[5] Google Scholar,[6] and Taylor & Francis.[7] The reason behind considering these digital databases is that they all together cover leading journals and conferences publication related to software engineering, and the evolution, development, quality, and maintenance of a software system. On these digital platforms, the following types of documents are considered for review:

a review papers

b conference proceedings

c published technical reports

d thesis

e bookchapters.

3.3 Vital keywords for search

Keywords play an important role while searching for research papers. They help in reducing efforts devoted by a researcher along with saving considerable time devoted during research paper's searching by restricting output produced. Therefore, in searching the databases, we looked for the specific set of keywords in all primary and supplementary databases. Below is a list of the keywords used across the various database sources including Code Smell, Refactoring, Software Maintenance, Antipatterns, Machine Learning, Software Quality Improvement, Object Oriented Design, Meta-Heuristic, and Software Metrics. Further, different logical operators like AND, OR, and NOT are applied to these different independent keywords in order to further enhance and explore search results. The search string used to retrieve various research papers from different digital database platforms is as follows:

> (*Software Refactoring OR Code Refactoring OR Smell Refactoring) OR (Bad Smell OR Code Smell) OR (Anti-pattern OR Design Patterns OR Object Oriented Design) OR (Machine Learning OR Meta Heuristic) AND (Software Metrics OR Metric Suites OR Maintainability OR Software Maintenance OR Software Quality) AND (Tool OR Approach OR Method OR Technique OR Practice OR Problem OR Survey OR Systematic Literature Survey OR Review*)

3.4 Inclusion and exclusion criteria

In this paper, we applied three levels of exclusion criteria to eliminate unrelated papers from the search and analysis criteria adopted. First of all, only papers that are related to computer science and engineering field are included in the search. This is because a few keywords like 'pattern' is multidisciplinary and are commonly found in other branches like biomechanics, medical, nanotechnology, material science, etc. Further, in order to have easy understandability, the papers written in only the English language are considered for evaluation. Moreover, we reviewed research papers available in digital libraries from January 2002 to January 2022 in order to have an extensive literature survey. The papers that cover and explain at least one of the research questions considered here are included. Moreover, to make the search stable and more accurate, we discard duplicate research papers from different libraries as part of PRISMA guidelines (indicated in Figure 2). It is also taken into account that the number of subsequent research papers published by the same authors with some changes and/or extensions is considered for evaluation. Likewise, a paper that has been published in a premier journal after being presented as part of conference proceedings is also considered. The considered exclusion criteria in this paper are

1 studies not directly related to code smell and refactoring

2 papers not fully explored such as short papers, editorial papers, and poster presentations

3 papers presented and published outside the considered time scale.

3.5 Research quality evaluation approach

Once the inclusion-exclusion criteria are set, the quality of the systematic literature survey is assessed. In order to evaluate the quality of considered research papers, an expert team of 10 professionals is constituted. The expert team is comprised of domain experts at the professor level, research scholars, and post-graduate level students. Moreover, standard guidelines as proposed by Kitchenham et al.(2009) and Brereton et al. (2007) are followed in order to carry out a quality-centric systematic literature survey. During a systematic literature survey, in the first instance, the expert team carries out scrutiny of downloaded research papers' using a three-stage approach. In the first stage, all research papers are filtered based on the relevancy and appropriateness of the title with the domain considered for evaluation in this paper. During this process, domain experts help the research scholars and students to carry out quality-centric filtering of papers. This stage results in filtering about 60% of total downloaded papers and the remaining 40% are further considered for evaluation in the next phase. During the second phase, the abstract of all the remaining papers is carefully examined in order to further discard the irrelevant papers. This phase results in 17% of total downloaded papers being considered for evaluation. Finally, in the third stage, the remaining 17% of research papers are fully explored in order to carry out a systematic literature survey.

Table 1 summarises papers collected after rigorous analysis of the three-step process divided according to considered keywords and finally, these papers are considered for carrying out the formulated investigation (in terms of different RQs) in this paper. Figure 3 depicts a detailed description of the share of different publications by their venue considered as primary studies to carry out a systematic literature survey in this paper. A detailed description of all these considered primary studies is provided in Appendix1 in this paper and provided details include the unique number assigned to the paper (ID), corresponding title (TITLE), authors' list (AUTHORS), publication year (YEAR), paper type (Journal/Conference), and source of publication (SOURCE). Figure 4 depicts the year-wise distribution of different research papers considered to carry out a systematic literature survey in this paper.

4 Results and discussions

This section of the paper gives details about obtained results acquired after systematically analysing selected key research papers. As a researcher, it is important to know key Journals and/or Conferences related to the field of code smell and refactoring.

Table 1 Summary of research papers considered for systematic literature survey

S. No.	*E-resource library*	*Search keyword*	*Search duration*	*No. of research papers*	*Research category*
1	Springer, ScienceDirect, ACM, IEEE Xplore, Taylor and Francis, Wiley Online Library, Google Scholar	Code Smell	January2002 to January2022	43	Research Papers, Review Papers, Book Chapters, Conferences
2		Refactoring		26	
3		Software Quality Improvement and Refactoring		10	
4		Antipatterns		9	
5		Machine Learning		14	
6		Software Maintenance and Refactoring		17	
7		Meta Heuristics and CodeSmell		13	
8		Software Metric and Code Smell		24	
9		Object-Oriented Design and Refactoring		8	
10		Code Smell and Refactoring		55	
11		Code Smell and Machine Learning		17	
12		Code Smell and Refactoring and Machine Learning		13	
13		Software Quality and Code Smell		24	
14		Software Maintenance and Refactoring		7	
Total Research Papers Considered for Systematic Literature Survey				**280**	

Figure5provides details of Top-5 Journals that are actively involved with publishing research work related to code smell and refactoring. *IEEE Transaction on Software Engineering* (abbreviated *IEEE Trans. Softw. Eng.*), Journal of Systems and Software (abbreviated *J. Syst. Softw.*), *Information and Software Technology* (abbreviated *Inf. Softw. Technol.*), *Journal of Software: Evolution and Process* (abbreviated *J. Softw.: Evol. Process*), and *Empirical Software Engineering* (abbreviated *Empir. Softw. Eng.*) are among Top-5 leading journal's list selected by researchers in past. Similarly, Figure 6 shows details about reputed conferences that are regularly engaged with handling code smell and refactoring problems. The reputed conferences/workshops in the field of code smell and refactoring include ICSM (IEEE International Conference on Software

Maintenance), ICSE (ACM/IEEE International Conference on Software Engineering), CSMR (IEEE European Conference on Software Maintenance and Reengineering), ICPC (IEEE/ACM International Conference on Program Comprehension), ESEM (ACM/IEEE International Symposium on Empirical Software Engineering and Measurement), WRT (ACMWorkshop on Refactoring Tools), and SANER (IEEE International Conference on Software Analysis, Evolution, and Reengineering). It is our strong belief that this knowledge is helpful in systematically guiding different researchers working in the direction of code smell and refactoring. Moreover, a thorough discussion and interpretation of acquired results are also provided in this section. The acquired results are presented as the answers to different research questions formulated earlier in this paper.

Figure 3 Different types of publications selected to perform systematic literature survey (see online version for colours)

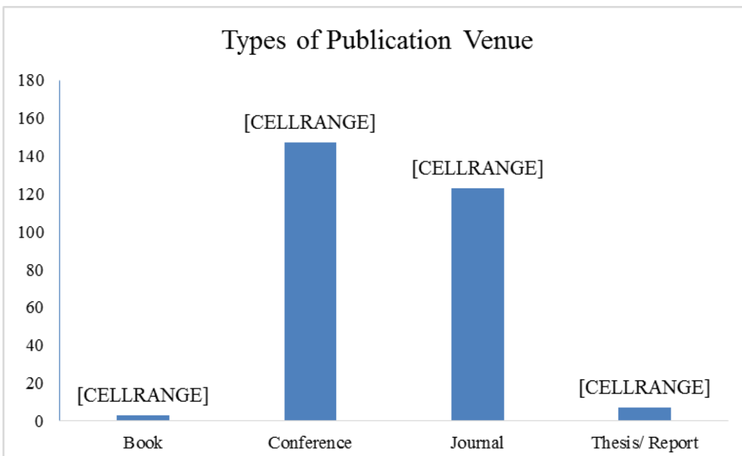


Figure 4 Year-wise distribution of different research papers considered for evaluation (see online version for colours)

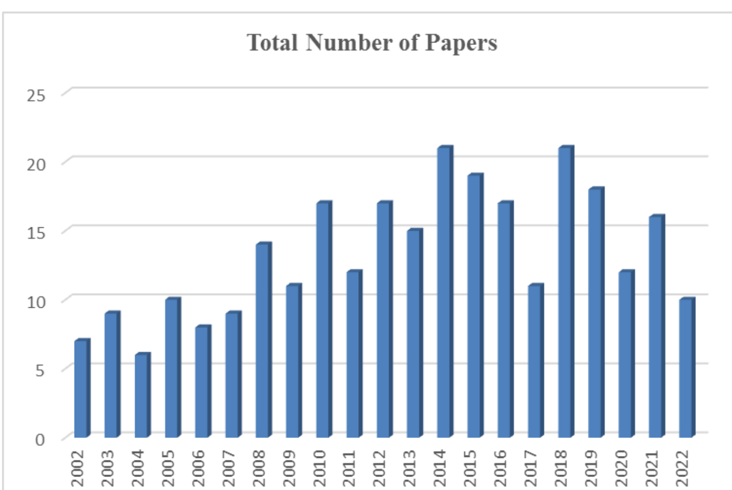


Figure 5 Top-5 journals related to code smell and refactoring (see online version for colours)

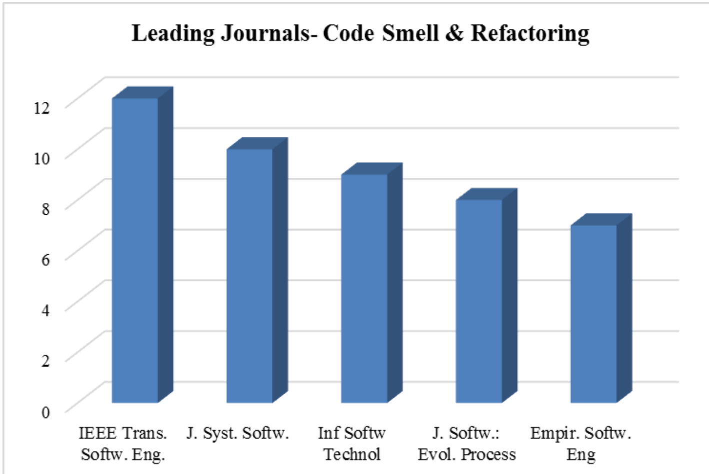
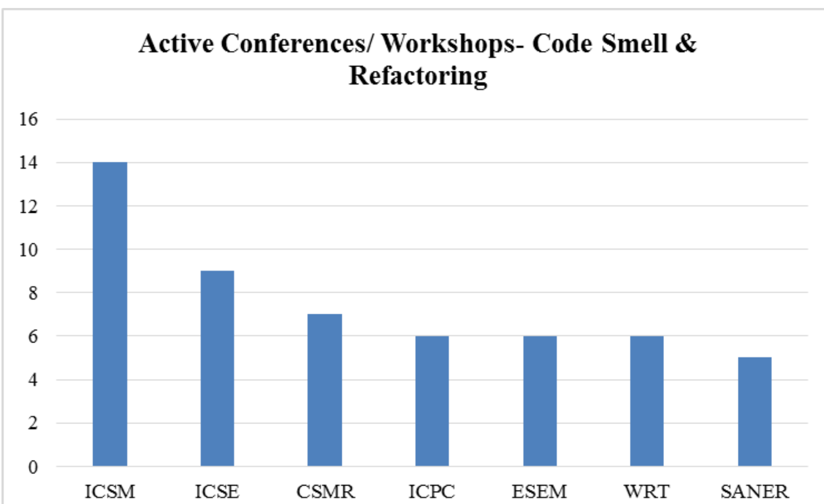


Figure 6 Leading conferences/Workshops engaged with code smell and refactoring (see online version for colours)



RQ1: *What are different techniques adopted by researchers for identifying code smells?*

A code smell is an active research area considered by various researchers, and different techniques/approaches are proposed and/or tested by researchers in the literature. Table 2 depicts the classification of different code smell detection techniques adopted by different researchers in the literature. The third column in the table depicts the total number of reference papers available that utilise the corresponding code smell detection technique mentioned in column two. Finally, the last column gives references to different programming languages that are used by different researchers to validate their approaches using the technique depicted in column two. Typically, Metric-Based (MB) approaches are used by researchers in the literature that utilises different source code metrics to capture different types of code smells based on the unique characteristics measured using code metrics. This method generally involves utilising various third-party tools that convert the underlying source code of the software into an Abstract Syntax Tree (AST) representation. This AST is later utilised to measure different characteristics of a code smell based using a code metric and a threshold value. As clearly noticeable from Table 2,the MB approach is widely and openly adopted by different researchers to discover different types of code smells. The machine learning based (MLB) approaches typically involves preparing a mathematical model that represents the code smell detection problem followed by the application of a supervised or unsupervised machine learning algorithm on the prepared mathematical model to identify the underlying set of code smells. Preparing the mathematical model step is dependent on identifying two types of variables for the studied system, namely, dependent and independent variables. The machine-learning algorithm explores various independent variables to predict the corresponding value of the dependent variable of the system. Moreover, the MLB approach's success is highly dependent on the availability of a quality large amount of data, which is derived from the underlying software system and is used to train the prepared mathematical model. The MLB approach is a recent trend that is gaining popularity among the research community and is clearly observed in Table 2. The change history based (CHB) approach is dependent on evolutionary information available for a software system that denotes how software undergoes modifications over a period of time. The evolutionary information is utilised using association-based rule mining to identify various sets of code smells present in a software system. In literature, this field is least elaborated on as compared to the rest of the other alternatives as depicted in Table 2. The Heuristics Based (HB) code smell detection techniques are based on formulating heuristics to target a particular code smell. The heuristics include the use of different code metrics and combining them under special detection rules which are specific to a particular type of code smell (here, rules are generally in the form of threshold values that are computed through empirical means). Sharma and Spinellis, 2018) are of the opinion that not every code smell can be detected alone by using code metrics. However, they are of the strong opinion that different metrics need to be combined under special circumstances (termed heuristics) in order to improve the detection accuracy of the code smell detection approach. Various approaches in the optimisation based (OB) category focus on the use of various optimisation algorithms to identify a set of code smells. In literature, various optimisation algorithms belong to the categories of genetic algorithm (GA), Harmony Search (HS), particle swarm optimisation (PSO), artificial bees colony (ABC), ant colony optimisation (ACO), simulated annealing (SA), etc. Moreover, these optimisation algorithms are applied to two categories of data in literature namely, computed software metrics, and/or existing code smells examples belonging to a software system.

