Software Quality Journal (2019) 27:1069–1148
https://doi.org/10.1007/s11219-018-9424-8

Software Design Smell Detection: a systematic mapping study

Khalid Alkharabsheh[1] · Yania Crespo[2] · Esperanza Manso[2] · José A. Taboada[1]

Published online: 27 October 2018
© Springer Science+Business Media, LLC, part of Springer Nature 2018

Abstract
Design Smells are indicators of situations that negatively affect software quality attributes such as understandability, testability, extensibility, reusability, and maintainability in general. Improving maintainability is one of the cornerstones of making software evolution easier. Hence, Design Smell Detection is important in helping developers when making decisions that can improve software evolution processes. After a long period of research, it is important to organize the knowledge produced so far and to identify current challenges and future trends. In this paper, we analyze 18 years of research into Design Smell Detection. There is a wide variety of terms that have been used in the literature to describe concepts which are similar to what we have defined as "Design Smells," such as design defect, design flaw, anomaly, pitfall, antipattern, and disharmony. The aim of this paper is to analyze all these terms and include them in the study. We have used the standard systematic literature review method based on a comprehensive set of 395 articles published in different proceedings, journals, and book chapters. We present the results in different dimensions of Design Smell Detection, such as the type or scope of smell, detection approaches, tools, applied techniques, validation evidence, type of artifact in which the smell is detected, resources used in evaluation, supported languages, and relation between detected smells and software quality attributes according to a quality model. The main contributions of this paper are, on the one hand, the application of domain modeling techniques to obtain a conceptual model that allows the organization of the knowledge on Design Smell Detection and a collaborative web application built on that knowledge and, on the other, finding how tendencies have moved across different kinds of smell detection, as well as different approaches and techniques. Key findings for future trends include the fact that all automatic detection tools described in the literature identify Design Smells as a binary decision (having the smell or not), which is an opportunity to evolve to fuzzy and prioritized decisions. We also find that there is a lack of human experts and benchmark validation processes, as well as demonstrating that Design Smell Detection positively influences quality attributes.

Keywords DesignSmell · Antipatterns · Detection tools · Quality models · Systematic mapping study

✉ Khalid Alkharabsheh
khalid.alkharabsheh@usc.es

Extended author information available on the last page of the article

1 Introduction

Software quality is one of the most important issues in software engineering, drawing attention from both practitioners and researchers. Developing quality software is important, but preserving or increasing quality during maintenance is even more important. Controlling maintenance processes is crucial and more difficult than other phases of the software development cycle for several reasons: source code size and complexity, amount and frequency of the maintenance tasks, the assortment of tools for controlling, managing, and documenting modifications, lack of staff experience, staff instability, etc. Maintenance processes consume most of the software development cost (Mens and Tourwé 2004). In order to improve and manage system maintainability, appropriate strategies that integrated with experts efforts should be applied.

Software pieces, which are suffering from poor programming, bad design, or other problems in source code or design, should be identified and improved. Identification of these pieces is what we call Design Smell Detection. A Design Smell does not produce compile-time or run-time errors, but it does negatively affect the system quality attributes, such as understandability, testability, extensibility, reusability or maintainability in general (Pérez-García, 2011; Yamashita and Counsell 2013). Design Smells are alerts, refactoring opportunity indicators (Bavota et al. 2015), or even alarm signals about an increment in the Technical Debt that can lead to project failure.

Ward Cunningham in 1992 presented the metaphor known as Technical Debt.[1] Cunningham himself in 2008 relaunched the concept in a series of conferences around the world. Technical Debt is associated with a lack of quality due to, for example, the presence of Design Smells. The evidence that a poor structure in source code or design (which is precisely the definition of a Design Smell) is one of the most important factors contributing to Technical Debt was corroborated in Budgen et al. (2008).

Several terms have been used to describe Design Smells in the literature. Code smell was the first related term, coined by Kent Beck, as a hint that something has gone wrong somewhere in the code, when Beck and Fowler prepared a chapter, subsequently called "Bad Smell in Code," of the Refactoring book (Beck and Fowler 1999, Chapter 3, pp. 83–93). They set out a list of 22 Bad Smells and a description of how to recognize them. The authors even related each Bad Smell with the refactoring catalog (set of steps that can be applied to remove each Bad Smell from code). Robert C. Martin (2003) used the term Design Smells in reference to higher-level smells that cause the decay of the software system's structure. Other relevant terms can be found in the literature, such as Antipatterns (Brown et al. 1998), Disharmonies (Lanza and Marinescu 2006), Design Flaws (Salehie et al. 2006), Design Defects (Moha 2007), Code Anomalies (Wasylkowski et al. 2007), and Design debt (Zazworka et al. 2011; Suryanarayana et al. 2014).

The term Design Smell was defined as a unifying term in (Pérez-García, 2011), in the line of Martin (2003). Consequently, we adopt Design Smell in this paper as a concept similar to Code Smell or Bad Smell as in (Hassaine et al. 2010; Líška and Polášek 2011; Polásek et al. 2012; Suryanarayana et al. 2014; Alkharabsheh et al. 2016a, b), but in a more general sense, covering the whole range of problems related to software structure, i.e., the design part.

Design Smells can be detected in different software artifacts from fine-grained to coarse-grained, including variables, instructions, operations, methods, classes, packages, subsystems,

[1] http://c2.com/doc/oopsla92.html

layers, and their dependencies. We can find several examples of Design Smells affecting different granularity levels, from methods (e.g., Long Parameter List from Bad Smells (Fowler et al. 1999)) to the whole system architecture (e.g., Stovepipe System from Antipatterns (Brown et al. 1998)). This could be one of the reasons why some authors have made further additions to smell terminology.

In the state of the art of Design Smell Detection, we have noticed the lack of systematic mapping reviews. However, the volume of research in the domain has multiplied over the more than 17 years of activity in the field. This has led to the need for critical integration and evaluation of the available research in Design Smell Detection. In our opinion, understanding this area is important because, nowadays, a considerable number of software projects have huge dimensions, so manual Design Smell Detection is not realistic. Problems are latent in code; detection usually occurs very late, and then, solutions are very complex. As a consequence, the software quality is negatively affected and technical debt increases, so redoing the software becomes the most realistic option. We believe, in a comparative perspective, that while refactoring has been extensively adopted by the software industry, Design Smell Detection is far from that reality.

According to our experience, this minor adoption of Design Smell Detection, instead of refactoring adoption, seems contradictory because of the intensive activity in both fields and the intimate relationship between refactoring and Design Smell Detection. A systematic analysis of the state of the art is essential to identify and evaluate the fields that require additional research compared to those in which there is more research available and thus help to define future work with this target.

The methodology followed in this article includes planning (determining the research questions), conditions, and criteria for identifying and selecting primary studies and a final report to gain an extensive overview on Design Smell Detection through answering the research questions. We explore different types of Design Smells, different approaches and strategies in detection, tools, techniques, methods, and their validation, in a broad sense. We follow the same methodology as (Laguna and Crespo 2013) used in their systematic mapping study of software product line evolution and refactoring, which is based on the seminal papers of Kitchenham et al. (2006) and (Kitchenham and Charters 2007). Moreover, we have taken a step forward and apply conceptual modeling techniques to organize the knowledge in the area as a domain model.

The rest of the article is structured as follows: Section 2 comments on related work. Section 3 presents how the mapping study protocol has been planned (research questions, research strategy, classification schema, data extraction, conceptual model). Section 4 presents an overview of the study results obtained at each step of the process described in the previous section and an executive summary of the results. Section 5 presents the detailed study to answer the research questions. Section 6 discusses the findings and their implications. Section 7 presents the threats to the validity of this mapping study. Section 8 presents the conclusions, identifying tendencies and open problems. Appendix A lists the references of all the papers included in the systematic mapping.

2 Related work

Zhang et al. (2011) published a literature review identifying current knowledge concerning the 22 Bad Smells described by Beck and Fowler. They reviewed 319 papers (published from

2000 to 2009) and analyzed in detail 39 articles related to these Bad Smells. They found that some Bad Smells, such as Duplicated Code, have attracted the most attention from researchers, while others, such as Message Chain, receive the least attention. They believe it could be due to the fact that the former have a completely different nature from other Bad Smells, are easy to understand, are widespread in code, and that developers are aware of the problems that can arise in maintenance because of Duplicated Code. They also found that the focus of the research papers can be split into two parts: developing novel tools/methods for Bad Smell detection and improving the current understanding and knowledge of Bad Smell refactoring.

Rattan et al. (2013) presented a systematic literature review of software clone detection (another name related to Duplicated Code detection). This code smell can be understood as the copying and pasting of code fragments during software development, with or without modification. They used the search string "code clone" in the title or abstract of articles. A set of 213 articles were analyzed in detail (published from the initial date of the digital library to 2011). The existing literature about Code Clones was classified broadly into different categories: semantic clone detection and model-based clone detection, empirical evaluation of clone detection, tools/techniques, clone management, etc. They found several contradicting studies stating that Code Clones are harmful and others stating the contrary.

Rasool and Arshad (2015) published a review study of the tools and techniques used for what they call code smell mining. The study period, from 1999 to 2015, included 42 articles reviewed and analyzed in detail. The authors focused on Fowler's 22 Bad Smells only and classified the tools and techniques based on their detection methods. They then compared a set of tools to detect four Bad Smells. The results found that different tools and techniques rendered different results.

Fernandes et al. (2016) performed a review-based study summarizing and comparing the available Design Smell Detection tools. They analyzed 107 papers relevant to detection tools published from 2000 to 2015. They found 84 tools able to detect different types of Design Smell, support programming languages, such as Java, C, C++, and C#, and which could be produced as commercial or open source release. They then conducted a comparative study between four detection tools in order to detect two of Fowler's Bad Smells, and they found (as opposed to (Rasool and Arshad 2015)) a high agreement between tools.

Singh and Kaur (2017) conducted a systematic review of refactoring concerning the code smells and Antipatterns. Two hundred thirty-eight papers were reviewed and analyzed from the initial dates of the digital libraries to September 2015. The authors focused on identifying the current status of refactoring, types of approaches, and tools. They revealed six types of approaches for performing refactoring, including automatic method, metric-based method, traditional method, visualization-based technique, semi-automatic method, and empirical studies. Also, different tools were used for detecting Design Smells.

Recently, Gupta et al. (2017) published a literature review to study the code bad smells in Java code. Sixty papers were analyzed in depth from 1999 to 2016. The authors focused, on the one hand, on the detection techniques and the correlation between them and, on the other hand, on identifying Design Smells that require more investigation from the research community. In this study, we have discarded the term "code clone" from the search string, because we assume that it is sufficiently covered by (Rattan et al. 2013) and any interested researcher can be redirected to read it. Our research interest focuses on Design Smell Detection in a broad sense.

This paper presents a systematic mapping that differs from the previously mentioned studies by focusing, on the one hand, on all types of Design Smell (Bad Smell, Antipatterns, Disharmonies, etc., using Design Smell as a unifying term) and, on the other hand, on the detection activity and some other related activities, such as specification, correcting (refactoring), and prioritization. Consequently, the main goal of our systematic mapping study is to collect and organize the knowledge on Design Smell Detection in general (approaches, tools, techniques, datasets, quality factors, etc.) and not just tools, as is the case of some related work. Table 1 shows a brief comparison between our work and the previous works regarding the goal, period of study, number of selected papers, and the total number of Design Smells (DS) that are covered in the study.

3 Systematic mapping methodology

Several methods have been performed by researchers for literature reviews, such as experiments, surveys, case studies and systematic reviews. In our study, to address the current status of the Design Smell Detection domain, we conducted a comprehensive systematic mapping study. In this section, we describe the phases of the method we use and how the mapping study has been planned.

The methodology includes defining and answering the research questions regarding the topic of research, defining the research strategy (scope of information, search string, period of study, search method, inclusion, and exclusion criteria), classification of included works and the synthesis of the results. Figure 1 presents the flowchart of our systematic mapping process. The full details of the steps in the chart are described in the following subsections.

3.1 Research questions

This study aims to organize the knowledge on the software Design Smell Detection domain. To clarify the scope of knowledge, several concepts should be identified before formulating the research questions. The core concepts identified are mainly focused on the type of detected Design Smell, the different approaches in detection, the availability of detection tools, the validation evidence for the proposed approach, and the influence on quality factors. The goal of the study is then broken down into five research questions (RQs):

Table 1 Related works summary

Reference	Goal	Period of study	# paper	# DS
Zhang et al. (2011)	Current knowledge of code smells	2000 to 2009	39	22
Rattan et al. (2013)	Software clone detection	1988 to 2011	213	1
Rasool and Arshad (2015)	Tools and techniques for mining code smells	1999 to 2015	42	22
Fernandes et al. (2016)	Detection tools	2000 to 2015	107	61
Singh and Kaur (2017)	Current status of refactoring with respect to code smells and antipattern identification	1990 to 2015	238	16
Gupta et al. (2017)	Detection techniques and correlation between them	1999 to 2016	60	22
Our work	Collect and organize the knowledge on Design Smell Detection (approaches, tools, techniques, quality factors, etc.)	2000 to 2017	395	662

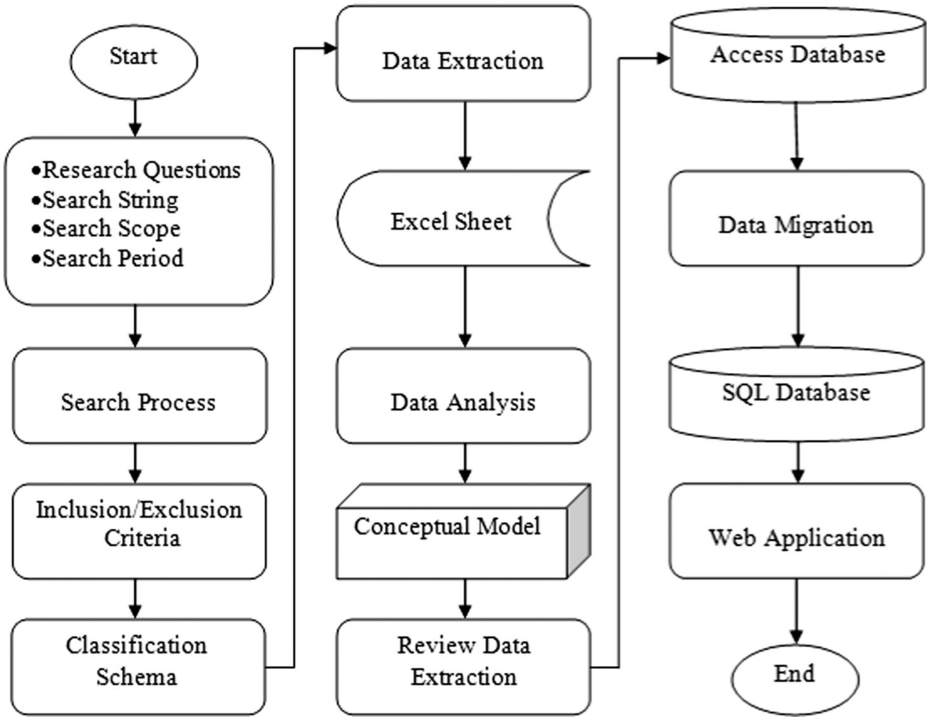


Fig. 1 Systematic mapping process performed for this paper

- *RQ1* Which types of Design Smell are detected?

This question can be completed with two subquestions:

(a) Which types of Design Smell are dealt with in papers and tools?
(b) Which types of Design Smell are more frequently detected in software?

Different types of smell have been found in the literature. We want to identify which Design Smells had gained the focus of researchers, determine which types of smell are widely detected in applications and measure the frequency with which each Design Smell appears. This information can help to reduce the gap between Design Smell Detection research, identifying the directions of future works and planning educational actions to improve developers' programming skills in specific aspects.

- *RQ2* What approaches have been proposed to detect Design Smells in software?

Several Design Smell Detection approaches have been proposed to detect Design Smells manually, semi-automatically, and automatically. We want to know which approaches are used, how often each approach is used, if there exist compound approaches working together, and the influence of these approaches on smell detection. This information helps to identify differences in results regarding the approach and new trends.

- *RQ3* Which (prototype) tools have been used to detect Design Smells in software?

Different (prototype) tools have been developed to detect Design Smells in software. The goal of this question is to know which tools are used and how often. Also, if the tool depends on a particular type of Design Smell, representation model, or software artifact. We can map the type of smell, the approach, and the kind of artifact with the tools.

- *RQ4* Is the (prototype) tool validated by expert or benchmark or by comparison with other tools? Is the tool or strategy assessing results and measuring performance, precision, and recall?

This question aims to assess the results that the proposed tool or prototype has produced. This assessment can be done through human experts, benchmarking, or comparing the results with different previous results from other tools/prototypes. We want to know which types of validation evidence were used, such as case studies, experiments, surveys, and which type of performance evaluations were used (precision, recall, false positive, false negative, etc.). We are also looking for whether there are guidelines from human experts to enhance the results of automatic detection provided by tools.

- *RQ5* Is Design Smell Detection related to the quality attributes of a quality model?

This question explores whether Design Smell Detection is useful to solve a particular quality problem, determining internal or external attributes from the quality model affected. It could be interesting for developers if Design Smell Detection is related to quality factors. Hence, they can know whether the effect of detecting, and later removing, a Design Smell will have an impact on quality factors and can also help in prioritizing them.

3.2 Search strategy

The research strategy addresses the search string we propose, the scope of the research, the period of search, the search method, and the inclusion/exclusion criteria we identify. In order to have a broad coverage of the state of the art, the search string should be carefully defined.

3.2.1 Search string

We constructed the search string according to the main goal and the research question. The strings should be simple, so as to achieve multiple results and cover the topics exactly. We use the OR Boolean operators to link the main terms and their synonyms. The final search string is:

(“Design Smell” or “design-smell” or “bad smell” or “bad-smell” or “code smell” or “code-smell” or “design defect” or “design-defect” or “design flaw” or “design-flaw” or “antipattern” or “anti-pattern” or “disharmony” or “disharmonies”) and (“detection” or “detecting” or “identification” or “identifying” or “finding” or “empirical”)

In order to use the string composed with the AND/OR Boolean operators, we used the available advanced search in each database.

3.2.2 Search scope

To obtain a high coverage of finding the relevant studies and publications, six electronic databases were included in the search scope. According to Dyba et al. (2007), Laguna and Crespo (2013), Novais et al. (2013), and Vasconcellos et al. (2017) in their systematic mapping studies, these electronic databases are the most popular and efficient to conduct systematic studies in the context of software engineering and reengineering. They are strongly recommended for searching widely. These databases are Science Direct (www.sciencedirect.com), Scopus (www.scopus.com), Web of Science (www.isiknowledge.com), IEEE Xplore (https://ieeexplore.ieee.org/Xplore/home.jsp), ACM Digital Library (www.acm.org), and Springer (www.springerlink.com). In order to find more useful studies that do not appear in the standard search process, the Google Scholar (https://scholar.google.com/) database was included, despite the fact that the search results tend to be repetitive with the search results of the selected databases.

3.2.3 Search period

The time period covered all related papers published in books, journals, conferences, symposiums, and workshops from January 2000 up to the end of December 2017. Design Smell Detection became popular around the beginning of 2002. Nevertheless, the seminal paper on Smells is considered to be Fowler and Beck's chapter on Bad Smells in 2000. Therefore, we chose this date as the starting time. January 2018 is the time that we started working on this systematic mapping.

3.2.4 Search method

We used both automatic and manual searches in the study. In the automatic search, each electronic database, provided by a search engine, checks the search terms against the metadata, including the title, abstract, and keywords of each paper in the database. In the manual search, on the one hand, we looked inside each conference and workshop report listed in the database that was related to the search string for the papers closely relevant to our mapping study, but which did not appear in the automatic search. On the other hand, we adopted the snow-balling technique (Wohlin 2014), which permits the reference list of the final determined studies from the automatic search (electronic databases) to be analyzed. Hence, we could include them in our mapping study.

3.2.5 Inclusion/exclusion criteria

The selection criteria aim to find all relevant papers in our topic of systematic mapping as follows.

- Inclusion criteria:
- Papers published from January 2000 to December 2017
- Full paper published in conference, journal, workshop, symposium, and chapter in a book
- Papers where the search string appears in title, abstract, and keywords
- Exclusion criteria:
- Papers that do not relate to search strings once their text was explored manually
- Reports, position papers, PhD theses, research proposals, projects
- Duplicated papers of the same study in different versions, journals, conferences, and workshops

- Papers not having their full text available
- Papers not written in English

Each selected study must satisfy all inclusion criteria and might not satisfy any of the exclusion criteria. Despite the inclusion/exclusion criteria to select the relevant papers, we still found papers that do not answer the research questions, because the keywords on the search string may have different meanings or were used in studies that are beyond the Design Smell Detection topic; for example, the word "smell" can be related to sensors. A cursory reading is required to obtain the final relevant papers, following which the number of papers decreased significantly, as can be seen in Fig. 2.

Figure 2 shows the filtering process after searching in the electronic databases and obtaining the primary studies. The first search process returned 3962 papers, discarding duplicated papers reduced the list to 2643. Discarding non-English, non-full text paper reports, position papers, PhD theses, and proposals reduced the list to 2476. The decision on excluding PhD theses could be controversial. We decided to exclude PhD theses because we found most of the theses which are relevant to the search string include a list of publications (Inproceedings, journal) already included in this work. Furthermore, we think the final number of selected studies is enough to identify the current knowledge of the Design Smell Detection field.

Finally, after exploring the text manually based on title, abstract and keywords, to discard non-related papers (not relevant to any of the research questions), we obtained 328 selected papers, and after the snow-balling, we ended up with 395 relevant papers, which represent 10.7% of the initial 3962. These papers are listed in appendix A.

3.3 Classification schema

According to (Kitchenham and Charters 2007), in phase two of the systematic literature review (SLR), the quality assessment process of the studies is used for analyzing and assessing the selected papers to be included in the data extraction and reporting process. Classifying the papers in facets is a nice base for answering research questions. Each facet is defined by means of different related keywords. As a result of the cursory reading, a set of facets can be determined into which the papers can be classified. In our case, the facets were also inspired by the classification framework for Design Smell Management, based on features proposed by Pérez, Moha, and Mens in Pérez et al. (2011).

Table 2 shows the facets we originally defined from this process and the selected keywords we used to categorize the selected papers in the field of study.

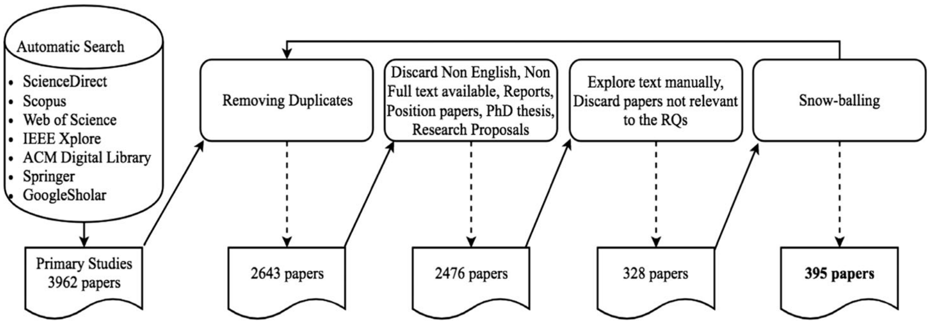

Automatic Search
• ScienceDirect
• Scopus
• Web of Science
• IEEE Xplore
• ACM Digital Library
• Springer
• GoogleSholar
Primary Studies 3962 papers
Removing Duplicates
2643 papers
Discard Non English, Non Full text available, Reports, Position papers, PhD thesis, Research Proposals
2476 papers
Explore text manually, Discard papers not relevant to the RQs
328 papers
Snow-balling
395 papers

Fig. 2 Flowchart of the search and filtering process

Table 2 Facets and keywords initially used to categorize the selected papers in the fields of study (inspired in Pérez et al. 2011)

Category facet	Keywords
Main focus	Design Smell Definition, Design Smell Detection, and other related activities (refactoring, prioritization, etc.)
Validation evidence	Tool support, case study, industrial experience, empirical study, experiment
Target artifact	Requirements, models (representation), code/design
Development paradigm of the target artifact	Object-oriented (C++, C#, java), Aspect-oriented (AspectJ), Procedural (C), Software Product Line (SPL)
Type of Design Smell	Disharmony, Code Smell, Antipattern, Bad Smell, Design Defect, Design Flaw
Level of Design Smell	Low level, High level, Architectural level
Scope of Design Smell	Variable, method, instruction, class, package, subsystem

3.4 Data extraction

One of the most important aspects of systematic mapping is extracting data from the relevant papers after the final search process is completed. A data extraction form is used to extract information relating to the research questions and with the facets stated in Section 3.3. The following items were included in the data extraction form:

- Paper reference data: title and year, author's name and email, author's institution
- Publication data: type (conference, journal, workshop, symposium, chapter in a book), characteristics (full name, acronym, series, publisher)
- Facets defined in Section 3.3: type of Design Smell detected and their description, scope of Design Smell, level of Design Smell, type of target artifact, development paradigm of the target artifact
- Validation evidence conducted by authors of the papers included in the study was also collected in terms of the type of validation, indicators of performance evaluation (precision, recall, false positive, false negative, etc.)
- Regarding the validation, we found it useful to extract information about the following: name, version of the software used in validation, source of the software used in validation (open source software from SourceForge, GitHub, or other public repositories, proprietary software from industry, some benchmark definition if any), URL (if any), some metrics for characterization of the software used in validation: lines of code, number of packages, classes, methods, and total number of Design Smells detected in software (if any), the count of each type of detected Design Smell in the software (if any), and implementation language of the software used in validation.

After the final search process, the data extraction procedure can be summarized as follows to extract the required information:

- Reviewing in depth the abstract and conclusions of each article
- If the extracted information is not found in the abstract and conclusion, the full paper should be read in detail

In order to answer some of the research questions, and after the first data extraction, we introduced extra information items in the data extraction form. These items can be very useful to organize knowledge in this field. Hence, we iterated on the data extraction in order to fulfill all this information.

- Quality model and quality factors regarding the Design Smell Detection (maintainability, reliability, etc.), if mentioned or analyzed
- Approach
- Techniques (algorithms, heuristics, strategies)
- Degree of automation
- Related activity with Design Smells (specification, detection, etc.). This was introduced in order to have information not just on the main focus (already collected in the first stage), but also on other activities tackled in the paper
- The internal representation of the software under analysis (Graph, AST, etc.)
- Related automatic Design Smell Detection tool (name, URL, free, open source, supported language, automation)
- Regarding the presence of tools in papers, we collected the kind of presence (i.e., introducing the tool for the first time, comparing tools, improving an existing tool, etc.) and the list of detected Design Smells.

In this step, we used an Excel spreadsheet to document all items obtained from the extraction process. Then, we analyzed the extracted data and applied domain modeling techniques in order to obtain a conceptual model and produce a domain model of Design Smell Detection.

The accomplished domain modeling makes the task more systematic and provides information for the quantitative and qualitative analysis of the domain of knowledge as described in the next subsection. The domain model was the base for relational database design. We designed a preliminary Access database[2] as a prototype to store all items obtained from the extraction process. After that, a full review of the extracted data was done on the Excel spreadsheet in order to populate the Access database and to check whether some information was missing or misplaced, according to the obtained model. A data migration process was then applied from the Access database to a fully revised MySQL database.

Finally, the MySQL database was included in a web application with a twofold purpose. On the one hand, it is a nice way to have all this knowledge available for the community. On the other hand, we decided to design the website to be a collaborative application where researchers can comment, propose a different classification for existing data, and introduce new knowledge on future work on this topic.

3.5 Conceptual model of the knowledge domain

Figure 3 shows the conceptual model represented as a UML class diagram. This conceptual model describes the Design Smell Detection domain of knowledge as an evolution of the previously described facets and the results of several iterations on the data extraction

[2] Access Database and Excel spreadsheet: https://gitlab.citius.usc.es/DesignSmell/DesignSmellDetectionSystematicMapping

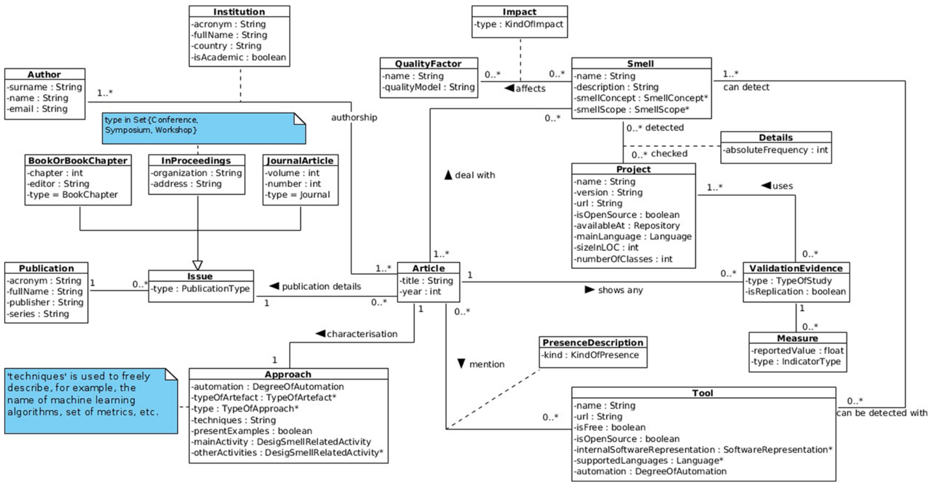


Fig. 3 Design Smell Detection knowledge as a conceptual model

process. According to the lessons learned from analyzing the extracted data, we proceeded to apply the domain modeling techniques. As the figure shows, there is a corner in the model devoted to the article and authors. This part was modeled in such a way that it takes into account the cases where the same author changes, over a period of time, from one institution to another.

Despite the fact that 1 to 1 associations are not considered a good practice in domain modeling, we separated both Article and Approach into two concepts, in order to have a kind of descriptive card of the Article in the Approach. As can be seen in the attributes of Approach, we used the enumerated types shown in Fig. 4. The type of some of the attributes is marked with an asterisk ("*"), signifying in conceptual modeling that there can be more than one value. For instance, an article can present an approach that combines different types of approaches (e.g., metric-based and machine learning based) or can deal with different types of artifacts (e.g., source code and test cases).