RQ2: *What are different refactoring techniques used by researchers to mitigate code smells?*

The process of refactoring involves changing the design of a system without changing its behaviour and is aimed at improving the underlying quality of the software product. In the field of refactoring, significant work has emerged since 2001 with regard to code smells. It involves reorganising variables, classes, and methods of software so that it enables easy future adaptations and comprehensions. In literature, refactoring is applied to two types of software artefacts namely model and source code. Most of the refactoring is applied and proposed for source code artefacts (85.76%) and only 14.24% of the selected studies target model-based refactoring. Out of various refactoring operations applied to source code artefacts, mainly targets object-oriented programming languages (mainly Java, maybe due to its wide popularity and major market share in software development). Programming languages like C++ (Tsantalisand Chatzigeorgiou, 2009), Fortran, AspectJ (Noguera et al., 2012; Mongiovi et al., 2014), Erlang (Horpácsi et al., 2017), Smalltalk (Gómez, 2012) and XML (Noguera et al., 2012) are targeted by only a few researchers in the past decade. Model-based refactoring is primarily proposed for UML and Alloy specification language is targeted only (Massoni et al., 2008). There are three most common criteria adopted to find various refactoring capabilities in a software system, namely

1 quality metrics-based
2 pre-conditions-based
3 clustering-based.

Quality metrics-based refactoring opportunities aim at applying various cohesion, distance (similarity) among software elements, and coupling code metrics. Code smells such as feature envy and code clones utilise pre-conditions-based refactoring opportunities that involve testing a condition before applying the corresponding refactoring. Finally, clustering-based refactoring opportunities are based on grouping different code elements such as lines, methods, fields, classes, etc. in order to identify extract, and/or move refactoring actions.

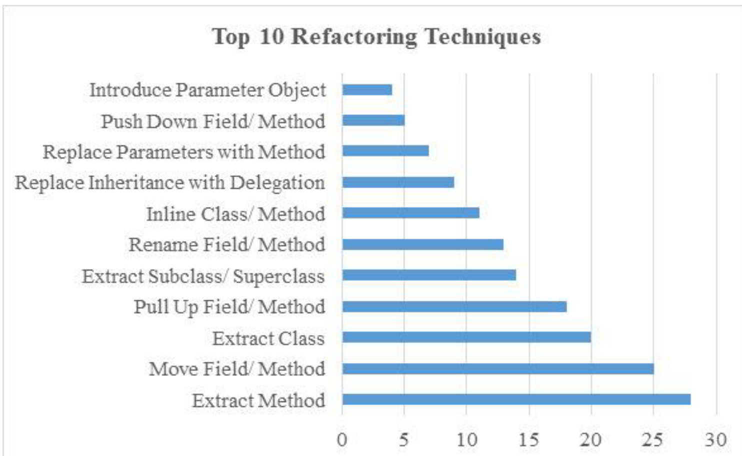
Table 2 Classification of code smell detection methods/techniques

S. no	*Code smell detection method*	*No. of papers (percentage)*	*Languages/artefacts references*
1	Metric-Based (MB)	38 (8.68%)	Java (Vidal et al., 2016), C++ (Marinescu, 2005, September), UML Diagrams (Fourati et al., 2011), Aspect-oriented Systems (Macia Bertran et al., 2011), JavaScript (Vidal et al., 2015), C (Fenske et al., 2015)
2	Machine learning based (MLB)	22 (5.02%)	Java (Fu and Shen, 2015)
3	Change History Based (CHB)	9 (2.05%)	Java (Palomba et al., 2014), C (Rama, 2010, February), C++ (Abebe et al., 2011), UML Models (Arcelli, 2015), REST APIs (Palma et al., 2014)
4	Rule/Heuristics Based (HB)	29 (6.62%)	C# (Sharmaand Spinellis, 2018)
5	Optimisation-Based (OB)	12 (2.74%)	Java (Ghannem, 2016), XML (Ouni et al., 2015)

Almost all of the refactoring that is described in the literature mirrors the definitions provided by Fowler and Beck (2018). However, extract and move refactoring are the most cited/used refactoring techniques in literature and it is evident from Figure 7which depicts the top 10 refactoring techniques targeted by different researchers. In the software industry, these techniques are likely to play a significant role due to their high interest.

Developers often find it difficult to know what kind of refactoring technique must be applied in the underlying software system in order to fix a problem by identifying refactoring opportunities (Marianiand Vergilio, 2017). There is no one-to-one relationship between identified code smells and the corresponding refactoring applied in order to mitigate code smells. In general, it is possible to use more than one refactoring technique on the same smell. Thus, refactoring is an open research area that needs further investigation for the benefit of researchers and industries.

Figure 7 Top-10 targeted refactoring techniques in literature (see online version for colours)



RQ3:*What are various tools* proposed *to handle code smell and refactoring support?*

Tool support is always handy for developers and research experts and helps in reducing maintenance efforts, cost, devoted time, and chances of manual errors. This RQ aims at investigating which semi-/automatic tool support is available to perform code smell detection and mitigation using refactoring techniques. Moreover, we also investigated the platforms/languages for which different tools are proposed by various researchers in the literature.

There are different automated/semi-automated tools available in the literature that can be used to reveal code smells and perform corresponding refactoring operations. All these tools differ from each other in several respects including language supported, number and type of code smells supported, and no/partial/full refactoring support. Table 3 summarises different tools proposed by various researchers in the literature that are capable of code smell detection and/or refactoring. The tool's summary includes information namely its name, availability nature of the tool, language supported, refactoring capability, download link/reference, and list of code smell that are supported by the corresponding tool. Out of these different tools, CCFinder (Hermans et al., 2016; Lacerda et al., 2020; Liu et al.,2015; Gupta and Suri, 2018; Geiger et al., 2006; Bavota et al., 2012; Liu et al., 2018), DÉCOR (Pecorelli et al., 2019; De Stefano et al., 2020; Fontana et al., 2011; Zhang et al., 2022; Kaur and Dhiman, 2019; Zhu et al., 2018; Boutaib et al., 2021; Zhang et al., 2022; Santos and Petronilo, 2022), inCode (Hamid et al., 2013; Saranya et al., 2018; Kaur and Dhiman, 2019; Fontana et al., 2015; Yamashita and Moonen, 2013), PMD (Rasool and Arshad, 2015; Paiva et al., 2017; Rani and Chhabra, 2017; Fontana et al., 2012; Lenhard et al., 2017; Elkhail and Cerny, 2019; Soomlek et al., 2021; Rahad et al., 2021) and InFusion (Masmali and Badreddin, 2021; Cairo et al., 2018; Caram et al., 2019; Paiva et al., 2017; Fontana et al., 2012; Fernandes et al., 2016; Mannan et al., 2016) are the most quoted and cited tools available in the literature. Different detection tools for code smells use metrics or ad-hoc rules for identifying patterns in the underlying source code of a software system, at the price of some loss in accuracy. Moreover, according to information available in the literature, authors of these tools have conducted their tool's experimentation/validation on a generally distinct set of datasets. Thus, in the case of the unavailability of these tools, comparisons between their results cannot be made in general. Hence, the accuracy of different code smell tools is a key and open research question for researchers.

RQ4: What types of code smells are mainly tackled in literature?

Categorising smells based on possible relationships between them is an interesting approach to understanding smells and it aims at improving understandability. In literature, different types of code smell and their different taxonomies are proposed by (Wake, 2004; Becker et al., 1999; Mantyla et al., 2003; Kerievsky, 2005; Brown, 1998). Brown et al.(1998) propose 40 antipatterns for 7 types of common problems (code smells) that result in negative consequences in the future. These code smells are a blob, poltergeist, lava flow, cut and paste programming, functional decomposition, Swiss army knife, and spaghetti code. The blob problem is a situation where one object is given too many responsibilities while other objects are doing only simple activities in the system. Poltergeist is a situation where a class is having very small functionality and a short life cycle with respect to the whole software system. The lava flow is related to the design that has been frozen with dead code and forgotten information. Cut and paste programming is the coding style where the developers extensively use copies of a code fragment. Functional decomposition is related to the object-oriented programming style and is a situation where experts break the responsibilities of a single class into the form of several classes. Fowler and Beck (2018) propose 26 types of code smells, namely divergent change, long method, long parameter list, duplicated code, large class, data clumps, shotgun surgery, feature envy, switch statements, primitive obsession, speculative generality, lazy class, parallel inheritance hierarchy, middle man, temporary field, message chains, data class, an alternative class with different interfaces, inappropriate intimacy, incomplete class library, mysterious names, comments, global data, refused bequest, lazy element, insider trading, and mutable data. The author in (Wake, 2004) proposes 8 different code smells that directly affect the understandability and maintainability of the software, namely magic numbers, type embedded in the name, inconsistent names, null check, uncommunicative names, dead code, special case, and complicated Boolean expression. Further, they categorise different code smells as

1 smells within classes

2 smells between classes based on the number of classes involved in the degraded quality of the software system.

Table 3 Code smells detection and refactoring tools

S. no.	Tool name	Availability	Language supports	Refactoring	Tool link/reference	Supported code smells
1	Stench Blossom	Open Source (Eclipse Plug-in)	Java	No	http://multiview.cs.pdx.edu/refactoring/smells/OR https://github.com/DeveloperLiberationFront/refactoring tools	Feature Envy, Long Method, Data Clumps, Large Class
2	Weka Nose	Open Source	Java	No	https://github.com/uazadi/WekaNose	Data Class, Feature Envy, and God Class using Machine Learning Algorithms
3	SACSEA	–	Java	No	Peters and Zaidman (2012)	God Class, Feature Envy, Data Class, Message Chain Class, Long Parameter List
4	LBS Detectors	–	Java	No	Abebe et al.(2011)	Produces lexicon bad smells
5	JCode Canine	–	Java	No	Maruyamaand Tokoda (2008)	Duplicated Code, Data Class, Switch Statement, and Feature Envy
6	BSDT	–	Java	No	Danphitsanuphanand Suwantada (2012)	Large Class, Data Class, Lazy Class, Parallel Inheritance Hierarchies
7	JDEv	–	Java	No	Lakshmanan and Manikandan (2014)	Duplicated Code, Long Method, Large Class, Long Parameter List
8	iplasma	Research Prototype	Java, C++	No	http://loose.cs.upt.ro/index.php?n = Main. IPlasma	God class, Data class, Refused parent bequest, Feature envy
9	FindBug	–	Java	No	Mittal et al.(2011)	Error Collection in source code
10	PMD	Open Source	Java	No	https://pmd.github.io/	Identify primary problems in code like Dead Code, God Class, Long Parameter List, etc.
11	JDeodorant	Open Source (Eclipse Plug-in)	Java	Yes	https://github.com/tsantalis/JDeodorant	God Class, Type Check, Feature Envy, Long Method

Table 3 Code smells detection and refactoring tools (continued)

S. no.	*Tool name*	*Availability*	*Language supports*	*Refactoring*	*Tool link/reference*	*Supported code smells*
12	DÉCOR	Commercial	Java	No	http://www.ptidej.net/research/designsmells/	Refused Bequest, Large Class, Lazy Class, Long Parameter List, Long Method, Feature envy, Message Chains, Shotgun Surgery, Duplicated Code, Data Class, Divergent change, and Speculative Generality
13	PRODEOOS	Research Prototype	Java, C++	No		
14	InFusion	Commercial	Java, C, C++	No	http://www.intooitus.com/inFusion.html	Duplicated Code, Feature Envy, God Class
15	InCode	Commercial	Java, C, C++	No	https://marketplace.eclipse.org/content/incode-helium	Large Class, Refused Bequest, Data Clumps, Shotgun Surgery, Duplicated Code, Divergent Change, Feature Envy, Refused Bequest, Long Method
16	CheckStyle	Open Source	Java	No	https://github.com/checkstyle	Long Method, Large Class, Long Parameter List, Duplicated Code
17	JSNOSE	Open Source	Java Script	No	https://github.com/crystalwm/jsnose	Switch Statement, Dead Code, Excessive Global Variables, Message Chain, Long Method, Empty Catch
18	CCFinder	Open Source	Java, VB, C#, C/C++, COBOL	No	http://www.ccfinder.net/index.html	Duplicated Code

Mantyla et al. (2003) provide a taxonomy and classify different code smells proposed by Becker et al. (2018) into the following categories:

1 bloaters

2 object-oriented abusers

3 change preventers

4 couplers

5 dispensable.