Articles can mention tools. Each mention of a tool in an article can be characterized in terms of the values that the enumerated type KindOfPresence represents, such as introducing the tool for the first time, improving an existing tool, comparing the tool with other tools, and reviewing some tool in depth.

Tools are described in terms of some plain data modeled as attributes, but also by means of the association with the Smells the tool can detect. The attribute called "automation" seems to be duplicated in Tool and Approach, but this is intentional. The degree of automation of a tool is considered different from the degree of automation of an approach described in an article that does not mention any particular tool or can even mention several tools in a comparison or related work revision.

Smells deserve a concept in the model with attributes, for instance, describing the scope of the smell (such as package, class, method, …). We also considered it important to represent the Concept the authors use to refer to a particular smell. We observed previously that authors can use the terminology Code Smell, Bad Smell, Design Flaw, …, and we wanted to analyze this and make some clarifications.

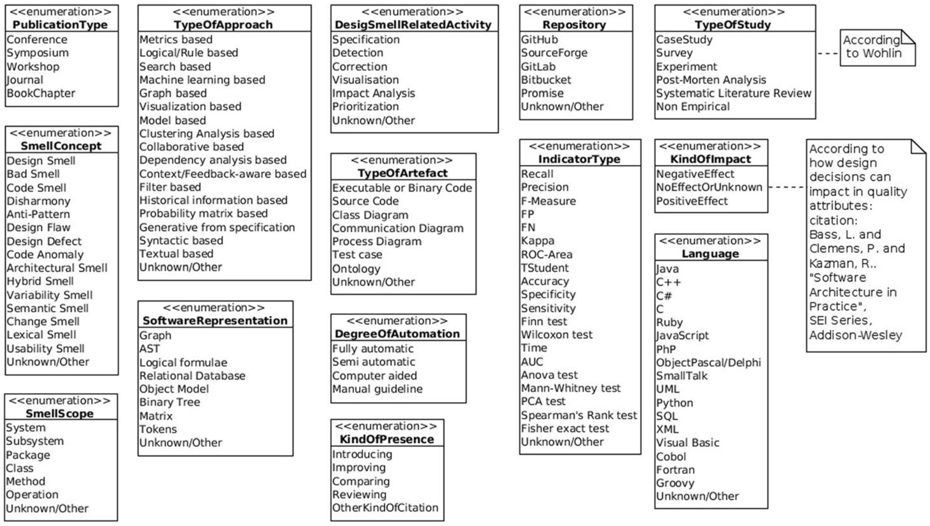


Fig. 4 Enumerated types as a result of evolving facets

In order to answer some of the research questions, such as RQ4, we model whether an article shows any validation evidence. If this is the case, we describe, according to Wohlin et al. (2012), whether the authors conducted a case study, a survey, an experiment, etc. There are also projects associated with the validation evidence.

Finally, RQ5 is concerned with quality factors. In the model, we captured the possibility of relating the detection of some smells with certain quality factors. This relation is characterized in terms of the impact the smell has on the quality factor. We considered the possibility that some smells can relate to more than one quality factor, and even whether, in some cases, the presence of the smell negatively affects the quality factor (which is the intuitive case), but also whether it can affect it positively (for example, the presence of the smell improves efficiency, but hinders maintainability).

Projects are used in case studies, experiments, and so on. Projects are characterized by the attributes showing the project domain, line of codes, and the number of classes. We also describe, on the one hand, which Smells are detected in those projects used in validation and, on the other, which indicators are presented as results of the validation.

Once the model was considered as definitive, a relational database design was obtained, and the physical design of a MySQL database was done. Then, we carried out data migration from the original Access database containing the data extracted at the first stage and completed it with the new data required to fulfill the domain of knowledge model. The data were double checked in order to guarantee the quality of the data extracted and stored in the database. The purpose of the final migration to the MySQL database was to obtain a web application providing the knowledge in this domain, which would then be put in the hands of the community, as mentioned before.

In order to give a descriptive analysis of the collected data and to answer the research questions, several SQL queries were defined and the results were processed, either with Excel and R tools, or with the tool ConExp,[3] which was used to obtain the Galois lattice when formal concept analysis (FCA) is applied to discover some structure and relationships in the data.

[3] http://conexp.sourceforge.net/

4 Descriptive analysis

We conducted the mapping study according to the steps described in Section 3. After the search, filtering, selection, and data extraction procedures, we have a global view of the investigated field. Firstly, we present an overview of the results based on different aspects of the mapping study. Then, we analyze the data from the selected papers through statistics and FCA operations to answer the research questions. In this section regarding descriptive analysis, we have focused on the most used approaches and tools, as well as the most detected Design Smells. For more details on the rest of the study results, all extracted information is available on the web application[4] that is connected with the fully populated database at https://smellswisdom.herokuapp.com/.

4.1 Overview of results

In the following sections, we describe the results regarding study scope, study selection, and the demographical directions (time period, publication type, geographical area).

4.1.1 Search scope results

Table 3 shows a total of 3962 papers as the result of the first search process, the number of papers per electronic database, and also the percentage that it represents. It can be observed that Springer, Scopus, and ACM DL returned the largest sets of papers. In these databases, we found many papers from other areas that were not related to our study. The IEEE Xplore database returned just 173 papers, but a majority of them were relevant to the study, so this was, as a result, more effective as compared to the remaining databases. Regarding the Google Scholar database, the number of relevant papers was 322, in which all the returned papers were repeated in the six main databases.

4.1.2 Time period results

Figure 5 presents the distribution of relevant papers over the time period from 2000 to 2017. During the year 2000, no papers were published in journals and proceedings, only a book chapter. In fact, this book chapter is not the result of the systematic mapping search. It was included manually. It represents the seminal work of Kent Beck and Martin Fowler (Fowler et al. 1999), introducing the term Bad Smell, describing the hints for detecting Bad Smells and relating them with Refactorings. This chapter was included as a reference.

From 2000 to 2003, the number of published papers is less than 5 per year, increasing slowly. As Fig. 5 shows, from 2004 to 2009, the increase is much clearer. From 2010 until the end date of this study (December 2017), there has been a big leap compared with the years before 2009, with a peak in the last 3 years, specifically in 2016.

4.1.3 Publication type results

The selected 395 papers were published as a journal, proceeding papers (conferences, workshops, and symposiums), or book chapters. Figure 5 shows the distribution of selected papers over the publication types and years. Approximately 76% (301 studies) of the selected papers

[4] A cloud platform supporting different programming languages that is used as a web application deployment model.

Table 3 Distribution of selected studies over electronic database

DB source	No. of papers	Ratio	Relevant papers	Effectiveness
Science Direct	313	7.9%	19	6.1%
Scopus	845	21.3%	280	33.0%
Web of Science	372	9.4%	176	47.3%
IEEE Xplore	173	4.4%	142	82.1%
ACM DL	696	17.6%	141	20.3%
Springer	1210	30.5%	42	3.5%
Google Scholar	353	8.9%	322	91.2%
Total	3962	100%		

were published in proceedings as follows: (200) conferences, (64) workshops, and (37) symposiums, and 22% (89 studies) of the selected papers were published in leading journals, while only 1% (5 studies) were publications in book chapters.

Figure 6 shows the proceedings that published five or more papers included in this study, and Fig. 7 shows the journals that published three or more papers. Table 4 shows the corresponding full name of the proceedings/journal acronym. As can be seen, the (ICSE) conference, published by ACM/IEEE, has the highest number of selected publications (17 studies) compared with other proceedings. More than 80% of the relevant papers in proceedings are covered by ICSE, CSMR, ICSM, ICPC, ICASE, ICSME, WCRE, SAC, and SANER. On the other hand, more than 80% of the relevant papers published in journals are covered by (TSE, JSS, IST, ESE, SEN, and SQJ). The quality quartile ranking (JCR) for the selected papers ranged from Q1 to Q3 for all journals along the study period as follows: Q1 (17 studies), Q2 (13 studies), Q3 (5 studies). Only four papers are publications in book chapters.

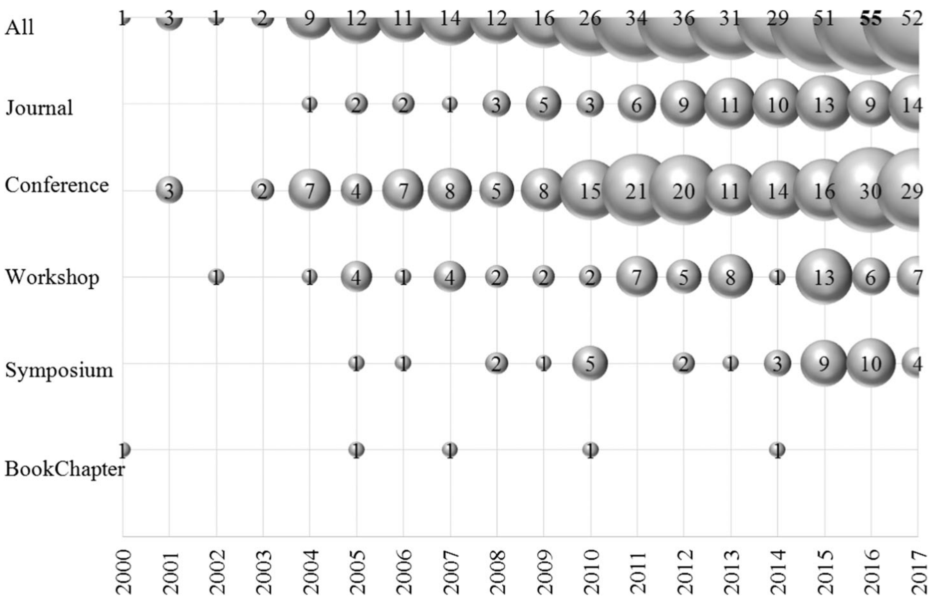


Fig. 5 Distribution of relevant papers over time period per publication type

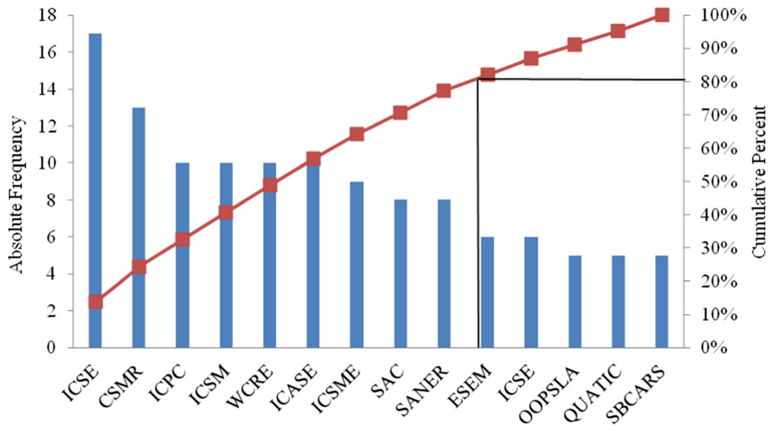


Fig. 6 Distribution of papers in proceedings (having published three or more papers included as relevant in this study)

4.1.4 Authors and institutions results

The authors of the relevant papers in the study are from 284 institutions in 48 different countries. Table 5 shows the distribution of the number of institutions and authors by country. Most of the authors (94.5%) were from the academic sector, while (5.5%) were from industry. In some cases, authors from the academic and industry sectors worked together to publish papers. There are 28 collaborative academic/industry papers in the study: [S21, S34, S37, S55, S68, S85, S107, S113, S142, S182, S189, S193, S202, S211, S233, S239, S240, S255, S262, S263, S274, S284, S302, S329, S383, S389, S391]. The highest number of articles were published by Yann-Gauel Guéhéneuc (27), Francesca Arcelli Fontana (23), Marouane Kessentini (18), Naouel Moha (15), and Houari Sahraoui (12). Table 5 shows the distribution of the number of institutions and authors over the countries. The highest number of articles were published by authors from the Université de Montréal in Canada (30 papers) and the Università degli Studi di Milano-Bicocca in Italy (21 papers).

More than half of the papers were by authors of the same institution (207). Hence, the remaining 188 papers were written in collaboration across institutions.

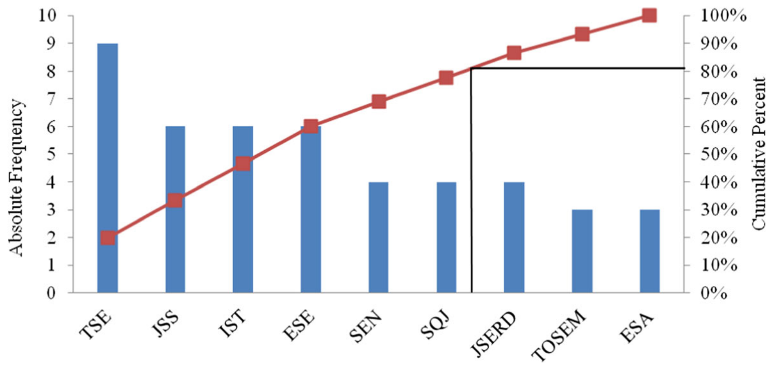


Fig. 7 Distribution of papers in journals (having published three or more papers included as relevant in this study)

Table 4 List of proceedings/journals acronym and the corresponding full name

Acronym	Full name	Acronym	Full name
ICSE	International Conference on Software Engineering	QUATIC	International Conference on the Quality of Information and Communications Technology
CSMR	Conference on Software Maintenance and Reengineering	SBCARS	Brazilian Symposium on Software Components, Architectures and Reuse
ICASE	International Conference on Automated Software Engineering	ESA	Expert Systems with Applications
ICSM	International Conference on Software Maintenance	TOSEM	Transactions on Software Engineering and Methodology
SAC	Symposium on Applied Computing	JSERD	Journal of Software Engineering 2Research and Development
WCRE	Working Conference on Reverse Engineering	TSE	Transactions on Software Engineering
ICPC	International Conference on Program Comprehension	JSS	Journal of Systems and Software
OOPSLA	Conference on Object-Oriented Programming Systems, Languages, and Application	ESE	Empirical Software Engineering
ICSME	International Conference on Software Maintenance and Evolution	SQJ	Software Quality Journal
ESEM	International Symposium on Empirical Software Engineering and Measurement	IST	Information and Software Technology
SANER	International Conference on Software Analysis, Evolution, and Reengineering	SEN	Software Engineering Notes

There are several examples of collaborations in the same country, but also from different European countries (35 papers), from different institutions in North America (USA and Canada: 6 papers), from different institutions in South America (Brazil and Argentina: 3 papers), from different Asian countries (Japan and Qatar: 1 paper). Singular papers (39, which represents 10%) were produced as cross-continental collaborations, namely: [S5, S11, S22, S24, S51, S62, S72, S81, S82, S85, S95, S96, S106, S107, S113, S116, S118, S134, S135, S137, S151, S154, S155, S159, S160, S161, S193, S40, S238, S218, S264, S301, S314, S315, S316, S318, S339, S378, S381].

4.1.5 Key terms in search string

The search terms appear clearly in the title of 284 articles and the keyword terms of 226 articles. Most authors used the concepts "Antipattern" and "Code Smell" to describe Design Smells. More details on this are presented in Section 5.1.1.

Figure 8 shows a cloud of tags produced using https://tagcrowd.com/. Tagcrowd was configured to show words appearing more than 5 times in a text, grouping similar words in English e.g., families of words such as learn, learned, learning in a single word; ignoring common English words, e.g., prepositions, articles, etc., and removing some other words, such as "based" or "towards"). The aim of showing this cloud of tags is to offer a nice picture of the concepts that were relevant for authors when titling their papers in this domain of knowledge.

Table 5 Distribution of the number of institutions and authors by country

Country	Institution	Author	Country	Institution	Author	Country	Institution	Author
USA	34	95	Poland	5	11	New Zealand	2	2
Germany	24	58	Austria	5	18	Ireland	2	4
Brazil	23	87	Turkey	5	7	UAE	1	1
India	19	46	Netherlands	4	17	Egypt	1	3
Canada	16	63	Spain	4	8	Nigeria	1	1
China	14	50	Pakistan	4	8	Bangladesh	1	3
France	14	41	Sweden	4	8	Bolivia	1	1
Italy	11	35	Taiwan	3	8	Czech Republic	1	1
Thailand	9	18	Finland	3	3	Iceland	1	1
UK	8	21	Korea	3	6	Iran	1	1
Portugal	7	14	Argentina	3	18	Saudi Arabia	1	1
Switzerland	7	22	Romania	3	15	Malaysia	1	3
Japan	7	26	Australia	2	3	Qatar	1	1
Norway	7	8	Slovakia	2	8	Singapore	1	2
Tunisia	6	11	Indonesia	2	3	South Africa	1	4
Greece	4	25	Belgium	2	4	Luxembourg	1	2

5 Answering the research questions

5.1 RQ1

Which types of Design Smell are detected?

In order to answer the first research question, we tackled the answer from different perspectives: the set of concepts and terms that authors used to describe problems in software structures (code and design); the scope of Design Smells dealt with in the studies; or the most mentioned and detected Design Smells in the selected papers. The information was extracted according to the conceptual model from the "deal with" between the concepts "Article" and "Smell," the attributes "smellConcept" and "smellScope" of "Smell," and the attribute "absoluteFrequency" of the association class "Details" that stores information on the association between the concepts "Smell" and "Project" in which each smell is detected.

5.1.1 Design Smell concepts

From our previous experience, we have noticed the variety of terms referring to Design Smells, so we have introduced them all in the search string. During the data extraction process, we confirmed that several terms in the state of the art were used to describe similar concepts. Figure 9 presents the distribution of different terms for Design Smells in the selected papers.

analysis android antipatterns applications approach architectural assessing automatic
bad code correction defects design detection effectiveness
empirical evaluation exploratory flaws god identifying impact improving investigating java method
metrics mining model object-oriented patterns performance prioritization problems programming
quality refactoring relations relationship rule smells software specification strategies
study systems techniques tool uml visualization

Fig. 8 Cloud of tags produced from joining the titles of the relevant papers

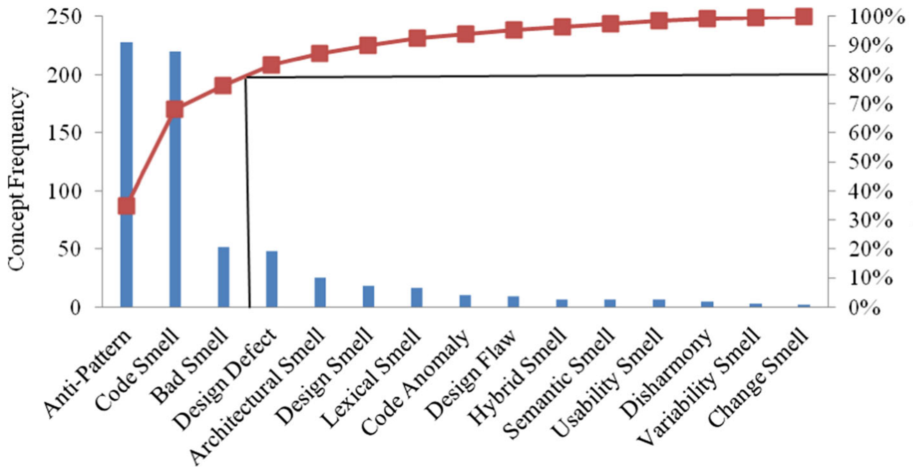


Fig. 9 Classification of Design Smell concepts based on the terminology

As can be seen, more than 80% of the smells cited or detected in the selected papers were mentioned by authors as being Antipattern, Code Smell, or Bad Smell. Nevertheless, there is a consistency problem in the terminology used by authors. The same Design Smell was described by several different terms. Some examples of this are The Blob, which is mentioned in different papers as Antipattern or Design Defect, and Data Class, which is mentioned as Code anomaly, Design Flaw, Bad Smell, etc.

In the relevant papers obtained, varying amounts of effort were made to classify Design Smells. The authors of [S117, S131, S208, S219] used different classifications.

In [S208], the authors proposed a classification of Design Flaws in three categories: Structural, Behavioral, and Architectural flaws. These categories can intersect with each other. In [S117], Design Defects are classified in the same way as Design flaws in [S208]: Structural, Behavioral, and Semantic smells. According to the study, Data Clumps and Long parameter List are considered Structural, Comments are considered Semantic, and God Package, God Class, and Feature Envy are considered Behavioral Design Defects.

In [S136], the authors used the Design Smell concept to describe Code smells and Antipatterns. They classify Design Smells based on the level of granularity as Inter-Class and Intra-Class. In addition, at each level, the Design Smells are divided into three groups according to their characteristics: Structural, Lexical, and Measurable. Functional Decomposition, The Blob, Duplicate Code, and Message Chain are considered to be Inter-Class, while Data Class, Large Class, Spaghetti Code, and Swiss Army Knife are considered Intra-Class Design Smells.

In [S219], the authors introduce a view of different Design Smell concepts. In their study, Bad Smells were the most general concept, which involved both Code and Architectural Smells. Hybrid Smells combine characteristics from both Code and Architectural smells. Finally, the term Variability Smells was specific for software product lines as a subset of Hybrid Smells, combining characteristics of both Code and Architectural smells.

According to (Brown et al. 1998), in their book, Antipatterns are classified into three categories: Code development, Architectural, and Management. In this context, the Management Antipatterns are not taken into account as they are not related to Design Smells.

Another classification of smells we found is given by (Pérez-García, 2011) according to the problem level. Smells are categorized as low-level and high-level smells.

The low-level smells are related to particular problems in the code such as Large Class, Long methods. The high-level smells are related to more complex problems that may be detected in the structure and code, such as The Blob or StovePipeSystem Antipatterns. Some of the low-level smells can be equivalent to the Code Smell classification. Some of the high-level smells can be equivalent to the Architectural Smells classification. Finally, some other can be a kind of Hybrid Smells.

As shown, some of these classifications have in common the ability to recognize what the authors call "Code" and "Architecture" categories. It is also interesting to consider the possibility of a "Hybrid" category. In the light of the above, as part of the conceptual modeling, we intend to organize the terminology used in the different studies included. Figure 10 shows this organization, which cannot be considered a taxonomy because it is based on the terminology and description used by the authors of the different papers. Design Smell is the most general concept that unifies all other smell types.

In the figure, Design Smells are classified into three main types: Code Smells, Hybrid Smells, and Architectural Smells. Code Smells are defined in the implementation (lower) level and include what the authors call in their papers Bad Smells in Code, Code Antipatterns, Code Anomalies, Lexical smells, or Change smells. Architectural Smells are defined in the architecture (higher) level (components, connectors, styles, packages, subsystems, communications) and involves what the authors call in their papers Architectural Antipatterns and Architectural Bad Smells. The Hybrid Smells combine the previous two types: Code and Architectural Smells. This category includes what authors call in their papers Design Defect, Disharmony, Design Flaw, Variability Smell, and Usability Smells.

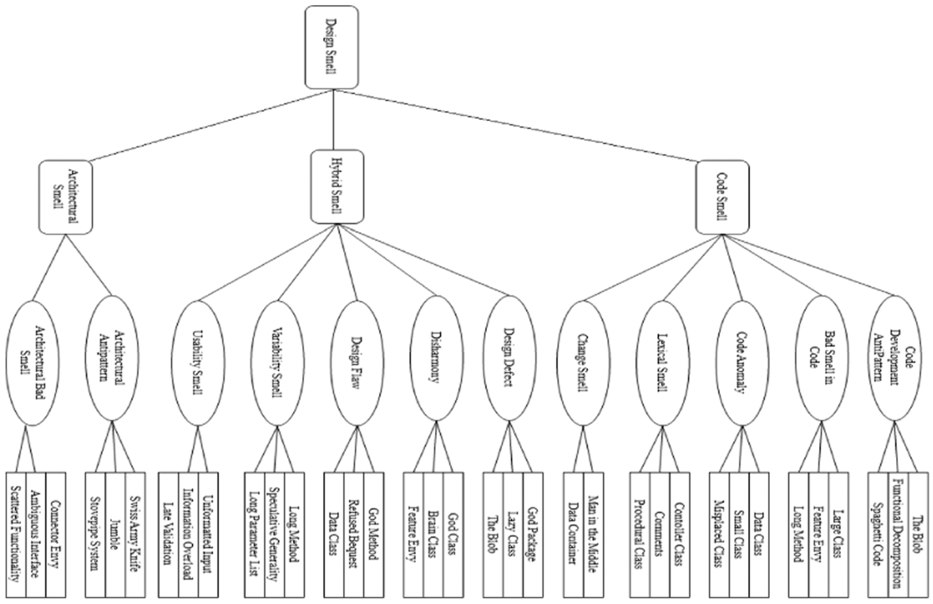


Fig. 10 Classification of Design Smells based on the author's terminology and smells

5.1.2 Design Smell scopes in code

Different sets of Design Smells are detected with different scopes at the code level. Smell scopes ranged from a small operation (instruction) to large systems. The conceptual model part, describing the enumerated types, shows the scopes we considered (system, subsystem, package, class, method, operation). Figure 11 presents the distribution of the different Design Smells mentioned in the selected papers. As can be seen, most Design Smells focused on the class (246) and method (183) levels respectively, while a few smells are focused on the package (4) and subsystem (5) levels.

5.1.3 Joint Design Smell catalog

In the selected papers, we found 662 different Design Smells present in the different catalogs and included by authors in the different smell concepts which are set out in Fig. 10. Putting them all together in a joint Design Smell catalog, they can be analyzed as a whole. We are particularly interested in which smells attract the attention of the research community and which smells are the most detected in software.

Figure 12 presents the most cited Design Smells in the selected papers. We select those smells cited in more than 25 papers to show in the Figure. The group formed by Feature Envy, God Class, Long Method, Data Class, Shotgun Surgery, The Blob, Long Parameter List, Refused Bequest, Large Class, Duplicate Code, Message Chain, and Data Clumps make up 80% of the most cited smells (more than 40 papers). It can be said that they have attracted the attention of the research community more than other smells.

We have also noticed, from the definition of the different Design Smells, that different authors have used different smell names to describe the same problem in design/code. In our experience, the God Class Disharmony, The Blob Antipattern, and Large Class Code Smell are describing similar problems that can essentially be considered as the same. From this set, the most cited Design Smell in the selected papers is God Class. However, we found several authors of studies who have taken these sets (The Blob, God Class, Large Class) independently, such as [S136, S237, S301, S318, S380]. Also, the authors of [S135, S298, S299] cited that The Blob was also called a God Class, but was different from the Large Class. Therefore, we decided to present the relevant information of these sets separately in this work.

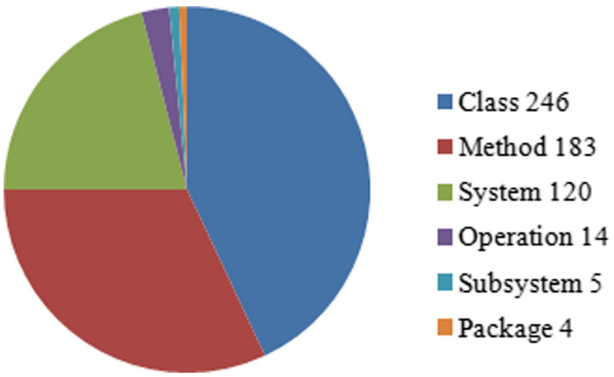


Fig. 11 Distribution of Design Smells over scopes in code

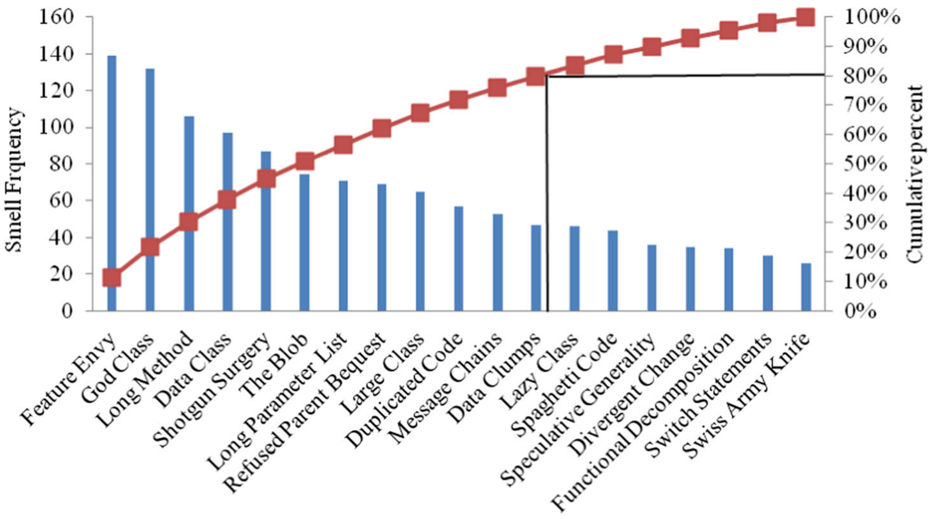


Fig. 12 Distribution of most cited Design Smells over the selected papers

Figure 13 presents the distribution of the most detected Design Smells over the total number of projects used in validation (case studies, experiments, etc.) in the relevant papers. The figure only shows those smells detected in more than 40 projects. The Long Method and Feature Envy are detected in the highest number of projects, and God Class, Long Parameter List, and The Blob come next. As can be seen, Long Method, Feature Envy, and God Class have been detected in 65% of the projects used in validation.

Figures 14 and 15 show, respectively, the class and method level Design Smells that have the highest frequencies in detection. This means the smells were detected in a total amount of classes/methods. All smells in the figures appeared more than 600 times for the method level and more than 1500 times for the class level in all the projects used in validation in the papers included in this study. Regarding the class level smells, Lazy Class was detected in 100 projects, and 80,597 classes were reported as having this smell. On the other hand, in the method level smells, Long Method, which was detected in more than 290 projects, 133,001 methods were reported as having the smell. Detecting Design Smells in a high number of projects does not mean that it has the highest frequency. For example, The Blob Antipattern, which was detected in more than 143 projects, appears in 16,994 classes, while The Blob Operation, which was detected in less than 20 projects, appears 6359 times because there are many more methods than classes.