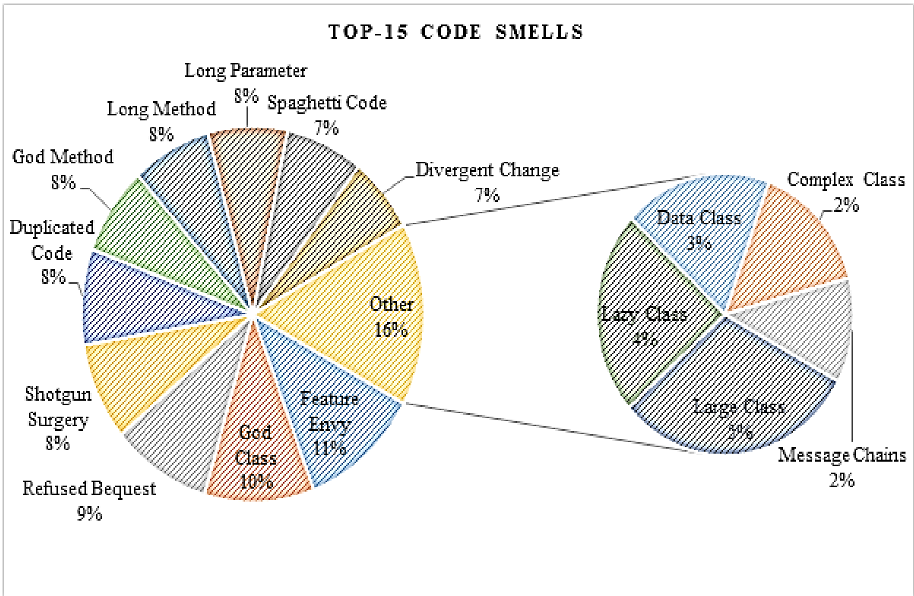
Table 4 explains this taxonomy in detail. Kerievsky (2005) proposes five types of new code smells that affect quality, namely, oddball solution, solution sprawl, conditional complexity, combinatorial explosion, and indecent exposure. An oddball solution is a situation where two or more solutions are provided in the source code for the same given problem. Solution sprawl is a situation where changes to one part of the system cause progressive changes to several other parts of the software. Conditional complexity is associated with the exaggerated use of conditional structures within the software system. Combinatorial explosion is another situation where the same functionality is being called many times through code snippets but with different types of objects involved in a software system. Indecent exposure is related to the increased complexity of the system and is identified with the high degree of access made by clients to various classes of the software system.

Table 4 Code smells taxonomy by Mantyla (2003)

S. no.	*Code smell taxonomy*	*Explanation*	*Code smells examples*
1	Bloaters	Affects understandability as well as modifiability and is generally identified as large-sized software elements which are difficult to handle	Data clamps, large class, long parameter list, long method, primitive obsession
2	Object-oriented Abusers	Situations where object-oriented design principles as proposed in Martin (2005) are compromised by providing workaround solutions in the code	Switch statement, parallel inheritance hierarchies, refused bequest, temporary field, classes with different interfaces
3	Change preventers	Software elements with a complicated structure that prevents a future modification	Shotgun surgery, divergent change
4	Couplers	Software elements with a high degree of interdependencies with each other	Inappropriate intimacy and feature envy
5	Dispensable	Smells that should not be present and, therefore, can be removed	Lazy class, speculative generality, duplicated code, data class

After discussing various code smells and their classification, it is necessary to determine key code smells that are actively targeted in literature by different researchers during their study. Based on the literature survey, it is determined that code smells as suggested by authors in Fowler and Beck (2018) are mainly targeted whereas the rest of the other code smells are given very little attention. Figure 8 shows Top-15 code smells that are majorly targeted in the literature using pie chart representation.

Figure 8 List of top-15 code smells that are actively targeted in literature (see online version for colours)



RQ5: *What are different standard code smell datasets available?*

Based on the literature review, we found that the literature lacks large datasets of code smell detection and refactoring. Thus, hindering the experimental validation of the approaches and/or tools available in the literature. However, only a few of the researchers tried in this direction to fill this research gap by providing a standard set of datasets for code smells available for experimental purposes. Guggulothu (2020) provided a standard multilabel dataset[8] that provides method-level code smells namely long method and feature envy and it contains tested code smell information of 74 software systems. The dataset is carefully prepared and tested using machine learning approaches. Further, the authors in Arcelli Fontana (2016) provided a dataset[9] of four code smells namely data class, feature envy, god class, and long method. This dataset is divided into two categories, namely class- and method-level. The proposed dataset is tested using 32 different machine learning classifiers including J48, Naïve Bayes, JRip, and Random Forest.

RQ6: *Which software systems are mainly used during the empirical evaluation?*

This research question aims to identify various software systems that are used by the majority of researchers for their study. Moreover, it guides and provides symmetry to future researchers during the empirical evaluation of their proposed code smell detection and mitigation approach. A number of experiments were conducted on different subject systems for the purpose of detecting code smells, and Table 5 depicts such Top-10 subject systems that are most commonly used by different researchers in past for their evaluation and testing purposes.

Table 5 Top-10 subject systems used most commonly for empirical evaluation purposes

S. no	*Subject system name*	*No. of references*	*Description*	*Link*
1	JUnit	15	A testing framework for Java Language	https://github.com/junit-team
2	JHotDraw	18	A two-dimensional graphics framework	https://sourceforge.net/projects/jhotdraw/
3	WEKA	10	A data mining tool that provides machine learning capabilities	https://github.com/Waikato/weka-trunk
4	Azureus	8	A BitTorrent client capable of transferring data via the BitTorrent protocol	http://qualitascorpus.com/docs/catalogue/20130901/corpus_catalogue.html
5	Apache Tomcat	21	An open-source implementation ofthe Jakarta EE platform	https://tomcat.apache.org/
6	ArgoUML	23	An open-source UML modelling tool	https://github.com/argouml-tigris-org/argouml
7	Eclipse	6	An IDE used for computer programming	https://github.com/eclipse
8	Xerces	15	An XML parser	https://xerces.apache.org/mirrors.cgi
9	Apache Ant	19	A Java library and command-line tool that helps build software	https://ant.apache.org/srcdownload.cgi
10	JEdit	22	A mature programmer's text editor	https://github.com/albfan/jEdit

The majority of authors detected code smells in Java-based open-source software applications and projects. Few of them have used in-house small software systems for experimental evaluation. Moreover, only four references (Sahin et al., 2014; Marinescu, 2005; Janckeand Speicher, 2010; Munro, 2005) utilised commercial and industry-standard software systems for their experimental evaluation. The datasets listed in Table 5cannot be found commonly in any of the studied publications that carry out their experimentation on exactly the same system as the one mentioned in Table 5. However, for evaluating various code smell and/or refactoring techniques and tools, it is necessary to have common case studies so as to generalise their published results.

5 Threats to validity

An attempt to mitigate some of the threats to validity is discussed in this section of the paper. Firstly, the search string is the key to carrying out a systematic literature survey, therefore, we used different synonyms of search strings so as to include every possible relevant literature in this study. Secondly, the choice of considered electronic databases is another key factor in the accuracy of the results presented in this paper. In order to reduce this threat, we carry out a snowballing process to find more studies that may be relevant. We then review references in the selected papers to identify other studies that may be relevant that were not initially included in our search. Thirdly, the selection of considered primary studies is another threat to the validity of the results presented in this paper. Therefore, in order to reduce any biasing during inclusion and exclusion criteria, different studies are reviewed multiple times by different experienced authors who have well-knowledge in the field of software engineering, code smell, and refactoring. Lastly, the language choice of different considered primary studies is another threat to validity. However, studies written in the English language are preferred in this study in order to cover a larger audience due to the wider popularity of this language.

6 Conclusion and future work

As a tertiary systematic literature study carried out in this paper, we focused on refactoring and code smells. Code smells and refactoring challenges and observations were systematically evaluated by systematically analysing 280 primary studies selected between January 2002 to January 2022 after rigorous analysis. Six RQs are formulated and answered in this paper that reflect the main challenges and observations needed necessarily for the research community engaged with code smell and refactoring. Our study revealed that both code smell and refactoring are directly affected by the same quality attributes and they have a direct relationship with the functionality, understandability, complexity, and maintainability of the software system. As a part of the investigation, we identified the top 5 journals (Figure 5) that represent the quality-centric source for research knowledge and future publications. Similarly, leading conferences and workshops are also identified (Figure 6) for the same purpose. Further, a taxonomy for techniques available in the literature for code smell identification is also presented. Metric-based and Heuristic-based approaches dominate the literature. Extract method and move field/method are among the top refactoring techniques recommended in the literature. Furthermore, state-of-art code smell detection and refactoring tools are identified in this paper. A taxonomy of code smells along with a list of top-15 code smells are also presented in this paper. We identified that feature envy and god class are among the top code smells that are mostly studied in the literature. Finally, a set of datasets available in the literature are identified along with a list of top-10 software systems is also presented that are mostly used in literature for validating proposed approaches. We still have several open questions about the relationship between code smell and refactoring. For instance, the choice of a refactoring action for a specific code smell is still an open issue and needs further investigation. It is hoped that this study can inspire researchers to investigate a deeper level of mitigation of code smells and the tools used to mitigate them as well as evaluate their impact on quality. Research in the future should address a number of open issues identified in our analysis. The first open issue is related to code smell nomenclature and a more effective approach to deal with code smell identification and refactoring. Secondly, researchers should explore opportunities related to code smell detection and refactoring tool support. Thirdly, developers' knowledge should be explored further to minimise refactoring efforts. Fourthly, the relationship between different applied refactoring operations underlying quality metrics such as complexity, coupling, etc. needs to be investigated further. Finally, but not least, the literature seriously lacks reliable datasets that are necessary to fast-track validation and reproducibility. The available datasets lack large-scale academic and industry-standard projects.

It is hoped that this study can inspire researchers to investigate a deeper level of mitigation of code smells and the tools used to mitigate them as well as evaluate their impact on quality. Research in the future should address a number of open issues identified in our analysis. The results of our study will provide a basis for future research and will guide researchers to produce more high quality research in this area, as a result of the recommendations provided in this report.

References

Abebe, S.L., Haiduc, S., Tonella, P. and Marcus, A. (2011) 'The effect of lexicon bad smells on concept location in source code', *11th International Working Conference on Source Code Analysis and Manipulation,* IEEE, September, Williamsburg, VA, USA, pp.125–134.

Abid, C., Alizadeh, V., Kessentini, M., Ferreira, T.D.N. and Dig, D. (2020) *30 Years of Software Refactoring Research: A Systematic Literature Review*, arXiv preprint arXiv: 2007.02194.

AbuHassan, A., Alshayeb, M. and Ghouti, L. (2021) 'Software smell detection techniques: a systematic literature review', *Journal of Software: Evolution and Process*, Vol. 33, No. 3, pp.2320.

Agnihotri, M. and Chug, A. (2020) 'A systematic literature survey of software metrics, code smells and refactoring techniques', *Journal of Information Processing Systems*, Vol. 16, No. 4, pp.915–934.

AlOmar, E.A., Mkaouer, M.W., Newman, C. and Ouni, A. (2021) 'On preserving the behavior in software refactoring: a systematic mapping study', *Information and Software Technology*, Vol. 140, pp.106675.

Al-Shaaby, A., Aljamaan, H. and Alshayeb, M. (2020) 'Bad smell detection using machine learning techniques: a systematic literature review', *Arabian Journal for Science and Engineering*, Vol. 45, No. 4, pp.2341–2369.

Arass, M.E., Ouazzani-touhami, K. and Souissi, N. (2019) 'The system of systems paradigm to reduce the complexity of data lifecycle management. Case of the security information and event management', *International Journal of System of Systems Engineering*, Vol. 9, No. 4, pp.331–361.

Arcelli Fontana, F., Mäntylä, M.V., Zanoni, M. and Marino, A. (2016) 'Comparing and experimenting machine learning techniques for code smell detection', *Empirical Software Engineering*, Vol. 21, No. 3, pp.1143–1191.

Arcelli, D., Berardinelli, L. and Trubiani, C. (2015) 'Performance antipattern detection through fuml model library', *Proceedings of the 2015 Workshop on Challenges in Performance Methods for Software Development,* January, New York, NY, USA, pp.23–28.

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

Baqais, A.A.B. and Alshayeb, M. (2020) 'Automatic software refactoring: a systematic literature review', *Software Quality Journal*, Vol. 28, No. 2, pp.459–502.

Bavota, G., Qusef, A., Oliveto, R., De Lucia, A. and Binkley, D. (2012) 'An empirical analysis of the distribution of unit test smells and their impact on software maintenance', *2012 28th IEEE International Conference on Software Maintenance (ICSM,*), September, IEEE, Trento, Italy, pp.56–65.

Becker, P., Fowler, M., Beck, K., Brant, J., Opdyke, W. and Roberts, D. (1999) *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Professional, ISBN: 978-0201485677, p.333.

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

Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M. and Khalil, M. (2007) 'Lessons from applying the systematic literature review process within the software engineering domain', *Journal of Systems and Software*, Vol. 80, No. 4, pp.571–583.

Brown, W.H., Malveau, R.C., McCormick, H.W.S. and Mowbray, T.J. (1998) *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, p.336, ISBN: 978-0-471-19713-3.

Cairo, A.S., Carneiro, G.D.F. and Monteiro, M.P. (2018) 'The impact of code smells on software bugs: a systematic literature review', *Information*, Vol. 9, No. 11, p.273.

Caram, F.L., Rodrigues, B.R.D.O., Campanelli, A.S. and Parreiras, F.S. (2019) 'Machine learning techniques for code smells detection: a systematic mapping study', *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, Vol. 29, No. 02, pp.285–316.

Danphitsanuphan, P. and Suwantada, T. (2012) 'Code smell detecting tool and code smell-structure bug relationship', *2012 Spring Congress on Engineering and Technology,* May, IEEE, Xi'an, China, pp.1–5.

de Paulo Sobrinho, E.V., De Lucia, A. and de Almeida Maia, M. (2018) 'A systematic literature review on bad smells–5 w's: which, when, what, who, where', *IEEE Transactions on Software Engineering*, Vol. 47, No. 1, pp.17–66.

De Stefano, M., Gambardella, M.S., Pecorelli, F., Palomba, F. and De Lucia, A. (2020) 'cASpER: a plug-in for automated code smell detection and refactoring', *Proceedings of the International Conference on Advanced Visual Interfaces,* September, New York, NY, USA, pp.1–3.

Dexun, J., Peijun, M., Xiaohong, S. and Tiantian, W. (2013) 'Detection and refactoring of bad smell caused by large scale', *International Journal of Software Engineering and Applications*, Vol. 4, No. 5, p.1.