In the data extraction process, we found a set of papers, such as [S7, S29, S48, S54, S74, S113, S167, S225, S234, S237, S264, S327], which are interested in studying and identifying the relationship between different types of Design Smells and if there exists a possible coupling between some types. In [S132], the authors find relations between the following pairs of smells (The Blob, Data Class) and (The Blob, Large Class). Also in [S236], they found a relation between (God Class, God Method), (God Class, Feature Envy), (God Class, Data Class), (God Class, Duplicate Code), (Data Class, Data Clumps), and (Interface Segregation Principle Violation, Shotgun Surgery). In [S264, S327], the authors called the group of related Design Smells that were detected together in the same software artifact (class, method,…) “agglomeration,” as in Divergent Change and Shotgun Surgery [S264], and the group of Divergent Change, Feature Envy, Long Method and Shotgun Surgery [S327].

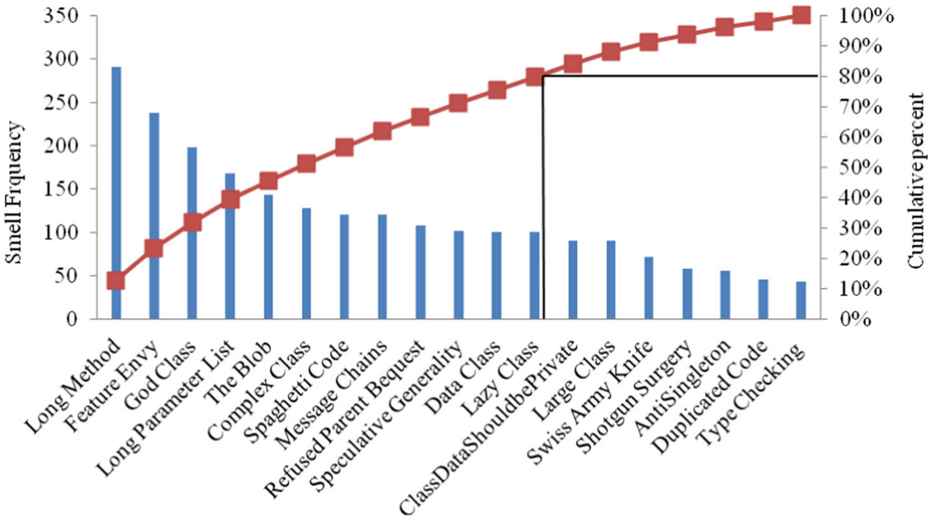


Fig. 13 Distribution of the most detected Design Smells over the projects used in validation (case studies, experiments,…)

5.2 RQ2

What approaches have been proposed to detect Design Smells in software?

In order to answer the research question, we analyzed it from different aspects related to the types of approaches, the degree of automation, activities, and types of artifact dealt with. This information was extracted according to the conceptual model using attribute "type" of concept "Approach" and the association called "characterization" with each "Article" (having the "year" of publication as attribute), the attributes "automation," "mainActivity," "otherActivities," and "typeOfArtifact" of the concept "Approach."

5.2.1 Types of approaches

Several approaches have been proposed in the literature based on different techniques and strategies for Design Smell Detection. In the conceptual model, described in Figs. 3 and 4,

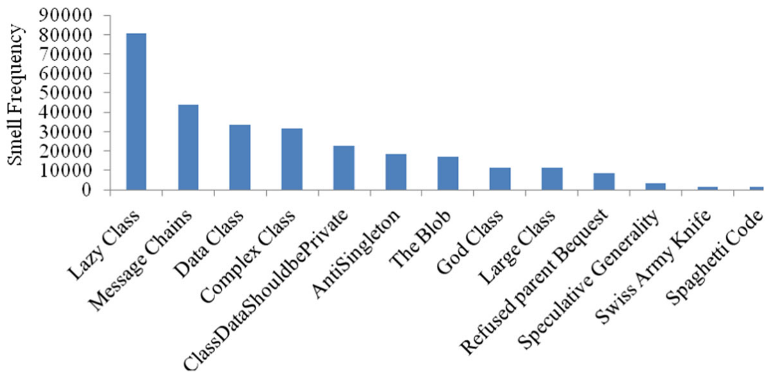


Fig. 14 Frequency of most detected class level Design Smells over the selected projects

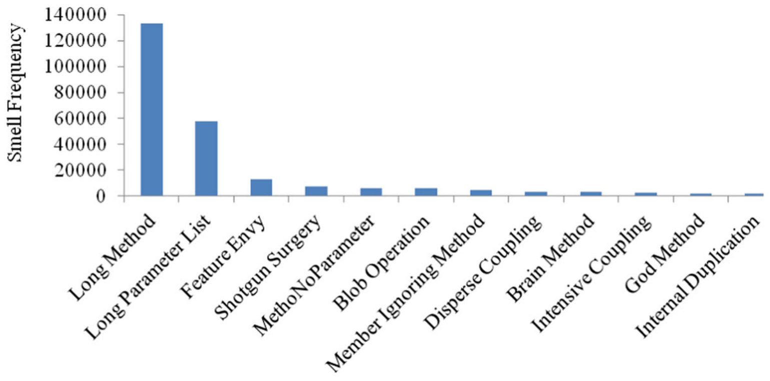


Fig. 15 Frequency of most detected method level Design Smells over the selected projects

each type of approach we detected during the processes of data extraction, classification, and iteration on conceptual modeling is represented in the enumerated type called "TypeOfApproach."

Figure 16 shows the distribution of the approaches over the publication years. The most frequent approaches found are metric-based, logical/rule-based, machine learning based, Context/Feedback-based, graph-based, and visualization-based. In the years 2000, 2001, and 2002, Design Smell Detection (if done) was carried out by manual inspection based on some metrics, rules, and visualization. From 2003 to 2009, the number of different proposed approaches increased. In 2007, we found 4 papers applying a combination of different approaches. Since 2010 to the end date of the study (December 2017), the different types of approach rose. As can be seen, there is a big leap from the year 2010, with peaks in the last 3 years of study 2015 to 2017.

More than 60% of the studies are covered by looking at the application of metric-based, logical/rule-based and machine learning-based approaches, where 34% (134 studies) of the

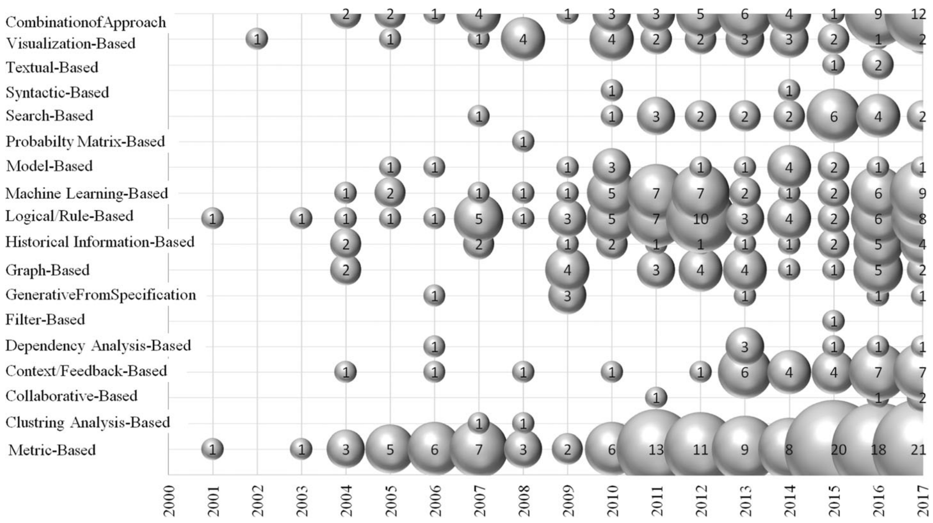


Fig. 16 Distribution approaches over the publication years

selected papers proposed a metric-based approach to detect different types of Design Smells. The following studies [S8, S31, S61, S149, S199, S207, S208, S276, S314, S330, S362, S377, S379] used different sets of defined or proposed quality metrics, such as size, cohesion, inheritance, dependencies, and coupling metrics for smell detection. The relationship between metrics and Design Smells is based on establishing relative threshold values using different techniques and strategies. A substantial effort is needed to identify the right threshold value for each metric. Several studies investigate different methods to obtain these thresholds, such as [S26, S36, S55, S86, S111].

Regarding the logical/rule-based approach, 15% (59 studies) of the selected papers focused on rules for smell detection [S27, S32, S95, S106, S117, S169, S204, S263, S341, S358]. These rules are either specified by using a combination of metrics and threshold values or other different types of facts directly related to the definition of the Design Smells. Each rule is specific to particular Design Smells and can be defined manually or automatically using different techniques.

Hence, the types of approaches applied in the papers are not mutually exclusive. Combinations of some of these types of approaches can lead to better solutions. Between 2010 and 2012, the research focus moved to machine learning approaches in order to improve smell detection. The authors of 11% (45 studies) of the selected papers applied machine learning. Another interesting point can be found in 2015, when the application of search-based approaches gained interest. However, automatic learning and search-based approaches have not been widely exploited in smell detection, when compared with metric and logical/rule-based approaches. Several learning techniques derived from specific classifiers were proposed in the literature [S9, S57, S79, S99, S119, S120, S151, S290, S294, S312, S320, S334, S365, S382] to solve classification problems.

In 8% (33 studies) of the papers, the authors focused their attention on the role of human experts in different Design Smell activities, from detection to prioritization to refactoring [S84, S158, S313, S364, S380]. The Context/Feedback-based approach considered important factors and should be taken into account when suggesting different techniques.

The authors of 6% (23 studies) of the selected papers [S11, S70, S98, S154, S335, S366] used the search-based approach to detect different types of Design Smells. Several techniques and algorithms were proposed for extracting the specified rules to detect smells with techniques based on genetic and heuristic search algorithms.

Regarding the Historical information-based, 6% (22 studies) of the selected papers adopted the historical change in source code as another source of information to identify different types of Design Smells [S155, S160, S260, S264, S307, S353]. The evolution of the software artifact over a long period assists software maintainers in prioritizing the most critical parts of the code.

The application of graph-based and visualization-based approaches are very similar. Diagrams and graphs are used to support the detection process. Graph-based approaches were applied in 7% (26 studies) of the selected papers, for instance [S147, S211, S277, S278, S347]; also 7% (26 studies) were applications of the visualization-based approach, for instance [S55, S140, S322, S326, S346, S378, S386, S391]. In some cases, when the software is very complex, the graphical representation of the software artifact arises as a solution to deal with complexity. These approaches have been integrated with some detection tools, as presented in [S47, S48].

Other approaches that are proposed for Design Smell Detection are applied in fewer than 16 papers each, such as Model-based (16 studies) in [S37, S221, S287], Dependency analysis-based (7 studies) in [S108, S167, S286, S293], Generative from Specification (7 studies) in [S51, S102, S298, S343], and Collaborative-based (4 studies) [S195, S332, S333]. There are also examples of Syntactic-based [S181, S228], Textual-based [S157, S163], and Clustering Analysis-based [S49, S125], with 2 studies for each of them, while the Probability matrix-based [S174] and Filter-based [S53] approaches are singular cases applied in just one study.

When these types of approach are combined to improve smell detection, it is interesting to analyze the relationships between the different types in the papers in the study. We applied formal concept analysis (FCA) (Ganter and Wille 1999) to study these relationships. As a result of FCA, we obtained a Galois lattice representing the relations between types of approaches and the papers that applied them. The construction of a Galois lattice guarantees that each attribute is introduced in a single point (node). Therefore, each node in the lattice represents the objects (papers) with the same attributes (the type of approaches applied in the paper). When a paper combined more than one type of approach, it has them as attributes. Hence, a hierarchical partial order is established. As can be seen in Fig. 17, nodes in the lower levels of the lattice, which are connected to higher level nodes, "inherit" the attributes and introduce new attributes. In this way, the nodes in the lower position represent the set of papers that combine approaches, while the nodes in the higher positions represent the set of papers that applied one type of approach. Descending from top to bottom of the lattice, following the connections, we can find the different combinations applied. For example, the authors of studies [S84, S158] applied the Context/Feedback-based approach only, while the author of [S223] combined two types of approaches: Context/Feedback-based and Historical information-based. Also, the authors of the studies [S6, S17, S52, S73, S89, S101, S103, S105, S107, S124, S126, S130, S133, S135, S137, S138, S139, S177, S187, S222, S272, S285, S306, S316] combined the metric-based approach with other types, such as logical/rule-based, visualization-based, search-based, Historical information-based, graph-based, and/or machine learning.

5.2.2 Degree of automation

The proposed approaches varied in the degree of automation from manual inspection to fully automated. Manual detection is time-consuming and error-prone because it is applied by software developers/maintainers without any assistance from other tools to find Design Smells. So, this type is not efficient with large size projects. The computer-aided degree is focused on using the computer to obtain some numerical measurements, such as metrics, statistical values,

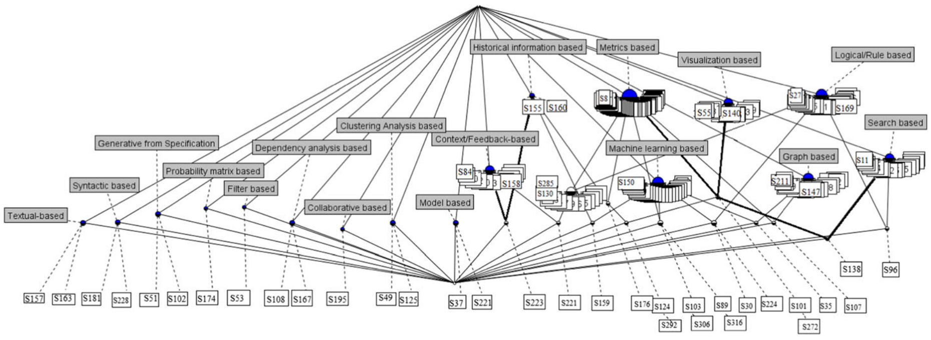


Fig. 17 Galois lattice for combinations of approaches

or a graphical representation of the software in order to assist in the detection task. Detection is done by the developer/maintainer but with some aid. This is placed between manual guidelines and semi-automated approaches. This kind of processes is less repeatable and efficient than the semi-automatic and automatic. The semi-automatic approaches perform the detection tasks, but they also require human intervention. Some feedback or interaction with experts is needed. Finally, the fully automatic approach is when the highest degree of automation is reached, where no human intervention is required.

Figure 18 shows the distribution of different degrees of automation over the papers in the study, classified according to what the authors said in their papers. As can be seen, 63% of the selected papers focused on fully automated detection, while 19% are considered semi-automated and a few approaches used the manual guidelines (12%) and computer aided (4%).

During the data extraction, we found a contradiction between many authors on classifying the work done as semi-automated or fully automated. For example, the proposal in [S132] is considered by the author of [S158, S228] as fully automated, while the authors of [S223] consider it as semi-automated. We consider the proposal in [S132] to be fully automated. Another example can be found in [S126, S139], which is considered in [S100] to be in the degree of manual detection, while in [S35] the authors consider them as semi-automated. We considered both to be semi-automated. Also in [S100], the authors considered the proposal in [S174] as semi-automated, while in [S158] the authors considered it as fully automated. We consider the work in [S174] is semi-automated. There is a conflict in classifying the proposed approaches based on the degree of automation in some papers, and the reason upon which this decision is based for distinguishing between automation concepts.