Dwivedi, A.K. and Satapathy, S.M. (2020) 'Mining patterns in open source software using software metrics and neural network models', *International Journal of System of Systems Engineering*, Vol. 10, No. 4, pp.397–409.

Elkhail, A.A. and Cerny, T. (2019) 'On relating code smells to security vulnerabilities', *2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity),IEEE Intl Conference on High Performance and Smart Computing* (*HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, May, IEEE, Washington, DC, USA, pp.7–12.

Fenske, W., Schulze, S., Meyer, D. and Saake, G. (2015) 'When code smells twice as much: metric-based detection of variability-aware code smells', *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM),* September, IEEE, Bremen, Germany, pp.171–180.

Fernandes (2016, E.O. and) 'A review-based comparative study of bad smell detection tools', *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering* June, New York, NY, USA, pp.1–12.

Fontana, F.A., Braione, P. and Zanoni, M. (2012) 'Automatic detection of bad smells in code: an experimental assessment', *J. Object Technol*, Vol. 11, No. 2, pp.5–1.

Fontana, F.A., Mariani, E., Mornioli, A., Sormani, R. and Tonello, A. (2011) 'An experience report on using code smells detection tools', *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, March, IEEE, Berlin, Germany, pp.450–457.

Fourati, R., Bouassida, N. and Abdallah, H.B. (2011) 'A metric-based approach for anti-pattern detection in UML designs', *Computer and Information Science*, Springer, Berlin, Heidelberg, pp.17–33.

Fowler, M. and Beck, K. (2018) *Refactoring: Improving the Design of Existing Code*, 2nd ed., Addison-Wesley.

Fu, S. and Shen, B. (2015) 'Code bad smell detection through evolutionary data mining', *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM),* October, IEEE, Beijing, China, pp.1–9.

Ghannem, A.E. (2016) 'On the use of design defect examples to detect model refactoring opportunities', *Software Quality Journal*, Vol. 24, No. 4, pp.947–965.

Geiger, R., Fluri, B., Gall, H.C. and Pinzger, M. (2006) 'Relation of code clones and change couplings', *International Conference on Fundamental Approaches to Software Engineering,* March, Springer, Berlin, Heidelberg, pp.411–425.

Gómez et al., V.U., Ducasse, S. and d'Hondt, T. (2012) 'Ring: A unifying meta-model and infrastructure for smalltalk source code analysis tools', *Computer Languages, Systems and Structures*, Vol. 38, No. 1, pp.44–60.

Guggulothu, T. and Moiz, S.A. (2020) 'Code smell detection using multi-label classification approach', *Software Quality Journal*, Vol. 28, No. 3, pp.1063–1086.

Gupta, A. and Suri, B. (2018) 'A survey on code clone, its behavior and applications', *Networking Communication and Data Knowledge Engineering*, Springer, Singapore, pp.27–39.

Hamid, A., Ilyas, M., Hummayun, M. and Nawaz, A. (2013) 'A comparative study on code smell detection tools', *International Journal of Advanced Science and Technology*, Vol. 60, pp.25–32.

Hermans, F. and Aivaloglou, E. (2016) 'Do code smells hamper novice programming? A controlled experiment on scratch programs', *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, May, IEEE, Austin, TX, USA, pp.1–10.

Horpácsi, D., Kőszegi, J. and Horváth, Z. (2017) *Trustworthy Refactoring Via Decomposition and Schemes: A Complex Case Study*, arXiv preprint arXiv: 1708.07225, pp.92–108.

Jancke, S. and Speicher, D. (2010) *Smell Detection in Context*, University of Bonn, p.113.

Kaur, A. (2020) 'A systematic literature review on empirical analysis of the relationship between code smells and software quality attributes', *Archives of Computational Methods in Engineering*, Vol. 27, No. 4, pp.1267–1296.

Kaur, A. and Dhiman, G. (2019) 'A review on search-based tools and techniques to identify bad code smells in object-oriented systems', *Harmony Search and Nature Inspired Optimization Algorithms*, pp.909–921.

Kaur, A., Jain, S., Goel, S. and Dhiman, G. (2021a) 'Prioritization of code smells in object-oriented software: a review', *Materials Today: Proceedings*, Vol. 14, No. 3, pp.290–303.

Kaur, H. and Sikka, G. (2022) 'Enriching module dependency graphs for improved software clustering', *International Journal of System of Systems Engineering*, Vol. 12, No. 1, pp.30–50.

Kaur, S. and Kaur, H. (2015) 'Identification and refactoring of bad smells to improve code quality', *International Journal of Scientific Engineering and Research*, Vol. 3, No. 8, pp.99–102.

Kaur, S., Awasthi, L.K. and Sangal, A.L. (2021b) 'A brief review on multi-objective software refactoring and a new method for its recommendation', *Archives of Computational Methods in Engineering*, Vol. 28, No. 4, pp.3087–3111.

Kerievsky, J. (2005) *Refactoring to Patterns*, Pearson Deutschland GmbH, ISBN: 0321213351, p.367.

Kitchenham, B., Brereton, O.P., Budgen, D., Turner, M., Bailey, J. and Linkman, S. (2009) 'Systematic literature reviews in software engineering–a systematic literature review', *Information and Software Technology*, Vol. 51, No. 1, pp.7–15.

Lacerda, G., Petrillo, F., Pimenta, M. and Guéhéneuc, Y.G. (2020) 'Code smells and refactoring: a tertiary systematic review of challenges and observations', *Journal of Systems and Software*, Vol. 167, p.110610.

Lakshmanan, M. and Manikandan, S. (2014) 'Multi-step automated refactoring for code smell', *International Journal of Research in Engineering and Technology (IJRET)*, Vol. 3, No. 03, pp.278–282.

Liu, H., Li, B., Yang, Y., Ma, W. and Jia, R. (2018) 'Exploring the impact of code smells on fine-grained structural change-proneness', *International Journal of Software Engineering and Knowledge Engineering*, Vol. 28, No. 10, pp.1487–1516.

Liu, H., Liu, Q., Niu, Z. and Liu, Y. (2015) 'Dynamic and automatic feedback-based threshold adaptation for code smell detection', *Transactions on Software Engineering (IEEE)*, Vol. 42, No. 6, pp.544–558.

Macia Bertran, I., Garcia, A. and von Staa, A. (2011) 'An exploratory study of code smells in evolving aspect-oriented systems', *Proceedings of the Tenth International Conference on Aspect-Oriented Software Development,* March, Porto de Galinhas Brazil, pp.203–214.

Mannan, U.A., Ahmed, I., Almurshed, R.A.M., Dig, D. and Jensen, C. (2016) 'Understanding code smells in android applications', *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft),* May, IEEE, Austin, TX, USA, pp.225–236.

Mantyla, M., Vanhanen, J. and Lassenius, C. (2003) 'A taxonomy and an initial empirical study of bad smells in code', *International Conference on Software Maintenance 2003, ICSM 2003 Proceedings,* September, IEEE, Amsterdam, Netherlands, pp.381–384.

Mariani, T. and Vergilio, S.R. (2017) 'A systematic review on search-based refactoring', *Information and Software Technology*, Vol. 83, pp.14–34.

Marinescu, R. (2005) 'Measurement and quality in object-oriented design', *21st IEEE International Conference on Software Maintenance (ICSM'05),* September, IEEE, pp.701–704.

Marinescu, R. and Ratiu, D. (2004) 'Quantifying the quality of object-oriented design: the factor-strategy model', *11th Working Conference on Reverse Engineering,* November IEEE, NW Washington, DC United States, pp.192–201.

Martin, R.C. (2005) 'Agile software development: principles, patterns, and practices', *Computing Reviews*, Vol. 46, No. 2, p.91.

Maruyama, K. and Tokoda, K. (2008) 'Security-aware refactoring alerting its impact on code vulnerabilities', *2008 15th Asia-Pacific Software Engineering Conference,* December, IEEE, Beijing, China, pp.445–452.

Masmali, O. and Badreddin, O. (2021) 'Metrics to measure code complexity based on software design: practical evaluation', *Future of Information and Communication Conference,* April, Springer, Cham, pp.142–157.

Massoni, T., Gheyi, R. and Borba, P. (2008) 'Formal model-driven program refactoring', *International Conference on Fundamental Approaches to Software Engineering,* March, Springer, Berlin, Heidelberg, pp.362–376.

Menshawy, R.S., Yousef, A.H. and Salem, A. (2021) 'Code smells and detection techniques: a survey', *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference* (*MIUCC*), May, IEEE, Cairo, Egypt, pp.78–83.

Michele, T., Fabio, P., Gabriele, B., Rocco, O., Di Penta, M., De Lucia, A. and Denys, P. (2015) 'When and why your code starts to smell bad', *37th IEEE/ACM International Conference on Software Engineering, ICSE2015*, IEEE Computer Society Press, Florence, Italy, pp.403–414.

Mittal, P., Singh, S. and Kahlon, K.S. (2011) 'Identification of error prone classes for fault prediction using object oriented metrics', *International Conference on Advances in Computing and Communications,* July, Springer, Berlin, Heidelberg, pp.58–68.

Mongiovi, M., Gheyi, R., Soares, G., Teixeira, L. and Borba, P. (2014) 'Making refactoring safer through impact analysis', *Science of Computer Programming*, Vol. 93, pp.39–64.

Mumtaz, H., Alshayeb, M., Mahmood, S. and Niazi, M. (2019) 'A survey on UML model smells detection techniques for software refactoring', *Journal of Software: Evolution and Process*, Vol. 31, No. 3, pp.2154.

Munro, M.J. (2005) 'Product metrics for automatic identification of "bad smell"design problems in java source-code', *11th IEEE International Software Metrics Symposium* (*METRICS'05*), September, IEEE, Como, Italy, pp.15–15.

Noguera, C., Kellens, A., De Roover, C. and Jonckers, V. (2012) 'Refactoring in the presence of annotations', *2012 28th IEEE International Conference on Software Maintenance (ICSM),* September, IEEE, Trento, Italy, pp.337–346.

Ouni, A., Gaikovina Kula, R., Kessentini, M. and Inoue, K. (2015) 'Web service antipatterns detection using genetic programming', *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation,* July, Madrid, Spain, pp.1351–1358.

Paiva, T., Damasceno, A., Figueiredo, E. and Sant'Anna, C. (2017) 'On the evaluation of code smells and detection tools', *Journal of Software Engineering Research and Development*, Vol. 5, No. 1, pp.1–28.

Palma, F., Dubois, J., Moha, N. and Guéhéneuc, Y.G. (2014) 'Detection of REST patterns and antipatterns: a heuristics-based approach', *International Conference on Service-Oriented Computing,* November, Springer, Berlin, Heidelberg, pp.230–244.

Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., Poshyvanyk, D. and De Lucia, A. (2014) 'Mining version histories for detecting code smells', *Transactions on Software Engineering*, IEEE, Vol.41, No. 5, pp.462–489.

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

Pérez, J. (2013) 'Refactoring planning for design smell correction: summary, opportunities and lessons learned', *2013 IEEE International Conference on Software Maintenance*, September, IEEE, Eindhoven, Netherlands, pp.572–577.

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

Rahad, K., Badreddin, O. and Mohsin Reza, S. (2021) 'The human in model-driven engineering loop: a case study on integrating handwritten code in model-driven engineering repositories', *Software: Practice and Experience*, Vol. 51, No. 6, pp.1308–1321.

Rama, G.M. (2010) 'A desiderata for refactoring-based software modularity improvement', *Proceedings of the 3rd India Software Engineering Conference*, February, Mysore India, pp.93–102.

Rani, A. and Chhabra, J.K. (2017) 'Evolution of code smells over multiple versions of softwares: an empirical investigation', *2017 2nd International Conference for Convergence in Technology (I2CT)*, April, IEEE, Mumbai, India, pp.1093–1098.

Rasool, G. and Arshad, Z. (2015) 'A review of code smell mining techniques', *Journal of Software: Evolution and Process*, Vol. 27, No. 11, pp.867–895.

Roberts, D., Brant, J. and Johnson, R. (1997) 'A refactoring tool for small talk', *Theory and Practice of Object Systems*, Vol. 3, No. 4, pp.253–263.

Sabir, F., Palma, F., Rasool, G., Guéhéneuc, Y.G. and Moha, N. (2019) 'A systematic literature review on the detection of smells and their evolution in object-oriented and service-oriented systems', *Software: Practice and Experience*, Vol. 49, No. 1, pp.3–39.

Sahin, D., Kessentini, M., Bechikh, S. and Deb, K. (2014) 'Code-smell detection as a bilevel problem', *Transactions on Software Engineering and Methodology (TOSEM)*, ACM, Vol. 24, No. 1, pp.1–44.

Santos, J.A.M. and Petronilo, G.X.A. (2022) 'Building empirical knowledge on the relationship between code smells and design patterns: an exploratory study', *Journal of Software: Evolution and Process*, Vol. 34, No. 9, pp.e2487.

Saranya, G., Nehemiah, H.K., Kannan, A. and Nithya, V. (2018) 'Model level code smell detection using Egapso based on similarity measures', *Alexandria Engineering Journal*, Vol. 57, No. 3, pp.1631–1642.

Sehgal, R., Mehrotra, D. and Nagpal, R. (2022) 'Is refactoring a solution to resolve code smell?', *International Journal of System of Systems Engineering*, Vol. 12, No 0.4, pp.371–385.

Sharma, T. and Spinellis, D. (2018) 'A survey on software smells', *Journal of Systems and Software*, Vol. 138, pp.158–173.

Sharma, T., Mishra, P. and Tiwari, R. (2016) 'Designite: A software design quality assessment tool', *Proceedings of the 1st International Workshop on Bringing Architectural Design Thinking into Developers' Daily Activities*, May, Austin, Texas, pp.1–4.

Singh, R., Bindal, A. and Kumar, A. (2019) 'A user feedback centric approach for detecting and mitigating god class code smell using frequent usage patterns', *Journal of Communications Software and Systems*, Vol. 15, No. 3, pp.245–253.

Singh, R., Bindal, A. and Kumar, A. (2020) 'A framework to improve quality of a java system by performing refactoring', *International Journal of System of Systems Engineering (IJSSE)*, Vol. 10, No. 4, pp.324–336.

Singh, R., Bindal, A.K. and Kumar, A. (2022) 'Improving software design by mitigating code smells', *International Journal of Software Innovation (IJSI)*, Vol. 10, No. 1, pp.1–21.

Singh, S. and Kaur, S. (2018) 'A systematic literature review: refactoring for disclosing code smells in object oriented software', *Ain Shams Engineering Journal*, Vol. 9, No. 4, pp.2129–2151.

Soomlek, C., Rijn, J.N.V. and Bonsangue, M.M. (2021) 'Automatic human-like detection of code smells', *International Conference on Discovery Science,* October, Springer, Cham, pp.19–28.

Tsantalis, N. and Chatzigeorgiou, A. (2009) 'Identification of move method refactoring opportunities', *Transactions on Software Engineering, IEEE*, Vol.35, No. 3, pp.347–367.

Vidal, S., Vazquez, H., Diaz-Pace, J.A., Marcos, C., Garcia, A. and Oizumi, W. (2015) 'JSpIRIT: a flexible tool for the analysis of code smells', *2015 34th International Conference of the Chilean Computer Science Society (SCCC),* November, IEEE, Santiago, Chile, pp.1–6.

Vidal, S.A., Marcos, C. and Díaz-Pace, J.A. (2016) 'An approach to prioritize code smells for refactoring', *Automated Software Engineering*, Vol. 23, No. 3, pp.501–532.

Wake, W.C. (2004) *Refactoring Workbook*, Addison-Wesley Professional.

Yamashita, A. and Moonen, L. (2013) 'To what extent can maintenance problems be predicted by code smell detection?–an empirical study', *Information and Software Technology*, Vol. 55, No. 12, pp.2223–2242.

Zhang, Y., Ge, C., Hong, S., Tian, R., Dong, C. and Liu, J. (2022) 'DeleSmell: Code smell detection based on deep learning and latent semantic analysis', *Knowledge-Based Systems*, Vol. 255, pp.109737.

Zhu, C., Zhang, X., Feng, Y. and Chen, L. (2018) 'An empirical study of the impact of code smell on file changes', *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, July, IEEE, Lisbon, Portugal, pp.238–248.