5.2.3 Activities

The work done in the different papers included in the study focused on one or more activities related to Design Smells. As a result of the effort made in the conceptual modeling, the activities developed in the selected papers are divided into six main types that involved Specification, Detection, Correction, Visualization, Prioritization, and Impact Analysis. As can be seen in Fig. 19, most of the proposed approaches focused on smell detection more than any other activity.

The specification activity is related to specifying new types of Design Smells, improving the definition of existing smells, or producing a new taxonomy. These papers used domain-specific languages that allow the specification of several Design Smells and can certainly be found in [S102, S135, S137, S144, S171, S228, S298, S306, S367].

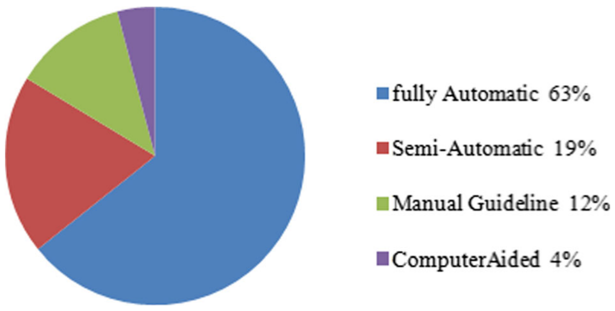


Fig. 18 Degree of automation over the selected papers

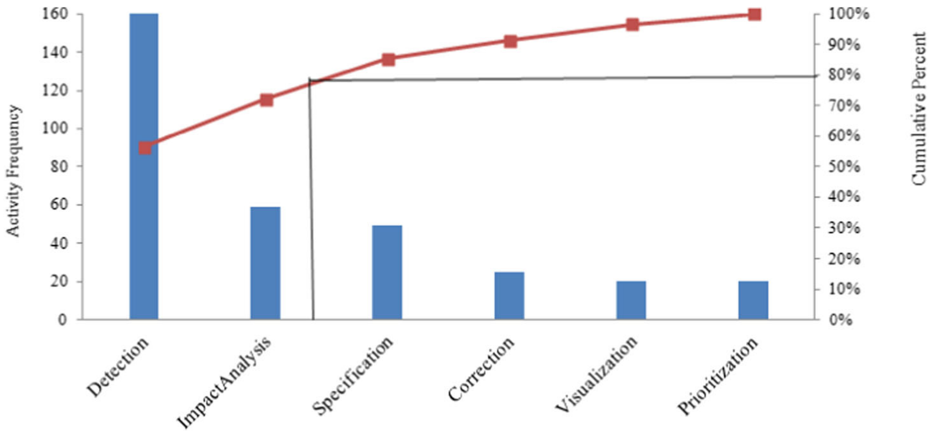


Fig. 19 Distribution of the Design Smell related activity over the selected papers

The majority of the works in the study focused on the detection activity, which is the main activity that we focused on in this work.

The correction activity is related to the proposed method for removing the Design Smells and improving the software design. This activity modifies the software artifact that includes the detected Design Smells, without changing the behavior. This is what refactoring does. This activity is implemented with various tools and supported in a semi-automated or fully automated manner. The following studies focused on this activity [S18, S191, S72, S118, S127, S130, S152, S153, S155, S173, S244, S255, S320, S335, S362, S387].

The visualization activity, such as that developed in the studies [S13, S28, S47, S48, S125, S147, S211, S322, S346, S378, S386], presents a graphical representation of the specified software artifact, which helps in the detection or graphical representation of the presence of the smell itself.

The Impact Analysis activity focused on analyzing the impact of relations between different types of Design Smells, or the impact of changing specific Design Smells on different software quality factors, such as maintainability and understandability. The authors of [S52, S59, S89, S190, S236, S240, S245, S268, S307, S327, S369] focused their attention on this activity.

Finally, the prioritization activity related to the ability of a Design Smell Detection proposal to rank them based on their impact on software quality or other factors. This activity can be supported in a semi-automated or fully automated way. Some works have appeared in the last 5 years, such as [S17, S152, S186, S192, S222, S223, S239, S275, S282, S292, S352, S360], focusing on this activity in parallel with other smell activities, such as detection, correction, visualization, and smell impact analysis.

5.2.4 Types of artifacts

Design Smells can be detected in different types of software artifacts, such as Executable or binary code, source code, UML Diagrams (class, communication, activities), workflows or process diagram, test cases, and ontologies. As a result of the conceptual modeling after several iterations on data extraction, we describe an enumerated type TypeOfArtefacts previously shown in Fig. 4.

Design Smells are detected in different types of software artifacts, as shown in Fig. 20. The majority of the papers in the study focus on detecting Design Smells in the software source code, while a few detect them on UML diagrams, such as class or communication diagrams, binary code, ontology and test cases. The nature of Design Smells influences the type of artifact used in detection.

For example, some of the smells related to the software architecture can be detected using UML class and component diagrams, and so on.

5.3 RQ3

Which (prototype) tools have been used to detect Design Smells in software?

In this section, we describe the results obtained from the data collected as part of the concept "Tool" in the conceptual model, such as Availability and Licensing (attributes "isFree" and "isOpenSource"), Internal Software Representation (attribute "internalSoftwareRepresentation"), languages that can be processed (attribute "supportedLanguages"), and the Design Smells detected (obtained throughout the association "can be detected with" with the concept "Smell"). The attribute "kind" of the association class "PresenceDescription" stores information regarding the association called "mention" between each article in the study and the tools mentioned.

Several tools/prototypes have been developed in the literature for semi-automatic or fully automatic smell detection. We found that in general the proposed tools/prototypes detect Design Smells as binary decisions (having the smell or not).

Different techniques are applied in tools/strategies to identify the candidate smells. Figure 21 presents a cloud of tags produced using TagCrowd with the same settings as those used in Fig. 8. The text is extracted from the string field called "technique" that we included in the "Approach" concept when elaborating the conceptual model. This free text was reserved for annotating algorithm names and other details. The size of the words indicates their frequency and importance. Different types of mining algorithms, dynamic or static analysis, meta-modeling, genetic algorithms (genetic programming), visualization, etc., were also used.

During the data extraction, we found 148 different tools and prototypes for Design Smell Detection. The first Design Smell Detection tool to appear was reported in 2002 (jCOSMO [S220]). There has been a continuous rise in the appearance of new detection tools between 2004 and 2010, such as Analyst4j, Cultivate (from jTransformer suite), DÉCOR, iPlasma, inCode, inFusion, jDeodorant, an emerging PMD, Reek, RevJava, SA4J, and Together.

As stated before, between 2010 and 2015, research activity regarding Design Smell Detection has experienced rapid and huge growth. A new group of tools has emerged, including Stench Blossom, ConcernReCS, SourceMiner, BSDT, JCodeCanine, GrouMiner,

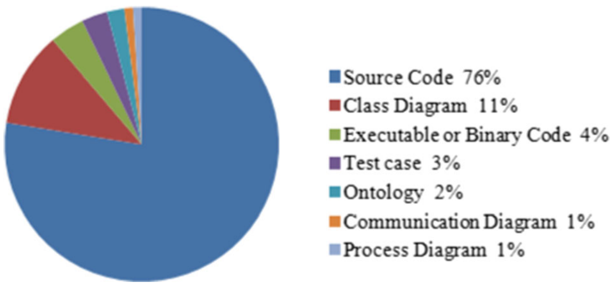


Fig. 20 Distribution of the software artifacts over the selected papers

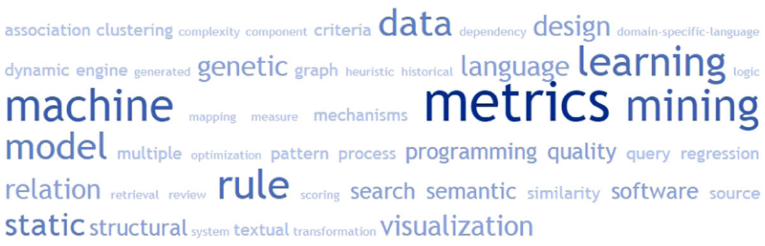

association clustering complexity component criteria data dependency design domain-specific-language
dynamic engine generated genetic graph heuristic historical language learning logic
machine mapping measure mechanisms metrics mining
model multiple optimization pattern process programming quality query regression
relation retrieval review rule scoring search semantic similarity software source
static structural system textual transformation visualization

Fig. 21 Cloud of tags produced from joining all the strings collected in attribute "techniques" from concept "Approach"

CodeVizard, JSNose, Hist-Inspect, SVMDetect, PTIDEJ suite (containing DÉCOR and its evolution DETEX), BLOP, and an evolution of the previously emergent PMD with a new set of rules for Design Smell Detection. There is also a set of research prototypes without any particular name that implements the techniques reported by their authors in their publications.

Figure 22 shows the most mentioned detection tools in the selected papers. The tools in the figure have more than 25 mentions in different papers. As can be seen, JDeodorant (Fokaefs and Tsantalis 2007) and DÉCOR (Moha et al. 2010) are the most mentioned, with significant differences concerning the other tools. JDeodorant appears in 65 studies and DÉCOR in 63 studies. The group of JDeodorant, DÉCOR, PMD, and InFusion covers more than 80% of the selected papers.

5.3.1 Reasons for the presence of a tool in a paper

There are different reasons for the presence of a tool in a paper. We classify the kinds of presence into five categories, as shown in Fig. 23. A tool is mentioned in a paper for comparison purposes 33% of the times, i.e., to compare results in detection with other different tools or prototypes as part of the evaluation processes of a particular tool, or in studies focused on comparing different tools.

The presence of tools in papers aimed at the introduction of a novel tool represent 25% of mentions, while improving the tools for detecting a new type of smell, obtaining better performance indicators, etc., represent 19%. The presence of tools in papers due to review studies was 12%.

We find that DÉCOR and JDeodorant were the tools with the greatest presence in comparisons, with DÉCOR appearing in 27 studies and JDeodorant in 20 studies. On the

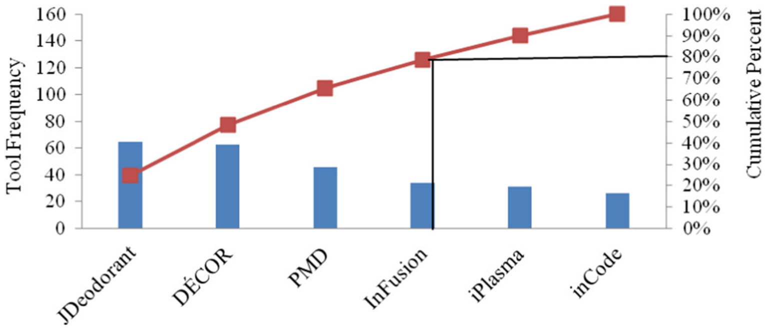


Fig. 22 Frequency of the most mentioned detection tools

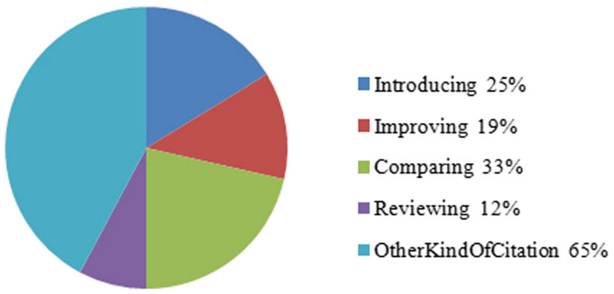


Fig. 23 Distribution of the kinds of presence of tools across the papers in the study

other hand, iPlasma and inCode were the most frequent in papers aimed at improvements (7 studies for each), while, regarding the presence in review studies, JDeodorant is the most mentioned.

5.3.2 Characteristics of the most cited tools

Table 6 summarizes some characteristics of the most mentioned detection tools (those shown in Fig. 22). The tools are characterized by whether they are free or not, whether they are open source or proprietary, their supported languages, the terms used to describe the Design Smells, the internal representation of the software artifact, the degree of automation, the ability to also perform refactoring, the way to run the tool (execution environment), their ability to generate metrics, the type of input source, the output format, the facility to work with command line (TUI) or Graphical User Interface (GUI), and the list of Design Smells the tool can detect.

As can be seen in the table, inFusion and inCode are commercial tools which include more features than open source tools. By looking at the origin of the pair of tools (inCode, inFusion), we find they were developed by the same institution, the intooitus company (https://www.intooitus.com/), which was a startup that originally arose from the group (LOOSE) at the Politecnica University of Timisoara in Romania. iPlasma was developed by the same LOOSE research group. Therefore, the iPlasma tool is the origin of inCode and inFusion, and their techniques have iPlasma as their base.

Each tool was developed to detect a particular set of Design Smells. Analyzing the most cited tools in detail, the majority adopted the metric-based approach to detect Design Smells, except JDeodorant, which uses clustering methods, and PMD, which uses a rule-based approach.

The inFusion tool has the ability to detect 22 Design Smells in different software artifacts. DÉCOR can detect both the higher-level smells (Architectural Antipatterns) and a set of lower-level smells (Code smell) as well.

The relation between detection tools and Design Smells will be explained in more detail with the FCA graph later in this section.

5.3.3 Programming languages that can be processed

Figure 24 shows the relation between the most cited detection tools and the supported programming languages. As can be seen from the Galois lattice figure, the objects are detection tools, and the attributes are the programming languages. The context is defined by

Table 6 Characteristics of the most cited detection tools

Tool	inFusion	inCode	iPlasma	JDeodorant	PMD	DÉCOR
Free	No	No	Yes	Yes	Yes	Yes
Type	Proprietary	Proprietary	Open source	Open source	Open source	Open source
Supported languages	Java, C++, C	Java, C++, C	Java, C++	Java	Java, C, C++, C#, PHP, Ruby, etc.	Java
Smell term	Design Flaw	Design Flaw	Disharmony	Bad Smell	Design Flaw	Code smell, Design Smell
Approach	Metric-based	Metric-based	Metric-based	Clustering analysis	Rule-based	Metric-based
Automation	Automatic	Automatic	Automatic	Automatic	Automatic	Automatic
Internal representation	AST	AST	Object model	AST	AST	Object model
Refactoring	No	No	No	Yes	No	No
Environment	Standalone	Eclipse plug-in, standalone	Standalone	Eclipse plug-in	Eclipse plug-in, standalone	Standalone
Generates metrics	Yes	Yes	Yes	No	No	No
Input data	Source code	Source code	Source code	Source code	Source code	Source code
Output data	Textual, visual	Textual, visual	Textual, visual	Textual	Textual	Textual
Command ine (_TUI)	No	No	No	No	Yes	No
Design Smells	BM, CD, DC, FE, GC, IC, MTM, RB, SD, SS, SB, ID, ED, B, BO, DaC, MsC, DH, SC, TB, UD	DC, DaC, DupC, FE, GC, GM, MC, RB, SS, MsC, SC	BC, BM, DC, DisC, FE, GC, IC, SS, RB, TB, LM, LPL, SG	FE, GC, LM, TC, DupC	LC, LM, LPL	LC, LzC, LM, LPL, RB, SG, MsC, SS, DupC, C, DC, NP, GV, CC, PC, LCO, DivC, B, SpC, FD, SAK

Design Smells acronym used in table: *B* Blob, *BC* Brain Class, *BM* Brain Method, *BO* Blob Operation, *C* Comments, *CC* Controller Class, *CD* Cyclic Dependencies, *DaC* Data Clump, *DC* Data Class, *DH* Distorted Hierarchy, *DisC* Dispersed Coupling, *DivC* Divergent Change, *DupC* Duplicate code, *ED* External Duplication, *FD* Functional Decomposition, *FE* Feature Envy, *GC* God Class, *GM* God Method, *GV* Global Variable, *IC* Intensive Coupling, *ID* Internal Duplication, *LC* Large Class, *LCO* Low Cohesion, *LM* Long Method, *LPL* Long Parameter List, *LzC* Lazy Class, *MC* Misplaced class, *MsC* Message chain, *MTM* Missing Template Method, *NP* No Polymorphism, *PC* Procedural Class, *RB* Refused Bequest, *SAK* Swiss Army Knife, *SB* SAPBreakers, *SC* Schizophrenic class, *SD* Sibling Duplication, *SG* Speculative Generality, *SpC* Spaghetti Code, *SS* Shotgun Surgery, *TB* Tradition Breaker, *TC* Type Checking, *UD* Unstable Dependencies indicating that an object (tool) has an attribute (language). The objects (tools) in the low level inherit all the attributes (languages they can process) of the objects connected with them in the higher level. At the top of the lattice, the objects (tools) we find are JDeodorant and DÉCOR, which have a single attribute Java. The structure of the lattice shows Java is supported by all these tools. The object PMD, which is in the lowest level, supports all programming languages (Java, C, C++) introduced in the higher levels of objects and the languages that only PMD can process (C#, Groovy, PHP, Ruby, Fortran, JavaScript, PLSQL). Eighty-one percent (120 tools) of the detection tools cited in the selected papers (including all the tools shown in the figure) supported detection in Java source code.