Notes

[1]http://www.springer.com/in/

[2]http://www.sciencedirect.com/

[3]http://dl.acm.org/

[4]http://ieeexplore.ieee.org/

[5]https://onlinelibrary.wiley.com/

[6]https://scholar.google.com/

[7]https://www.tandfonline.com/

[8]https://github.com/thiru578/Multilabel-Dataset

[9]http://essere.disco.unimib.it/reverse/MLCSD.html

Appendix 1: List of primary studies included for systematic literature review

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S1	A quantitative evaluation of maintainability enhancement by refactoring	Kataoka, Y., Imai, T., Andou, H. and Fukaya, T.	2002	Conference	*International Conference on Software Maintenance, IEEE*
S2	Supporting software development through declaratively codified programming patterns	Mens, K., Michiels, I. and Wuyts, R.	2002	Journal	*Expert Systems with Applications*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S3	Tools and Environments A Survey of Refactoring Tools	Adam, M.J. and Slifka, R.D.	2002	Report	*CMPT 487-SE*
S4	Object-oriented reengineering patterns	Demeyer, S., Ducasse, S. and Nierstrasz, O.	2002	Book	*Elsevier*
S5	Extreme Programming and Database Administration: Problems, Solutions, and Issues	Hassan, A.M. and Elssamadisy, A.	2002	Journal	*Proceedings XP*
S6	Software design quality: Style and substance	Bieman, J.M., Alexander, R., Munger III, P.W. and Meunier, E.	2002	Conference	*ICSE 2002 Workshop on Software Quality*
S7	Recognizing and responding to "bad smells" in extreme programming	Elssamadisy, A. and Schalliol, G.	2002	Conference	*Proceedings of the 24th International conference on Software Engineering*
S8	Refactoring: Current research and future trends	Mens, T., Demeyer, S., Du Bois, B., Stenten, H. and Van Gorp, P.	2003	Journal	*Electronic Notes in Theoretical Computer Science*
S9	A taxonomy and an initial empirical study of bad smells in code	Mantyla, M., Vanhanen, J. and Lassenius, C.	2003	Conference	*International Conference on Software Maintenance*
S10	Refactoring in Large Software Projects	Roock, S. and Lippert, M.	2003	Book	*John Wiley and Sons*
S11	Graph theoretical indicators and refactoring	Zimmer, J.A.	2003	Conference	*Conference on Extreme Programming and Agile Methods*
S12	Automated Code Smell Detection and Refactoring by Source Transformation	Grant, S.	2003	Workshop	*n WCRE Workshop on REFactoring: Achievements, Challanges, Effects*
S13	A stochastic approach to automated design improvement	O'Keeffe, M. and Cinnéide, M.O.	2003	Conference	*ACM International Conference Proceeding Series*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S14	Quality-driven object-oriented code restructuring	Tahvildari, L. and Kontogiannis, K.	2003	Conference	*Proceedings of Proceedings of ICSE Workshop on Software Quality*
S15	Towards automating source-consistent UML refactorings	Gorp, P.V., Stenten, H., Mens, T. and Demeyer, S.	2003	Conference	*International Conference on the Unified Modeling Language*
S16	Bad smells in software-a taxonomy and an empirical study	Mantyla, M.	2003	Thesis	*Helsinki University of Technology*
S17	A survey of software refactoring	Mens, T. and Tourwé, T.	2004	Journal	*IEEE Transactions on Software Engineering*
S18	Applying refactoring techniques to UML/OCL models	Correa, A. and Werner, C.	2004	Conference	*International Conference on the Unified Modeling Language*
S19	Bad smells-humans as code critics	Mantyla, M.V., Vanhanen, J. and Lassenius, C.	2004	Conference	*20th IEEE International Conference on Software Maintenance*
S20	Jiad: a tool to infer design patterns in refactoring	Rajesh, J. and Janakiram, D.	2004	Conference	*Proceedings of the 6th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*
S21	Automatic Detection of Refactoring Opportunities	Carneiro, G., Mendonça, M. and Maldonado, J.C.	2004	Journal	*Transactions on Software Engineering*
S22	Developments trends in refactoring and measurement tools	Juhász, I. and Guta, G.	2004	Conference	*Proceedings of the International Conference on Applied Computing*
S23	Improving evolvability through refactoring	Ratzinger, J., Fischer, M. and Gall, H.	2005	Conference	*Proceedings of the 2005 International Workshop on Mining Software repositories*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S24	An experiment on subjective evolvability evaluation of object-oriented software: explaining factors and interrater agreement	Mantyla, M.V.	2005	Conference	*2005 International Symposium on Empirical Software Engineering*
S25	Diagnosing design problems in object oriented systems	Trifu, A. and Marinescu, R.	2005	Conference	*12th Working Conference on Reverse Engineering (WCRE'05)*
S26	Refactoring OCL annotated UML class diagrams	Marković, S. and Baar, T.	2005	Conference	*International Conference On Model Driven Engineering Languages And Systems*
S27	On refactoring support based on code clone dependency relation	Yoshida, N., Higo, Y., Kamiya, T., Kusumoto, S. and Inoue, K.	2005	Conference	*11th IEEE International Software Metrics Symposium*
S28	Multi-criteria detection of bad smells in code with UTA method	Walter, B. and Pietrzak, B.	2005	Conference	*International Conference on Extreme Programming and Agile Processes in Software Engineering*
S29	Detecting structural refactoring conflicts using critical pair analysis	Mens, T., Taentzer, G. and Runge, O.	2005	Journal	*Electronic Notes in Theoretical Computer Science*
S30	Exploring Bad Code Smells Dependencies	Pietrzak, B. and Walter, B.	2005	Journal	*Software Engineering: Evolution and Emerging Technologies*
S31	Detecting Bad Code Smells for Refactoring by using Historical Data of Source Control System	Sheikh, S.I.	2005	Thesis	*National University of Computer and Emerging Sciences, Lahore, Pakistan*
S32	Refactoring to Patterns/Data Access Patterns	Schneider, R.	2005	Journal	*Software Quality Professional*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S33	Evaluating software refactoring tool support	Mealy, E. and Strooper, P.	2006	Conference	*Australian Software Engineering Conference (ASWEC'06)*
S34	Leveraging code smell detection with inter-smell relations	Pietrzak, B. and Walter, B.	2006	Conference	*International Conference on Extreme Programming and Agile Processes in Software Engineering*
S35	Drivers for software refactoring decisions	Mäntylä, M.V. and Lassenius, C.	2006	Conference	*Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*
S36	Does refactoring improve reusability?	Moser, R., Sillitti, A., Abrahamsson, P. and Succi, G.	2006	Conference	*International conference on software reuse*
S37	Predicting classes in need of refactoring: an application of static metrics	Zhao, L. and Hayes, J.	2006	Conference	*Proceedings of the 2nd International PROMISE Workshop*
S38	Subjective evaluation of software evolvability using code smells: An empirical study	Mäntylä, M.V. and Lassenius, C.	2006	Journal	*Empirical Software Engineering*
S39	Supporting refactoring activities using histories of program modification	Hayashi, S., Saeki, M. and Kurihara, M.	2006	Journal	*IEICE transactions on information and systems*
S40	REFACTORING Refactoring can help you wash away code smells. Here's how to get started	Ray, C.K.	2006	Journal	*BETTER SOFTWARE*
S41	A heuristic-based approach to code-smell detection	Kirk, D., Roper, M. and Wood, M.	2007	Conference	*Proc. 1st Workshop on Refactoring Tools*
S42	Challenges in model refactoring	Mens, T., Taentzer, G. and Müller, D.	2007	Conference	*Proc. 1st Workshop on Refactoring Tools*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S43	High-impact refactoring based on architecture violations	Bourquin, F. and Keller, R.K.	2007	Conference	*11th European Conference on Software Maintenance and Reengineering*
S44	An empirical evaluation of refactoring	Wilking, D., Kahn, U.F. and Kowalewski, S.	2007	Journal	*e-Informatica Software Engineering Journal*
S45	Refactoring object constraint language specifications	Correa, A. and Werner, C.	2007	Journal	*Software and Systems Modeling*
S46	Analysing refactoring dependencies using graph transformation	Mens, T., Taentzer, G. and Runge, O.	2007	Journal	*Software and Systems Modeling*
S47	From Bad Smells to Refactoring: Metrics Smoothing the Way	Crespo, Y., López, C. and Martinez, M.E.M.	2007	Conference	*Object-Oriented Design Knowledge: Principles, Heuristics and Best Practices*
S48	Refactoring--does it improve software quality?	Stroggylos, K. and Spinellis, D.	2007	Conference	*Fifth International Workshop on Software Quality*
S49	Towards automated restructuring of object oriented systems	Trifu, A. and Reupke, U.	2007	Conference	*11th European Conference on Software Maintenance and Reengineering*
S50	Towards a refactoring guideline using code clone classification	Schulze, S., Kuhlemann, M. and Rosenmüller, M.	2008	Conference	*Proceedings of the 2nd Workshop on Refactoring Tools*
S51	Seven habits of a highly effective smell detector	Murphy-Hill, E. and Black, A.P.	2008	Conference	*Proceedings of the 2008 international workshop on Recommendation systems for software engineering*
S52	Impact of metrics based refactoring on the software quality: a case study	Shrivastava, S.V. and Shrivastava, V.	2008	Conference	*TENCON 2008-2008 IEEE Region 10 Conference*
S53	Scalable, expressive, and context-sensitive code smell display	Murphy-Hill, E.	2008	Conference	*Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S54	A critical analysis of two refactoring tools	Drozdz, M.Z.	2008	Thesis	*University of Pretoria*
S55	A model to identify refactoring effort during maintenance by mining source code repositories	Moser, R., Pedrycz, W., Sillitti, A. and Succi, G.	2008	Conference	*International Conference on Product Focused Software Process Improvement*
S56	A catalogue of lightweight visualizations to support code smell inspection	Parnin, C., Görg, C. and Nnadi, O.	2008	Conference	*Proceedings of the 4th ACM Symposium on Software Visualization*
S57	JDeodorant: Identification and removal of type-checking bad smells	Tsantalis, N., Chaikalis, T. and Chatzigeorgiou, A.	2008	Conference	*2008 12th European conference on software maintenance and reengineering*
S58	Visualizing Java code smells with dot plots	Jefferson, A.H.	2008	Book	*Southern Illinois University at Carbondale*
S59	Classifying desirable features of software visualization tools for corrective maintenance	Sensalire, M., Ogao, P. and Telea, A.	2008	Conference	*Proceedings of the 4th ACM symposium on Software visualization*
S60	A metric-based approach to identifying refactoring opportunities for merging code clones in a Java software system	Higo, Y., Kusumoto, S. and Inoue, K.	2008	Journal	*Journal of Software Maintenance and Evolution: Research and Practice*
S61	Search-based refactoring for software maintenance	O'Keeffe, M. and Cinnéide, M.O.	2008	Journal	*Journal of Systems and Software*
S62	Adaptation of refactoring strategies to multiple axes of modularity: characteristics and criteria	Arnaoudova, V. and Constantinides, C.	2008	Conference	*2008 Sixth International Conference on Software Engineering Research, Management and Applications*
S63	Improving the precision of fowler's definitions of bad smells	Zhang, M., Baddoo, N., Wernick, P. and Hall, T.	2008	Conference	*2008 32nd Annual IEEE Software Engineering Workshop*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S64	Classification of model refactoring approaches	Mohamed, M., Romdhani, M. and Ghédira, K.	2009	Journal	*Journal of Object Technology*
S65	Identifying architectural bad smells	Garcia, J., Popescu, D., Edwards, G. and Medvidovic, N.	2009	Conference	*2009 13th European Conference on Software Maintenance and Reengineering*
S66	Strengthening refactoring: Towards software evolution with quantitative and experimental grounds	Bryton, S. and Abreu, F.B.	2009	Conference	*2009 Fourth International Conference on Software Engineering Advances*
S67	Toward a catalogue of architectural bad smells	Garcia, J., Popescu, D., Edwards, G. and Medvidovic, N.	2009	Conference	*International conference on the quality of software architectures*
S68	The evolution and impact of code smells: A case study of two open source systems	Olbrich, S., Cruzes, D.S., Basili, V. and Zazworka, N.	2009	Conference	*2009 3rd international symposium on empirical software engineering and measurement*
S69	JSmell: A Bad Smell detection tool for Java systems	Roperia, N.	2009	Thesis	*California State University*
S70	An exploratory study of the impact of code smells on software change-proneness	Khomh, F., Di Penta, M. and Gueheneuc, Y.G.	2009	Conference	*2009 16th Working Conference on Reverse Engineering*
S71	Refactoring to improve the understandability of specifications written in object constraint language	Correa, A., Werner, C. and Barros, M.	2009	Journal	*IET software*
S72	Refactoring of crosscutting concerns with metaphor-based heuristics	da Silva, B.C., Figueiredo, E., Garcia, A. and Nunes, D.	2009	Journal	*Electronic Notes in Theoretical Computer Science*
S73	Empirical investigation of refactoring effect on software quality	Alshayeb, M.	2009	Journal	*Information and software technology*
S74	Identification of move method refactoring opportunities	Tsantalis, N. and Chatzigeorgiou, A.	2009	Journal	*IEEE Transactions on Software Engineering*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S75	A literature review on code smells and refactoring	Wangberg, R.	2010	Thesis	*University of Oslo*
S76	An interactive ambient visualization for code smells	Murphy-Hill, E. and Black, A.P.	2010	Conference	*Proceedings of the 5th international symposium on Software visualization*
S77	Building empirical support for automated code smell detection	Schumacher, J., Zazworka, N., Shull, F., Seaman, C. and Shaw, M.	2010	Conference	*Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement*
S78	Investigating the evolution of bad smells in object-oriented code	Chatzigeorgiou, A. and Manakos, A.	2010	Conference	*2010 Seventh International Conference on the Quality of Information and Communications Technology*
S79	Identification of refactoring opportunities introducing polymorphism	Tsantalis, N. and Chatzigeorgiou, A.	2010	Journal	*Journal of Systems and Software*
S80	Reducing subjectivity in code smells detection: Experimenting with the long method	Bryton, S., Abreu, F.B. and Monteiro, M.	2010	Conference	*2010 Seventh International Conference on the Quality of Information and Communications Technology*
S81	Are all code smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems	Olbrich, S.M., Cruzes, D.S. and Sjøberg, D.I.	2010	Conference	*2010 IEEE international conference on software maintenance*
S82	A visual based framework for the model refactoring techniques	Štolc, M. and Polášek, I.	2010	Conference	*2010 IEEE 8th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S83	An empirical investigation of code smell 'deception' and research contextualisation through paul's criteria	Counsell, S., Hamza, H. and Hierons, R.M.	2010	Journal	*Journal of computing and information technology*
S84	Empirical software evolvability-code smells and human evaluations	Mäntylä, M.V.	2010	Conference	*2010 IEEE International Conference on Software Maintenance*
S85	Assure high quality code using refactoring and obfuscation techniques	Long, T., Liu, L., Yu, Y. and Wan, Z.	2010	Conference	*2010 Fifth International Conference on Frontier of Computer Science and Technology*
S86	Domain-specific tailoring of code smells: an empirical study	Guo, Y., Seaman, C., Zazworka, N. and Shull, F.	2010	Conference	*Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*
S87	Automatic Detection of Possible Refactorings	Peldzius, S.	2010	Conference	*Proceedings of the 16th International Conference on Information and Software Technologies (ICIST)*
S88	The theory of relative dependency: Higher coupling concentration in smaller modules	Koru, A.G. and El Emam, K.	2010	Journal	*IEEE software*
S89	Sub-clone refactoring in open source software artifacts	Tairas, R. and Gray, J.	2010	Conference	*Proceedings of the 2010 ACM Symposium on Applied Computing*
S90	Evaluation and improvement of software architecture: Identification of design problems in object-oriented systems and resolution through refactorings	Tsantalis, N.	2010	Thesis	*Univ. Macedonia*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S91	CodeVizard: a tool to aid the analysis of software evolution	Zazworka, N. and Ackermann, C.	2010	Conference	*Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*
S92	Impact of refactoring on quality code evaluation	Fontana, F.A. and Spinelli, S.	2011	Conference	*Proceedings of the 4th Workshop on Refactoring Tools*
S93	Ranking refactoring suggestions based on historical volatility	Tsantalis, N. and Chatzigeorgiou, A.	2011	Conference	*2011 15th European Conference on Software Maintenance and Reengineering*
S94	TrueRefactor: An automated refactoring tool to improve legacy system and application comprehensibility	Griffith, I., Wahl, S. and Izurieta, C.	2011	Conference	*24th International Conference on Computer Applications in Industry and Engineering, ISCA*
S95	Understanding the longevity of code smells: preliminary results of an explanatory survey	Arcoverde, R., Garcia, A. and Figueiredo, E.	2011	Conference	*Proceedings of the 4th Workshop on Refactoring Tools*
S96	Code bad smells: a review of current knowledge	Zhang, M., Hall, T. and Baddoo, N.	2011	Journal	*Journal of Software Maintenance and Evolution: research and practice*
S97	An experience report on using code smells detection tools	Fontana, F.A., Mariani, E., Mornioli, A., Sormani, R. and Tonello, A.	2011	Conference	*2011 IEEE fourth international conference on software testing, verification and validation workshops*
S98	Using software metrics to select refactoring for long method bad smell	Meananeatra, P., Rongviriyapanish, S. and Apiwattanapong, T.	2011	Conference	*The 8th Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI)*
S99	Exploring the eradication of code smells: An empirical and theoretical perspective	Counsell, S., Hierons, R.M., Hamza, H., Black, S. and Durrand, M.	2011	Journal	*Advances in Software Engineering*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S100	Detecting architecturally-relevant code smells in evolving software systems	Bertran, I.M.	2011	Conference	*Proceedings of the 33rd International Conference on Software Engineering (pp. 1090-1093).*
S101	Looking for patterns in code bad smells relations	Walter, B. and Martenka, P.	2011	Conference	*2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*
S102	An empirical assessment of refactoring impact on software quality using a hierarchical quality model	Shatnawi, R. and Li, W.	2011	Journal	*International Journal of Software Engineering and Its Applications*
S103	Schedule of bad smell detection and resolution: A new way to save effort	Liu, H., Ma, Z., Shao, W. and Niu, Z.	2011	Journal	*IEEE transactions on Software Engineering*
S104	Code smell detecting tool and code smell-structure bug relationship	Danphitsanuphan, P. and Suwantada, T.	2012	Conference	*2012 Spring Congress on Engineering and Technology*
S105	Evaluating the lifespan of code smells using software repository mining	Peters, R. and Zaidman, A.	2012	Conference	*2012 16th European conference on software maintenance and reengineering*
S106	Quantifying the effect of code smells on maintenance effort	Sjøberg, D.I., Yamashita, A., Anda, B.C., Mockus, A. and Dybå, T.	2012	Journal	*IEEE Transactions on Software Engineering*
S107	Automatic detection of bad smells in code: An experimental assessment	Fontana, F.A., Braione, P. and Zanoni, M.	2012	Journal	*J. Object Technol.*
S108	Automated refactoring to the strategy design pattern	Christopoulou, A., Giakoumakis, E.A., Zafeiris, V.E. and Soukara, V.	2012	Journal	*Information and Software Technology*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S109	Investigating the impact of code smells debt on quality code evaluation	Fontana, F.A., Ferme, V. and Spinelli, S.	2012	Conference	*2012 Third International Workshop on Managing Technical Debt (MTD)*
S110	CodeSmellExplorer: Tangible exploration of code smells and refactorings	Raab, F.	2012	Conference	*2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*
S111	Identifying refactoring sequences for improving software maintainability	Meananeatra, P.	2012	Conference	*Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*
S112	Identifying extract-method refactoring candidates automatically	Sharma, T.	2012	Conference	*Proceedings of the Fifth Workshop on Refactoring Tools*
S113	Can software faults be analyzed using bad code smells?: An empirical study	Dhillon, P.K. and Sidhu, G.	2012	Journal	*Int J Sci Res Publ*
S114	Reconciling manual and automatic refactoring	Ge, X., DuBose, Q.L. and Murphy-Hill, E.	2012	Conference	*2012 34th International Conference on Software Engineering (ICSE)*
S115	Assuring software quality by code smell detection	Van Emden, E. and Moonen, L.	2012	Conference	*2012 19th Working Conference on Reverse Engineering*
S116	Do code smells reflect important maintainability aspects?	Yamashita, A. and Moonen, L.	2012	Conference	*2012 28th IEEE International Conference on Software Maintenance (ICSM)*
S117	Refactoring edit history of source code	Hayashi, S., Omori, T., Zenmyo, T., Maruyama, K. and Saeki, M.	2012	Conference	*2012 28th IEEE International Conference on Software Maintenance (ICSM)*
S118	Move code refactoring with dynamic analysis	Kimura, S., Higo, Y., Igaki, H. and Kusumoto, S.	2012	Conference	*2012 28th IEEE International Conference on Software Maintenance (ICSM)*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S119	On the existence of high-impact refactoring opportunities in programs	Dietrich, J., McCartin, C., Tempero, E. and Shah, S.M.A.	2012	Conference	*Proceedings of the Thirty-fifth Australasian Computer Science Conference*
S120	Assessment of Code Smells for Predicting Class Change Proneness	Malhotra, R. and Pritam, N.	2012	Journal	*Software Quality Professional*
S121	Monitor-based instant software refactoring	Liu, H., Guo, X. and Shao, W.	2013	Journal	*IEEE Transactions on Software Engineering*
S122	A multidimensional empirical study on refactoring activity	Tsantalis, N., Guana, V., Stroulia, E. and Hindle, A.	2013	Conference	*CASCON*
S123	A comparative study on code smell detection tools	Hamid, A., Ilyas, M., Hummayun, M. and Nawaz, A.	2013	Journal	*International Journal of Advanced Science and Technology*
S124	Do developers care about code smells? An exploratory survey	Yamashita, A. and Moonen, L.	2013	Conference	*2013 20th working conference on reverse engineering (WCRE)*
S125	Conflict-aware optimal scheduling of prioritised code clone refactoring	Zibran, M.F. and Roy, C.K.	2013	Journal	*IET software*
S126	Identification of refused bequest code smells	Ligu, E., Chatzigeorgiou, A., Chaikalis, T. and Ygeionomakis, N.	2013	Conference	*2013 IEEE International Conference on Software Maintenance*
S127	Detection and refactoring of bad smell caused by large scale	Dexun, J., Peijun, M., Xiaohong, S. and Tiantian, W.	2013	Journal	*International Journal of Software Engineering and Applications*
S128	Jsnose: Detecting javascript code smells	Fard, A.M. and Mesbah, A.	2013	Conference	*2013 IEEE 13th international working conference on Source Code Analysis and Manipulation (SCAM)*
S129	Implementation and analysis of a refactoring tool for detecting code smells	Kaur, A. and Raperia, H.	2013	Journal	*International Journal of Computers and Technology*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S130	Investigating the impact of code smells on system's quality: An empirical study on systems of different application domains	Fontana, F.A., Ferme, V., Marino, A., Walter, B. and Martenka, P.	2013	Conference	*2013 IEEE International Conference on Software Maintenance*
S131	Detecting bad smells in source code using change history information	Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A. and Poshyvanyk, D.	2013	Conference	*2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*
S132	Identification of generalization refactoring opportunities	Liu, H., Niu, Z., Ma, Z. and Shao, W.	2013	Journal	*Automated Software Engineering*
S133	Search-based refactoring using recorded code changes	Ouni, A., Kessentini, M. and Sahraoui, H.	2013	Conference	*2013 17th European Conference on Software Maintenance and Reengineering*
S134	Code smells as system-level indicators of maintainability: An empirical study	Yamashita, A. and Counsell, S.	2013	Journal	*Journal of Systems and Software*
S135	Exploring the impact of inter-smell relations on software maintainability: An empirical study	Yamashita, A. and Moonen, L.	2013	Conference	*2013 35th International Conference on Software Engineering (ICSE)*
S136	Trends, opportunities and challenges of software refactoring: A systematic literature review	Abebe, M. and Yoo, C.J.	2014	Journal	*International Journal of software engineering and its Applications*
S137	A robust multi-objective approach for software refactoring under uncertainty	Mkaouer, M.W., Kessentini, M., Bechikh, S. and Ó Cinnéide, M.	2014	Conference	*International Symposium on Search Based Software Engineering*
S138	Investigating the evolution of code smells in object-oriented systems	Chatzigeorgiou, A. and Manakos, A.	2014	Journal	*Innovations in Systems and Software Engineering*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S139	Recommendation system for software refactoring using innovization and interactive dynamic optimization	Mkaouer, M.W., Kessentini, M., Bechikh, S., Deb, K. and Ó Cinnéide, M.	2014	Conference	*Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*
S140	Automated pattern-directed refactoring for complex conditional statements	Liu, W., Hu, Z.G., Liu, H.T. and Yang, L.	2014	Journal	*Journal of Central South University*
S141	Bulk fixing coding issues and its effects on software quality: Is it worth refactoring?	Szoke, G., Antal, G., Nagy, C., Ferenc, R. and Gyimóthy, T.	2014	Conference	*2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*
S142	A case study of refactoring large-scale industrial systems to efficiently improve source code quality	Szőke, G., Nagy, C., Ferenc, R. and Gyimóthy, T.	2014	Conference	*International Conference on Computational Science and Its Applications*
S143	Classification and summarization of software refactoring researches: a literature review approach	Abebe, M. and Yoo, C.J.	2014	Journal	*Advanced Science and Technology Letters*
S144	Mining version histories for detecting code smells	Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., Poshyvanyk, D. and De Lucia, A.	2014	Journal	*IEEE Transactions on Software Engineering*
S145	Multi-Step Automated Refactoring For Code Smell	Lakshmanan, M. and Manikandan, S.	2014	Journal	*IJRET: International Journal of Research in Engineering and Technology*
S146	Identifying accurate refactoring opportunities using metrics	Bian, Y., Su, X. and Ma, P.	2014	Conference	*Proceedings of International Conference on Soft Computing Techniques and Engineering Application*
S147	Recommending refactoring operations in large software systems	Bavota, G., Lucia, A.D., Marcus, A. and Oliveto, R.	2014	Journal	*Recommendation Systems in Software Engineering*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S148	Functional over-related classes bad smell detection and refactoring suggestions	Dexun, J., Peijun, M., Xiaohong, S. and Tiantian, W.	2014	Journal	*International Journal of Software Engineering and Applications*
S149	Assessing the capability of code smells to explain maintenance problems: an empirical study combining quantitative and qualitative data	Yamashita, A.	2014	Journal	*Empirical Software Engineering*
S150	Some code smells have a significant but small effect on faults	Hall, T., Zhang, M., Bowes, D. and Sun, Y.	2014	Journal	*ACM Transactions on Software Engineering and Methodology (TOSEM)*
S151	Ranking The Refactoring Techniques Based on The External Quality Attributes	Alshehri, S. and Aljuhani, A.	2014	Journal	*International Journal of Research in Engineering and Science (IJRES)*
S152	Distance metric based divergent change bad smell detection and refactoring scheme analysis	Jiang, D., Ma, P., Su, X. and Wang, T.	2014	Journal	*International Journal of Innovative Computing, Information and Control*
S153	Manual refactoring changes with automated refactoring validation	Ge, X. and Murphy-Hill, E.	2014	Conference	*Proceedings of the 36th International Conference on Software Engineering*
S154	High dimensional search-based software engineering: finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III	Mkaouer, M.W., Kessentini, M., Bechikh, S., Deb, K. and Ó Cinnéide, M.	2014	Conference	*Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*
S155	Case study on software refactoring tactics	Liu, H., Liu, Y., Xue, G. and Gao, Y.	2014	Journal	*IET software*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S156	Do they really smell bad? a study on developers' perception of bad code smells	Palomba, F., Bavota, G., Di Penta, M., Oliveto, R. and De Lucia, A.	2014	Conference	*2014 IEEE International Conference on Software Maintenance and Evolution*
S157	FaultBuster: An automatic code smell refactoring toolset	Szőke, G., Nagy, C., Fülöp, L.J., Ferenc, R. and Gyimóthy, T.	2015	Conference	*2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*
S158	On experimenting refactoring tools to remove code smells	Fontana, F.A., Mangiacavalli, M., Pochiero, D. and Zanoni, M.	2015	Conference	*Scientific Workshop Proceedings of the XP2015*
S159	An experimental investigation on the innate relationship between quality and refactoring	Bavota, G., De Lucia, A., Di Penta, M., Oliveto, R. and Palomba, F.	2015	Journal	*Journal of Systems and Software*
S160	Prioritizing code-smells correction tasks using chemical reaction optimization	Ouni, A., Kessentini, M., Bechikh, S. and Sahraoui, H.	2015	Journal	*Software Quality Journal*
S161	Identifying refactoring opportunities in object-oriented code: A systematic literature review	Al Dallal, J.	2015	Journal	*Information and software Technology*
S162	AutoRefactoring: A platform to build refactoring agents	dos Santos Neto, B.F., Ribeiro, M., Da Silva, V.T., Braga, C., De Lucena, C.J.P. and de Barros Costa, E.	2015	Journal	*Expert systems with applications*
S163	Improving multi-objective code-smells correction using development history	Ouni, A., Kessentini, M., Sahraoui, H., Inoue, K. and Hamdi, M.S.	2015	Journal	*Journal of Systems and Software*
S164	A review of code smell mining techniques	Rasool, G. and Arshad, Z.	2015	Journal	*Journal of Software: Evolution and Process*