C++/C are the second most supported, while C# is the third. Also, other languages not mentioned in the figure are supported by some tools, such as Paython by Pysmell and PMD-CPD, Smalltalk by JSpIRIT and SpIRIT, COBOL and Visual Basic by CCFinder, ObjectPascal/Delphi by Sissy,[5] and UML supported by Together and the PCM Bench tool. However, we do not find any detection tool that can handle a source code with multiple programming languages in the same project, except Understand,[6] which is a commercial tool.

5.3.4 Internal software representation

In order to achieve detection and determine the candidate Design Smells in software artifacts, each tool/prototype was designed to internally represent that artifact in a different manner. In the selected papers, we find eight different types of representation, which the tools adapted to internally represent software artifacts.

Figure 25 presents the frequencies of the most used internal representations for all the cited tools in the selected papers (148 tools). As can be seen, the majority of the cited tools adopted Graph and Abstract Syntax Tree (AST) representations, as compared with the remaining representations. The group of inFusion, inCode, JDeodorant, and PMD used AST, while the group of SourceMiner, Stench Blossom, Understand, and JCOSMO adopted graph representation. On the other hand, the group of iPlasma, DÉCOR, Together, and JSpirit used complex object models. Some tools used more than one representation, such as Sissy, which used graph and AST, and FxCop[7] that used object model and graph.

All the tools in the group of the most cited support a fully automated approach for Design Smell Detection, the input source is source code, while the output is text in different formats, such as CSV, txt, or XML, with the exception of the group inFusion, inCode, and iPlasma, which bring with them a visualization environment, as visually representing the source code might be easier for identifying Design Smells. This group has the ability to generate a wide set of code metrics, including size, cohesion, coupling metrics, and others. For example, inFusion gives 60 different metrics, inCode 50 metrics, and iPlasma 80 metrics, and they do not allow the users to modify the metrics threshold value.

Also, most of these tools focused only on smell detection, except JDeodorant, which includes refactoring operations that can be performed after the Design Smells have been identified. Besides, some of these tools can be integrated as an Eclipse plug-in or are available as a standalone tool. Nearly all of these tools should be used through a GUI, except PMD, which is the only tool that was developed to support both a Textual User Interface (TUI) and a

[5] http://http://sissy.fzi.de/sissy/
[6] https://scitools.com/
[7] https://msdn.microsoft.com/en-us/library/bb429476

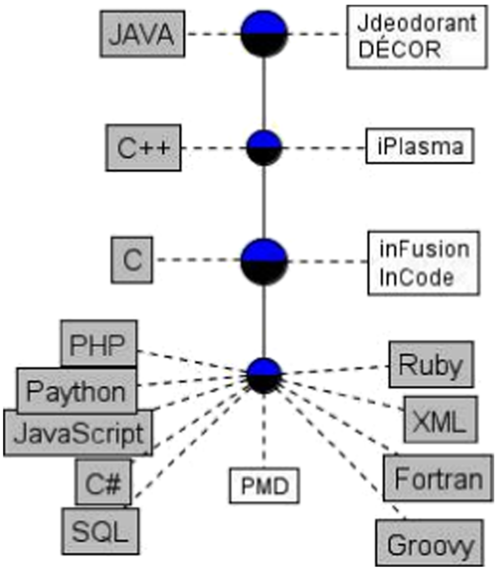


Fig. 24 Relation between most cited detection tools and supported programming languages

GUI. Having a TUI available is an advantage when looking to automate the smell detection process, for example in different tool integrations, or as part of the continuous integration and continuous delivery processes of software projects.

5.3.5 Design Smells detected: analyzing the most cited tools

Figure 26 shows the relation between the most cited tools and the set of Design Smells that they can detect. The relation is introduced by constructing the Galois lattice from a formal context (FCA), where the objects are detection tools, and the attributes are Design Smells. The context is defined by indicating that an object (tool) has an attribute (Design Smells the tool can detect). The set of attributes (Design Smells detected) at the lower level nodes (tools), which are connected with higher-level nodes, indicates the "inheritance" of attributes, which is to say the tool can detect the Design Smell directly, tagged at the node, and all the "inherited" attributes.

The lowest-level nodes (tools) show Design Smells that can be detected only by those tools. For example, the attribute (TC: Type Checking) at the lowest-level node representing the object (JDeodorant tool) indicates that only JDeodorant can detect TC. DÉCOR shows a list of Design Smells that can only be detected with this tool.

On the other hand, in the highest-level nodes, the set of attributes (GC: God Class, FE: Feature Envy) are hierarchically connected with more than one tool in the lower levels, and this means it can be detected by all of them. For example, the node tagged with (GC, FE) attributes connected (directly or indirectly) with nodes representing objects (tools) such as JDeodorant, inCode, iPlasma, and inFusion. Also, the LM attribute is connected with JDeodorant, PMD, and DÉCOR. The attribute (B) is only connected with DÉCOR and inFusion. We can say that the main observation is that there were no common attributes (Design Smells) at the top of the lattice, which means there are no smells that can be detected by all tools.

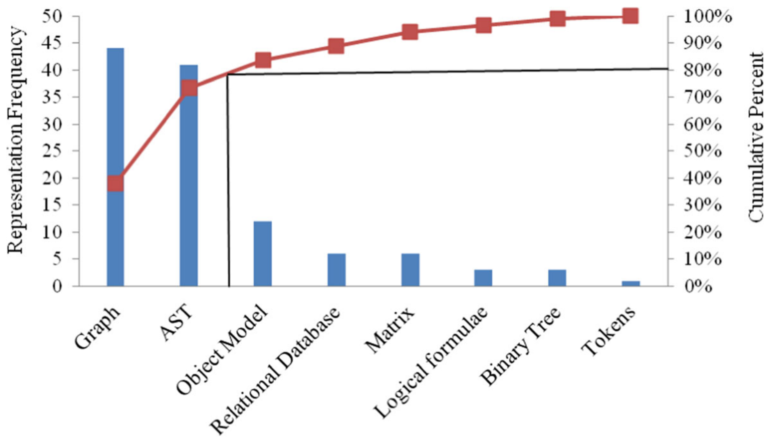


Fig. 25 Distribution of internal software representation over the detection tools

5.4 RQ4

Is the (prototype) tool validated by expert or benchmark or by comparison with other tools? Is the tool or strategy assessing results and measuring performance, precision, and recall?

In this section, we answer the fourth research question RQ4 regarding validation and performance indicators. The information used in this section is obtained from the attribute "type" of the concept "ValidationEvidence," the "uses" association between concepts "Project" and "ValidationEvidence," and the association of this last concept with "Measure" (having the attributes "type" and "reportedValue").

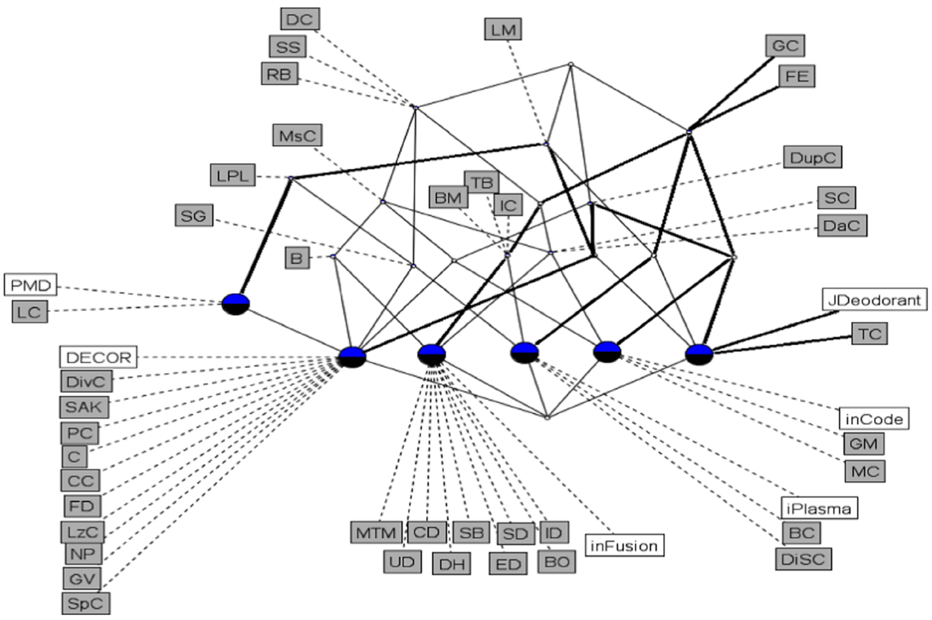


Fig. 26 Relation between most cited detection tools and set of Design Smells detected

5.4.1 Types of validations used in the studied papers

During data extraction, we have found a great deal of evidence that the tools/strategies in the study have adopted to validate their proposals. We model this with an enumerated type in the conceptual model, called TypeOfStudy, and whose values are a case study, experiment, systematic literature review, survey, and non-empirical studies.

Figure 27 shows the distribution of different types of validation evidence over the selected papers. As can be seen, most of the authors adopted experiments or case studies to validate their tools/strategies. The authors of 41% of the selected papers validated their work by conducting experiments, as in [S9, S48, S60, S96, S217, S332, S364], or controlled experiments (in the authors' words) as in [S123, S190, S188, S189,S289, S317, S338, S395]. Another group of studies (29%), as in [S71, S154, S163, S213, S223, S257, S273, S283, S330, S388], focused on analyzing one or more case studies to validate the proposed tool/strategy. In a few studies (6%), validation was performed using surveys. In this type of study, the experience of humans plays the main role, as in [S43, S154, S160, S175, S250, S268,S297, S313, S328, S355, S380]. In 2% of studies such as [S14, S110, S142, S154, S371, S390], the authors conducted a systematic literature review to identify the current state of the art on Design Smell-related topics, while the authors used post-mortem analysis for validation in less than 1% of studies [S270, S321, S357]. The remaining 16% of papers are considered theoretical.

5.4.2 Projects used in validation processes

A large number of projects from different domains, sizes, status, and versions had been used to validate the proposed tool/strategy. The vast majority of the validated projects used are open source obtained from different public repositories. By looking at the project repository source mentioned by the authors, we have found that most of the projects were obtained from SourceForge (24%), GitHub (20%), and less than 1% from the PROMISE.[8] For the remaining projects used in validation, the authors did not mention the repository source.

In most of the papers, the authors give some metrics on the analyzed projects, such as the number of lines of codes (LoC), while a few mentioned some other metrics, such as number of packages, classes, and methods or other nominal information, such as the project domain, the project status (beta, production/stable, etc.), or the URL of the analyzed project.

Figure 28 shows the distribution of implementation languages over the projects used for validation in the selected papers. As can be seen, most of the projects are implemented in Java and a few in another language. Figure 29 shows the distribution of the most commonly used projects over the selected papers. All projects in the figure are used in 15 papers or more. As we can see, the group of projects that includes GanttProject, JFreeChart, Xerces, and ArgoUML was the most used, especially the versions GanttProject v1.10.2, JFreeChart v1.0.9, ArgoUML v0.19.8, and Xerces v2.7.0.

We have found that the validation processes conducted in the papers in the study may suit one of the following categories: a group of different projects, the same project with different versions, benchmarks, or a quality corpus. Most of the studies were performed using different groups of projects, for example [S14, S26, S45, S56, S106, S107, S116,

[8] http://openscience.us/repo/

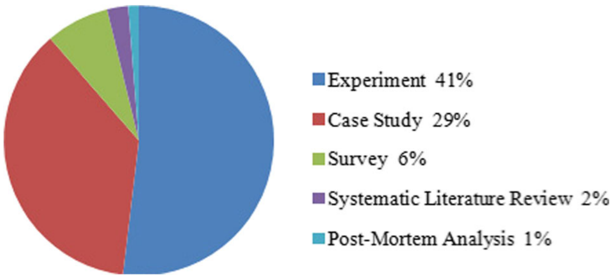


Fig. 27 Distribution of validation evidence over the selected paper

S118, S122, S147, S154, S163, S176, S278, S302, S316, S320], while the authors of [S71, S113, S186, S218, S225, S234, S237, S260, S387, S392] conducted their studies using several versions of the same project. A limited number of studies, such as [S53, S54, S55, S57, S60, S121, S272, S274, S275], validated the proposed work using a quality corpus (corpus including a large number of projects ranging from 74 to 500 projects). Almost all of them used the same corpus, except [S121]. The main observation was a lack of benchmark definitions and, hence, validation based on benchmarking. From all the selected papers, only the following studies are validated on benchmarks: [S21, S55, S64, S98, S101, S115, S144, S155, S159, S160, S171, S186, S227], while the same benchmark was used in [S98, S187].