ID	Title	Authors	Year	Category	Source
S165	When and why your code starts to smell bad	Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A. and Poshyvanyk, D.	2015	Conference	*2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*
S166	On the use of time series and search based software engineering for refactoring recommendation	Wang, H., Kessentini, M., Grosky, W. and Meddeb, H.	2015	Conference	*Proceedings of the 7th International Conference on Management of computational and collective intElligence in Digital EcoSystems*
S167	Towards assessing software architecture quality by exploiting code smell relations	Fontana, F.A., Ferme, V. and Zanoni, M.	2015	Conference	*2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics*
S168	Challenges to and solutions for refactoring adoption: An industrial perspective	Sharma, T., Suryanarayana, G. and Samarthyam, G.	2015	Journal	*IEEE Software*
S169	Investigation of code smells in different software domains	Delchev, M. and Harun, M.F.	2015	Journal	*Full-scale Software Engineering*
S170	JSpIRIT: a flexible tool for the analysis of code smells	Vidal, S., Vazquez, H., Diaz-Pace, J.A., Marcos, C., Garcia, A. and Oizumi, W.	2015	Conference	*2015 34th International Conference of the Chilean Computer Science Society*
S171	Landfill: An open dataset of code smells with public evaluation	Palomba, F., Di Nucci, D., Tufano, M., Bavota, G., Oliveto, R., Poshyvanyk, D. and De Lucia, A.	2015	Conference	*2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*
S172	UML model refactoring: a systematic literature review	Misbhauddin, M. and Alshayeb, M.	2015	Journal	*Empirical Software Engineering*
S173	Dynamic and automatic feedback-based threshold adaptation for code smell detection	Liu, H., Liu, Q., Niu, Z. and Liu, Y.	2015	Journal	*IEEE Transactions on Software Engineering*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S174	Architectural refactoring: A task-centric view on software evolution	Zimmermann, O.	2015	Journal	*IEEE Software*
S175	Are test smells really harmful? an empirical study	Bavota, G., Qusef, A., Oliveto, R., De Lucia, A. and Binkley, D.	2015	Journal	*Empirical Software Engineering*
S176	An approach to prioritize code smells for refactoring	Vidal, S.A., Marcos, C. and Díaz-Pace, J.A.	2016	Journal	*Automated Software Engineering*
S177	Revisiting the relationship between code smells and refactoring	Yoshida, N., Saika, T., Choi, E., Ouni, A. and Inoue, K.	2016	Conference	*2016 IEEE 24th International Conference on Program Comprehension (ICPC)*
S178	Does refactoring improve software structural quality? a longitudinal study of 25 projects	Cedrim, D., Sousa, L., Garcia, A. and Gheyi, R.	2016	Conference	*Proceedings of the 30th Brazilian Symposium on Software Engineering*
S179	JDeodorant: clone refactoring	Mazinanian, D., Tsantalis, N., Stein, R. and Valenta, Z.	2016	Conference	*Proceedings of the 38th international conference on software engineering companion*
S180	Code smell analyzer: a tool to teaching support of refactoring techniques source code	Sirqueira, T.F.M., Brandl, A.H.M., Pedro, E.J.P., de Souza Silva, R. and Araujo, M.A.P.	2016	Journal	*IEEE Latin America Transactions*
S181	Measuring refactoring benefits: a survey of the evidence	Ó Cinnéide, M., Yamashita, A. and Counsell, S.	2016	Conference	*Proceedings of the 1st International Workshop on Software Refactoring*
S182	On the use of design defect examples to detect model refactoring opportunities	Ghannem, A., El Boussaidi, G. and Kessentini, M.	2016	Journal	*Software Quality Journal*
S183	An empirical study on the effect of the order of applying software refactoring	Khrishe, Y. and Alshayeb, M.	2016	Conference	*2016 7th International Conference on Computer Science and Information Technology (CSIT)*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S184	A code refactoring dataset and its assessment regarding software maintainability	Kádár, I., Hegedus, P., Ferenc, R. and Gyimóthy, T.	2016	Conference	*2016 IEEE 23rd International conference on software analysis, Evolution, and Reengineering (SANER)*
S185	Do developers focus on severe code smells?	Saika, T., Choi, E., Yoshida, N., Haruna, S. and Inoue, K.	2016	Conference	*2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*
S186	Identifying extract method refactoring opportunities based on functional relevance	Charalampidou, S., Ampatzoglou, A., Chatzigeorgiou, A., Gkortzis, A. and Avgeriou, P.	2016	Journal	*IEEE Transactions on Software Engineering*
S187	An empirical study of bad smell in code on maintenance effort	Kumar, R., Singh, J. and Kaur, A.	2016	Journal	*Int. J. Comput. Sci. Eng*
S188	Context-based code smells prioritization for prefactoring	Sae-Lim, N., Hayashi, S. and Saeki, M.	2016	Conference	*2016 IEEE 24th International Conference on Program Comprehension (ICPC)*
S189	Designing and developing automated refactoring transformations: An experience report	Szoke, G., Nagy, C., Ferenc, R. and Gyimóthy, T.	2016	Conference	*2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*
S190	Multi-criteria code refactoring using search-based software engineering: An industrial case study	Ouni, A., Kessentini, M., Sahraoui, H., Inoue, K. and Deb, K.	2016	Journal	*ACM Transactions on Software Engineering and Methodology (TOSEM)*
S191	Assessment of the Code Refactoring Dataset Regarding the Maintainability of Methods	Kádár, I., Hegedűs, P., Ferenc, R. and Gyimóthy, T.	2016	Conference	*International Conference on Computational Science and Its Applications*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S192	Comparing and experimenting machine learning techniques for code smell detection	Arcelli Fontana, F., Mäntylä, M.V., Zanoni, M. and Marino, A.	2016	Journal	*Empirical Software Engineering*
S193	MORE: A multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells	Ouni, A., Kessentini, M., Ó Cinnéide, M., Sahraoui, H., Deb, K. and Inoue, K.	2017	Journal	*Journal of Software: Evolution and Process*
S194	Understanding the impact of refactoring on smells: A longitudinal study of 23 software projects	Cedrim, D., Garcia, A., Mongiovi, M., Gheyi, R., Sousa, L., de Mello, R.,... and Chávez, A.	2017	Conference	*Proceedings of the 2017 11th Joint Meeting on foundations of Software Engineering*
S195	A robust multi-objective approach to balance severity and importance of refactoring opportunities	Mkaouer, M.W., Kessentini, M., Cinnéide, M.Ó., Hayashi, S. and Deb, K.	2017	Journal	*Empirical Software Engineering*
S196	A systematic review on search-based refactoring	Mariani, T. and Vergilio, S.R.	2017	Journal	*Information and Software Technology*
S197	An exploratory study on the relationship between changes and refactoring	Palomba, F., Zaidman, A., Oliveto, R. and De Lucia, A.	2017	Conference	*2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*
S198	Empirical evaluation of the impact of object-oriented code refactoring on quality attributes: A systematic literature review	Al Dallal, J. and Abdin, A.	2017	Journal	*IEEE Transactions on Software Engineering*
S199	When and why your code starts to smell bad (and whether the smells go away)	Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A. and Poshyvanyk, D.	2017	Journal	*IEEE Transactions on Software Engineering*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S200	Code smell severity classification using machine learning techniques	Fontana, F.A. and Zanoni, M.	2017	Journal	*Knowledge-Based Systems*
S201	How do developers select and prioritize code smells? A preliminary study	Sae-Lim, N., Hayashi, S. and Saeki, M.	2017	Conference	*2017 IEEE International Conference on Software Maintenance and evolution (ICSME)*
S202	Empirical study on refactoring large-scale industrial systems and its effects on maintainability	Szőke, G., Antal, G., Nagy, C., Ferenc, R. and Gyimóthy, T.	2017	Journal	*Journal of Systems and Software*
S203	How developers perceive smells in source code: A replicated study	Taibi, D., Janes, A. and Lenarduzzi, V.	2017	Journal	*Information and Software Technology*
S204	A systematic literature review: Refactoring for disclosing code smells in object oriented software	Singh, S. and Kaur, S.	2018	Journal	*Ain Shams Engineering Journal*
S205	The scent of a smell: An extensive comparison between textual and structural smells	Palomba, F., Panichella, A., Zaidman, A., Oliveto, R. and De Lucia, A.	2018	Conference	*Proceedings of the 40th International Conference on Software Engineering*
S206	An empirical study to improve software security through the application of code refactoring	Mumtaz, H., Alshayeb, M., Mahmood, S. and Niazi, M.	2018	Journal	*Information and Software Technology*
S207	Assessing the refactoring of brain methods	Vidal, S., Berra, I., Zulliani, S., Marcos, C. and Pace, J.A.D.	2018	Journal	*ACM Transactions on Software Engineering and Methodology (TOSEM)*
S208	Recommending refactoring solutions based on traceability and code metrics	Nyamawe, A.S., Liu, H., Niu, Z., Wang, W. and Niu, N.	2018	Journal	*IEEE Access*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S209	Beyond technical aspects: How do community smells influence the intensity of code smells?	Palomba, F., Tamburri, D.A., Fontana, F.A., Oliveto, R., Zaidman, A. and Serebrenik, A.	2018	Journal	*IEEE transactions on software engineering*
S210	A large-scale empirical study on the lifecycle of code smell co-occurrences	Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R. and De Lucia, A.	2018	Journal	*Information and Software Technology*
S211	Identifying and prioritizing architectural debt through architectural smells: a case study in a large software company	Martini, A., Fontana, F.A., Biaggi, A. and Roveda, R.	2018	Conference	*European conference on software architecture*
S212	A survey of search-based refactoring for software maintenance	Mohan, M. and Greer, D.	2018	Journal	*Journal of Software Engineering Research and Development*
S213	Empirical evaluation of software maintainability based on a manually validated refactoring dataset	Hegedűs, P., Kádár, I., Ferenc, R. and Gyimóthy, T.	2018	Journal	*Information and Software Technology*
S214	On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation	Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R. and De Lucia, A.	2018	Conference	*Proceedings of the 40th International Conference on Software Engineering*
S215	An interactive and dynamic search-based approach to software refactoring recommendations	Alizadeh, V., Kessentini, M., Mkaouer, M.W., Ocinneide, M., Ouni, A. and Cai, Y.	2018	Journal	*IEEE Transactions on Software Engineering*
S216	Can you tell me if it smells? a study on how developers discuss code smells and anti-patterns in stack overflow	Tahir, A., Yamashita, A., Licorish, S., Dietrich, J. and Counsell, S.	2018	Conference	*Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S217	Analyzing refactoring trends and practices in the software industry	Khanam, Z.	2018	Journal	*International Journal of Advanced Research in Computer Science*
S218	An investigative study on how developers filter and prioritise code smells	Sae-Lim, N., Hayashi, S. and Saeki, M.	2018	Journal	*IEICE TRANSACTIONS on Information and Systems*
S219	Context-based approach to prioritize code smells for prefactoring	Sae-Lim, N., Hayashi, S. and Saeki, M.	2018	Journal	*Journal of Software: Evolution and Process*
S220	Refactoring opportunity identification methodology for removing long method smells and improving code analyzability	Meananeatra, P., Rongviriyapanish, S. and Apiwattanapong, T.	2018	Journal	*IEICE Transactions on Information and Systems*
S221	Improving code: The (mis) perception of quality metrics	Pantiuchina, J., Lanza, M. and Bavota, G.	2018	Conference	*2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*
S222	Barriers to Refactoring: Issues and Solutions	Khanam, Z.	2018	Journal	*International Journal on Future Revolution in Computer Science and Communication Engineering*
S223	Detecting and managing code smells: Research and practice	Sharma, T.	2018	Conference	*Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings*
S224	Causes, impacts, and detection approaches of code smell: a survey	Haque, M.S., Carver, J. and Atkison, T.	2018	Conference	*Proceedings of the ACMSE 2018 Conference*
S225	A quantitative study on characteristics and effect of batch refactoring on code smells	Bibiano, A.C., Fernandes, E., Oliveira, D., Garcia, A., Kalinowski, M., Fonseca, B.,... and Cedrim, D.	2019	Conference	*2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S226	An approach to suggest code smell order for refactoring	Guggulothu, T. and Moiz, S.A.	2019	Conference	*International Conference on Emerging Technologies in Computer Engineering*
S227	Can refactoring be self-affirmed? an exploratory study on how developers document their refactoring activities in commit messages	AlOmar, E., Mkaouer, M.W. and Ouni, A.	2019	Conference	*2019 IEEE/ACM 3rd International Workshop on Refactoring (IWoR)*
S228	Deep learning based code smell detection	Liu, H., Jin, J., Xu, Z., Zou, Y., Bu, Y. and Zhang, L.	2019	Journal	*IEEE transactions on Software Engineering*
S229	A survey on UML model smells detection techniques for software refactoring	Mumtaz, H., Alshayeb, M., Mahmood, S. and Niazi, M.	2019	Journal	*Journal of Software: Evolution and Process*
S230	Machine learning techniques for code smells detection: a systematic mapping study	Caram, F.L., Rodrigues, B.R.D.O., Campanelli, A.S. and Parreiras, F.S.	2019	Journal	*International Journal of Software Engineering and Knowledge Engineering*
S231	A review on search-based tools and techniques to identify bad code smells in object-oriented systems	Kaur, A. and Dhiman, G.	2019	Journal	*Harmony search and nature inspired optimization algorithms*
S232	Code smells analysis mechanisms, detection issues, and effect on software maintainability	Lafi, M., Botros, J.W., Kafaween, H., Al-Dasoqi, A.B. and Al-Tamimi, A.	2019	Conference	*2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*
S233	Toward proactive refactoring: An exploratory study on decaying modules	Sae-Lim, N., Hayashi, S. and Saeki, M.	2019	Conference	*2019 IEEE/ACM 3rd International Workshop on Refactoring (IWoR)*
S234	On the impact of refactoring on the relationship between quality attributes and design metrics	AlOmar, E.A., Mkaouer, M.W., Ouni, A. and Kessentini, M.	2019	Conference	*2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S235	A case study on the effects and limitations of refactoring	Békefi, B.F., Szabados, K. and Kovács, A.	2019	Conference	*2019 IEEE 15th International Scientific Conference on Informatics*
S236	How does object-oriented code refactoring influence software quality? Research landscape and challenges	Kaur, S. and Singh, P.	2019	Journal	*Journal of Systems and Software*
S237	Self-admitted technical debt removal and refactoring actions: Co-occurrence or more?	Iammarino, M., Zampetti, F., Aversano, L. and Di Penta, M.	2019	Conference	*2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*
S238	A large-scale empirical exploration on refactoring activities in open source software projects	Vassallo, C., Grano, G., Palomba, F., Gall, H.C. and Bacchelli, A.	2019	Journal	*Science of Computer Programming*
S239	Reducing the large class code smell by applying design patterns	Turkistani, B. and Liu, Y.	2019	Conference	*2019 IEEE International Conference on Electro Information Technology (EIT)*
S240	Machine learning techniques for code smell detection: A systematic literature review and meta-analysis	Azeem, M.I., Palomba, F., Shi, L. and Wang, Q.	2019	Journal	*Information and Software Technology*
S241	Ranking architecturally critical agglomerations of code smells	Vidal, S., Oizumi, W., Garcia, A., Pace, A.D. and Marcos, C.	2019	Journal	*Science of Computer Programming*
S242	Generating code-smell prediction rules using decision tree algorithm and software metrics	Mhawish, M.Y. and Gupta, M.	2019	Journal	*International Journal of Computer Sciences and Engineering*
S243	Code smells and refactoring: A tertiary systematic review of challenges and observations	Lacerda, G., Petrillo, F., Pimenta, M. and Guéhéneuc, Y.G.	2020	Journal	*Journal of Systems and Software*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S244	A systematic literature survey of software metrics, code smells and refactoring techniques	Agnihotri, M. and Chug, A.	2020	Journal	*Journal of Information Processing Systems*
S245	cASpER: A plug-in for automated code smell detection and refactoring	De Stefano, M., Gambardella, M.S., Pecorelli, F., Palomba, F. and De Lucia, A.	2020	Conference	*Proceedings of the International Conference on Advanced Visual Interfaces*
S246	Are code smell co-occurrences harmful to internal quality attributes? a mixed-method study	Martins, J., Bezerra, C., Uchôa, A. and Garcia, A.	2020	Conference	*Proceedings of the 34th Brazilian Symposium on Software Engineering*
S247	How does incomplete composite refactoring affect internal quality attributes?	Bibiano, A.C., Soares, V., Coutinho, D., Fernandes, E., Correia, J.L., Santos, K.,... and Oliveira, D.	2020	Conference	*Proceedings of the 28th International Conference on Program Comprehension*
S248	Increasing the trust in refactoring through visualization	Bogart, A., AlOmar, E.A., Mkaouer, M.W. and Ouni, A.	2020	Conference	*Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*
S249	Automatic software refactoring: a systematic literature review	Baqais, A.A.B. and Alshayeb, M.	2020	Journal	*Software Quality Journal*
S250	Bad smell detection using quality metrics and refactoring opportunities	Bafandeh Mayvan, B., Rasoolzadegan, A. and Javan Jafari, A.	2020	Journal	*Journal of Software: Evolution and Process*
S251	Refactoring graphs: Assessing refactoring over time	Brito, A., Hora, A. and Valente, M.T.	2020	Conference	*2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S252	When are smells indicators of architectural refactoring opportunities: A study of 50 software projects	Sousa, L., Oizumi, W., Garcia, A., Oliveira, A., Cedrim, D. and Lucena, C.	2020	Conference	*Proceedings of the 28th International Conference on Program Comprehension*
S253	Developer-driven code smell prioritization	Pecorelli, F., Palomba, F., Khomh, F. and De Lucia, A.	2020	Conference	*Proceedings of the 17th International Conference on Mining Software Repositories*
S254	Refactoring test smells: A perspective from open-source developers	Soares, E., Ribeiro, M., Amaral, G., Gheyi, R., Fernandes, L., Garcia, A.,... and Santos, A.	2020	Conference	*Proceedings of the 5th Brazilian Symposium on Systematic and Automated Software Testing*
S255	A longitudinal study of the impact of refactoring in android applications	Hamdi, O., Ouni, A., Cinnéide, M.Ó. and Mkaouer, M.W.	2021	Journal	*Information and Software Technology*
S256	A brief review on multi-objective software refactoring and a new method for its recommendation	Kaur, S., Awasthi, L.K. and Sangal, A.L.	2021	Journal	*Archives of Computational Methods in Engineering*
S257	Toward the automatic classification of self-affirmed refactoring	AlOmar, E.A., Mkaouer, M.W. and Ouni, A.	2021	Journal	*Journal of Systems and Software*
S258	How do Code Smell Co-occurrences Removal Impact Internal Quality Attributes? A Developers' Perspective	Martins, J., Bezerra, C., Uchôa, A. and Garcia, A.	2021	Conference	*Brazilian Symposium on Software Engineering*
S259	Prioritization of code smells in object-oriented software: A review	Kaur, A., Jain, S., Goel, S. and Dhiman, G.	2021	Journal	*Materials Today: Proceedings*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S260	Refactoring practices in the context of modern code review: An industrial case study at Xerox	AlOmar, E.A., AlRubaye, H., Mkaouer, M.W., Ouni, A. and Kessentini, M.	2021	Conference	*2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*
S261	A fuzzy genetic automatic refactoring approach to improve software maintainability and flexibility	Saheb Nasagh, R., Shahidi, M. and Ashtiani, M.	2021	Journal	*Soft Computing*
S262	Behind the scenes: On the relationship between developer experience and refactoring	AlOmar, E.A., Peruma, A., Mkaouer, M.W., Newman, C.D. and Ouni, A.	2021	Journal	*Journal of Software: Evolution and Process*
S263	Understanding code smell detection via code review: A study of the openstack community	Han, X., Tahir, A., Liang, P., Counsell, S. and Luo, Y.	2021	Conference	*2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*
S264	Software refactoring side effects	AbuHassan, A., Alshayeb, M. and Ghouti, L.	2021	Journal	*Journal of Software: Evolution and Process*
S265	Deep analysis of quality of primary studies on assessing the impact of refactoring on software quality	Kaur, S., Kaur, A. and Dhiman, G.	2021	Journal	*Materials Today: Proceedings*
S266	The Prevalence of Code Smells in Machine Learning projects	van Oort, B., Cruz, L., Aniche, M. and van Deursen, A.	2021	Conference	*2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*
S267	Supporting Proactive Refactoring: An Exploratory Study on Decaying Modules and Their Prediction	Sae-Lim, N., Hayashi, S. and Saeki, M.	2021	Journal	*IEICE Transactions on Information and Systems*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S268	A Study of Relevant Parameters Influencing Code Smell Prioritization in Object-Oriented Software Systems	Verma, R., Kumar, K. and Verma, H.K.	2021	Conference	*2021 6th International Conference on Signal Processing, Computing and Control (ISPCC)*
S269	On preserving the behavior in software refactoring: A systematic mapping study	AlOmar, E.A., Mkaouer, M.W., Newman, C. and Ouni, A.	2021	Journal	*Information and Software Technology*
S270	Addressing the trade off between smells and quality when refactoring class diagrams	Barriga, A., Bettini, L., Iovino, L., Rutle, A. and Heldal, R.	2021	Journal	*J. Object Technol.*
S271	An automated extract method refactoring approach to correct the long method code smell	Shahidi, M., Ashtiani, M. and Zakeri-Nasrabadi, M.	2022	Journal	*Journal of Systems and Software*
S272	How do i refactor this? An empirical study on refactoring trends and topics in Stack Overflow	Peruma, A., Simmons, S., AlOmar, E.A., Newman, C.D., Mkaouer, M.W. and Ouni, A.	2022	Journal	*Empirical Software Engineering*
S273	An Empirical Study on the Occurrences of Code Smells in Open Source and Industrial Projects	Rahman, M.M., Satter, A., Joarder, M.M.A. and Sakib, K.	2022	Conference	*ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*
S274	Refactoring for reuse: an empirical study	AlOmar, E.A., Wang, T., Raut, V., Mkaouer, M.W., Newman, C. and Ouni, A.	2022	Journal	*Innovations in Systems and Software Engineering*
S275	Toward Understanding the Impact of Refactoring on Program Comprehension	Sellitto, G., Iannone, E., Codabux, Z., Lenarduzzi, V., De Lucia, A., Palomba, F. and Ferrucci, F.	2022	Conference	*29th International Conference on Software Analysis, Evolution, and Reengineering (SANER)*

ID	*Title*	*Authors*	*Year*	*Category*	*Source*
S276	Code Smell Co-occurrences: A Systematic Mapping	Neto, A., Bezerra, C. and Serafim Martins, J.	2022	Conference	*Proceedings of the XXXVI Brazilian Symposium on Software Engineering*
S277	A severity-based classification assessment of code smells in Kotlin and Java application	Gupta, A. and Chauhan, N.K.	2022	Journal	*Arabian Journal for Science and Engineering*
S278	Exploring the relationship between refactoring and code debt indicators	Halepmollasi, R. and Tosun, A.	2022	Journal	*Journal of Software: Evolution and Process*
S279	Understanding Refactoring Tactics and their Effect on Software Quality	Agnihotri, M. and Chug, A.	2022	Conference	*2022 12th International Conference on Cloud Computing, Data Science and Engineering*
S280	Categorical Analysis of Code Smell Detection Using Machine Learning Algorithms	Bansal, A., Jayant, U. and Jain, A.	2022	Conference	*Intelligent Sustainable Systems*