5.4.3 Performance indicators and quality measurements reported in validation processes

Several types of indicator or quality measurement are used to evaluate the performance of the proposed tool or strategy. Figure 30 presents the distribution of the most used quality measurements or performance indicators over the selected papers. From the figure, it can be seen that most often Precision, Recall, False Positive (FP), Accuracy, F-Measure, and False Negative (FN), are given as performance indicators. Particularly, Precision and Recall are by far the most commonly used in comparison with the others. The value of precision and recall metrics ranged from zero to one. The highest obtained value of precision is (1) in the studies [S69, S91, S107, S144, S209, S228, S277, S323, S326, S348], and the lowest value is (0.25), as reported in [S45]. On the other hand, the following studies [S2, S80, S91, S99, S107, S134,

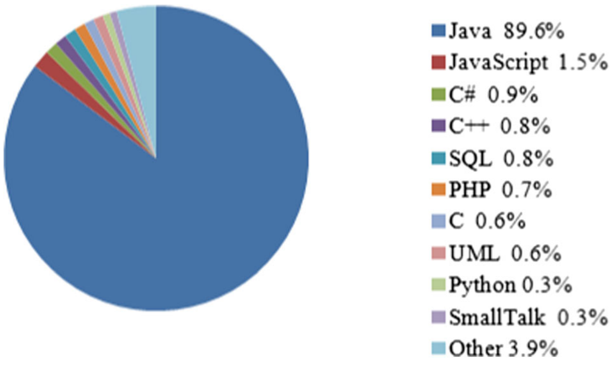


Fig. 28 Distribution of implementation languages over the number of projects

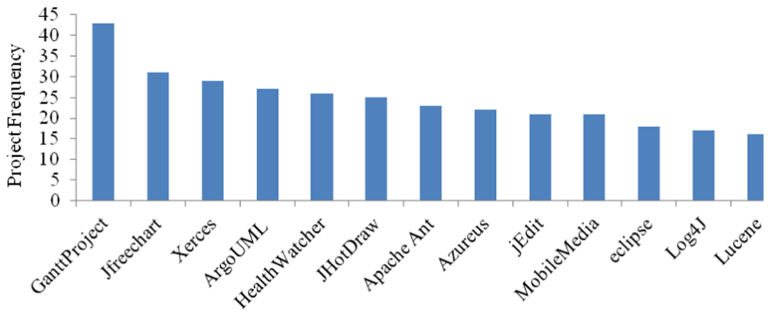


Fig. 29 Most used projects in validation of tools/strategies

S135, S144, S209, S228, S259] obtained the highest value of recall, which is (1), while the lowest value is (0.50), reported in [S157].

A few studies use parametric or non-parametric statistical tests, such as *T* test, Wilcoxon test, ANOVA test, Mann-Whitney test, Spearman's rank test, Fisher exact test, and principle component analysis (PCA). Other studies aimed to identify the degree of agreement between different raters, in this type of studies the authors used the Kappa [S287, S288, S290, S308, S337, S338] and Finn tests.

5.5 RQ5

Is Design Smell Detection related to the quality attributes of a quality model?

As a result of the data extraction, we found 662 Design Smells mentioned or detected in the selected papers. A limited number of the selected papers speak about the relationship between some software quality factors and Design Smells. In the conceptual model, we state that a Design Smell can negatively affect a quality factor, have no effect on other quality factors, and positively affect others (for example, the presence of a Design Smell can negatively affect reusability but positively affect efficiency). According to the conceptual model, the information used in this section was obtained from the concepts "QualityFactor" and "Smell" related

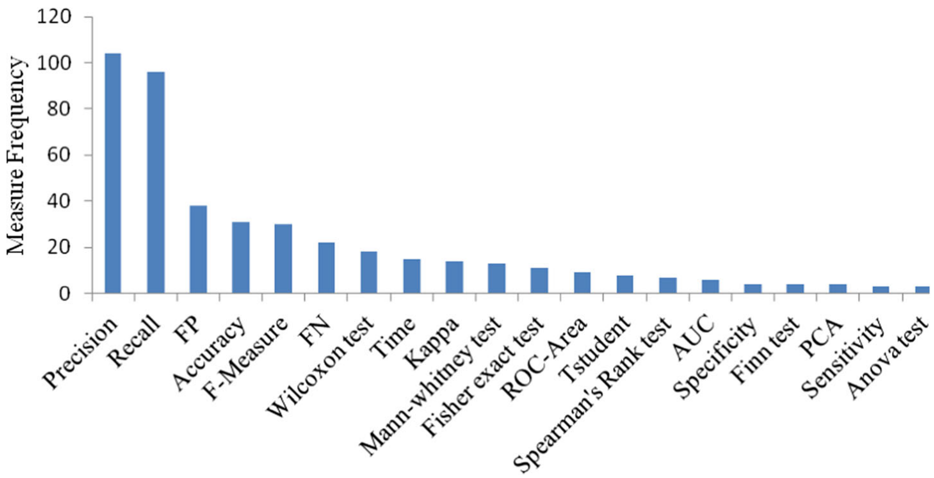


Fig. 30 Distribution of performance indicators or quality measurements used as evidence over the selected paper

by the “affects” association, having an association class called “Impact” which stores information regarding the “type” of impact (positive, negative, no impact, or unknown impact).

5.5.1 Mentioned quality models and quality factors

We found 22 software quality factors in the papers analyzed as being affected by different Design Smells. These factors can be classified into internal or external features that belong to popular quality models such as ISO/IEC 9126, Test Specification Model, FURPS Model, and McCalls Factor Model. Figure 31 shows the distribution of the quality factors mentioned in the relevant papers regarding Design Smells grouped into the quality models. The vast majority of software quality factors mentioned in the selected papers related to the ISO/IEC 9126 model.

The McCalls Factor Model[9] was proposed in 1977. It includes three main quality features related to software: revision, transition, and operations. Each feature includes a set of sub-features in a total of 11 internal sub-features for internal quality.

The FURPS Model[10] is the acronym of its five main quality factors: Functionality, Usability, Reliability, Performance, and Supportability. Each feature is related to several sub-features in a total of 28 sub-features.

In 1991, the ISO/IEC 9126 (Jung et al. 2004) model was issued. This model includes six main features classified as external software quality, such as Functionality, Reliability, Usability, Efficiency, Maintainability, and Portability. Each external feature involves a set of sub-features related to internal software quality. The total number of sub-features in the whole model is 27.

The author of [S143], in 2008, proposed a Test Specification Model and adopted the ISO/IEC 9126 model. In this model, seven main external features are proposed, which are: Test Effectivity, Reliability, Usability, Efficiency, Maintainability, Portability, and Reusability. Each main feature includes a set of sub-features in a total of 28 internal sub-features.

Figure 32 shows the quality factors most mentioned as being affected by the Design Smells in the relevant papers. As can be seen, the Maintainability factor was the most analyzed as affected by a wide number of Design Smells compared to the other factors. More than 70% of Design Smells were analyzed as influencing Maintainability, Performance, Understandability, Changeability, Reusability, Complexity, Efficiency, and Usability factors.

According to the previous quality models, Maintainability, Performance, Efficiency, Usability, Functionality, Reusability, and Reliability are considered external quality factors, and the remainder as internal quality factors. Also, regarding the ISO/IEC 9126 model, the group of internal factors, which include Changeability, Testability, Analyzability, and Stability, is related to the Maintainability external factor, while Understandability is related to the Usability factor.

5.5.2 Impact of design smells on quality factors: analyzing the most cited design smells

Approximately 52% (345 of 662) of the Design Smells present in the study were mentioned in some paper(s) as having a negative impact on quality factors, while less than 1% (4 of 662)

[9] http://www.sqa.net/softwarequalityattributes.html
[10] https://nl.wikipedia.org/wiki/FURPS

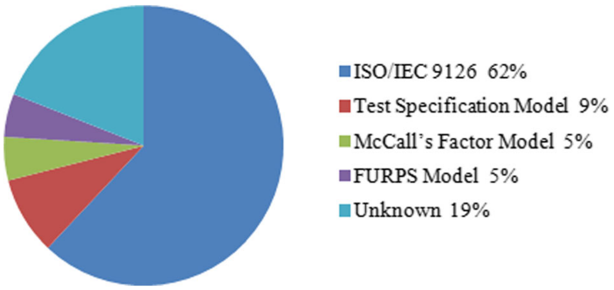


Fig. 31 Distribution of quality factors mentioned regarding the Design Smells in the relevant papers grouped by quality models

were mentioned in some paper recognizing its positive impact on some quality factor. The remaining 47% have no mention of their influence on quality factors. Table 7 presents the relationship between the most cited/detected Design Smells and the set of quality factors that have been analyzed as affected by these smells across the relevant papers. Most of these Design Smells are analyzed as affecting Maintainability, as can be read in [S153, S231, S232, S233, S236, S245, S357].

6 Discussion

This section discusses the results obtained from this systematic mapping study and the impacts of the said results for researchers interested in Design Smell Detection.

Regarding the study presented in Section 4, and observing the problems related to Design Smell Definition and Detection that still need the attention of the research community, the opportunity of launching a series of specific workshops or symposiums centered in this topic arises. This could be similar to the workshop on Refactoring Tools started in 2007, or the International Workshop on Refactoring, which appeared in 2016. Another possibility could be to define a track on Design Smell Detection as part of the recently launched series of

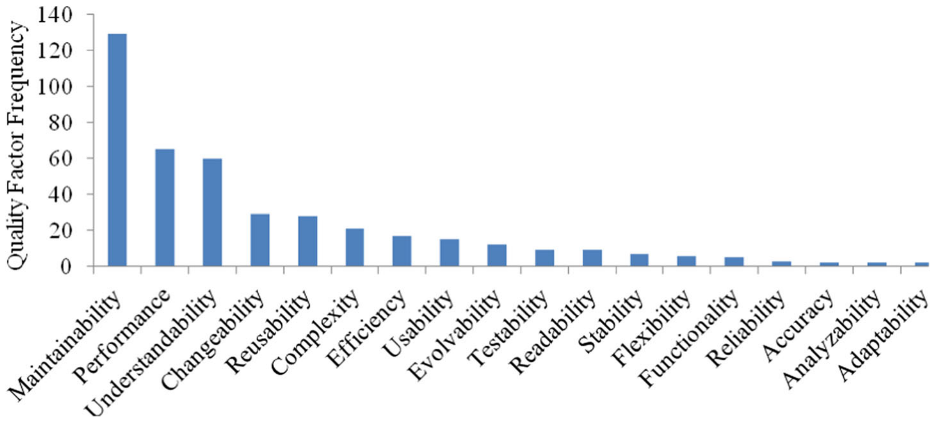


Fig. 32 Quality factors most mentioned as affected by the number of Design Smells that analyze their impact on them

Table 7 Relationship between Design Smells and software quality factors

Design Smell	Maintainability	Understandability	Usability	Readability	Changeability	Flexibility	Functionality
The Blob	X	X	–	–	–	X	–
Brain Method	X	X	–	–	–	–	–
Data Clumps	X	–	–	–	–	–	–
Data Class	X	X	–	–	–	X	X
Duplicate code	X	–	–	–	X	–	X
Functional Decomposition	X	X	–	–	–	X	–
Feature Envy	X	X	X	–	–	–	–
God Class	X	X	–	–	X	–	–
Large Class	X	X	–	X	–	–	–
Long Method	X	X	–	X	X	–	–
Long Parameter List	X	–	X	–	X	–	–
Lazy Class	X	–	–	–	–	–	X
Message Chain	X	–	–	–	–	–	–
Refused Bequest	X	–	–	–	–	–	–
Complex Class	X	–	–	–	–	–	–
Spaghetti Code	X	X	–	–	–	X	–
Shotgun Surgery	X	–	X	–	–	–	–
Temporary Field	–	X	–	–	–	–	–

Design Smell	Complexity	Reusability	Testability	Evolvability	Stability	Performance
The Blob	–	X	–	X	–	X
Brain Method	–	X	–	–	–	–
Data Clumps	–	–	–	–	–	–
Data Class	–	X	X	–	–	X
Duplicate code	–	–	–	–	–	–
Functional Decomposition	–	X	–	X	–	–
Feature Envy	X	–	X	–	–	–
God Class	X	X	–	X	X	X
Large Class	–	X	–	–	–	–
Long Method	X	X	X	–	–	–
Long Parameter List	–	–	–	–	–	–

Table 7 (continued)

Design Smell	Complexity	Reusability	Testability	Evolvability	Stability	Performance
Lazy Class	–	–	–	–	–	–
Message Chain	–	–	–	–	–	–
Refused Bequest	–	–	–	–	–	–
Complex Class	–	–	–	–	–	–
Spaghetti Code	–	X	–	–	–	–
Shotgun Surgery	X	–	–	–	–	–
Temporary Field	–	–	–	–	–	–

conferences called International Conference on Technical Debt, due to the correlation between the presence of Design Smells and Technical Debt.

Concerning authors and institutions, two important focuses of activity can be observed in the Université de Montréal in Canada (30 papers) and the Universitá degli Studi di Milano-Bicocca in Italy (21 papers), contributing to this topic with important publications and results continuously over time.

The importance of reaching a consensus in several aspects related with Design Smell Detection makes those studies that can be obtained through a collaboration between different institutions particularly interesting, even more so when the collaboration is across different countries and continents. Approximately 47% of the selected papers are authored with collaboration across institutions of different countries and only 14% are cross-continental collaborations. For these reasons, we consider this should be encouraged, specifically with the creation of groups of experts with standardization purposes.

6.1 RQ1

It was confirmed that different concepts are used in the selected papers to describe Design Smells. The variety of Design Smell concepts led the researchers to classify the detected Design Smells into different categories based on different criteria, such as structure, scope, level, variability, behavior, etc., as shown in Sections 5.1.1 and 5.1.2. Most of the Design Smells are included in more than one of these classifications.

To organize this knowledge, we show a joint classification that includes all the others, as well as the most used terms to describe different types of Design Smell, as seen in Fig. 10. In this paper, we have used the concept "Design Smell" as the main concept that unifies all other kinds of smells. It is worth noting that terms and classifications used by different authors are not sound. On the one hand, there are works where different classifications (different terms) are given for the same Design Smell (same Smell name) while, on the other hand, the same Design Smell (if we attend to its definition, i.e., what it means) is presented with different names.

This requires a standardization process to be accomplished that allows the terminology and its precise meaning to be unified. With this standard, cataloging all the Design Smells defined up to the present time should be possible, determining those that refer to the same smell with different names, and presenting (in the same way as the Design Patterns Catalog does) a unique entrance in the catalog enriched with "other names" or "also known as." Each catalog entrance should present a precise description of the Design Smell, but variants and idioms as well.

We found a total of 662 different Design Smells mentioned in the selected papers. Nevertheless, most of the selected studies focused on some groups of Design Smell more than on others, as shown in Figs. 12 and 13. Feature Envy, Long Method, and God Class (Blob, Large Class) are representative of this. These smells are frequently mentioned, are detected in many projects and by several tools.

There can be several reasons behind this: a better understanding of the smell definition; the identification of adverse impacts of these smells on different software quality factors such as maintainability, understandability, and changeability; the detection of groups of related Design Smells, i.e., the detection of other smells because they are tightly coupled somehow, such as God Class and Data Class, Feature Envy, and God Methods, whose presence is closely correlated with the presence of the other.

6.2 RQ2

Different types of Design Smell Detection approaches were proposed in the selected papers. In [S98], the authors classify the detection approaches into seven categories: manual, symptoms, metric, probabilistic, visualization, search and cooperative approaches. In this systematic mapping, we classify the state of the art approaches based on the technique, method or algorithm used to detect the Design Smells into Metrics-based, Logical/Rule-based, Search-based, Machine learning-based, Graph-based, Visualization-based, Model-based, Clustering Analysis-based, Collaborative-based, Dependency Analysis-based, Context/Feedback-based, Filter-based, Historical information-based, Probability matrix-based, Generative from specification-based, Syntactic-based, and Textual-based. This classification includes the above mentioned, except "manual" and "symptoms." We think the categories "manual" and "symptoms" are orthogonal to the type of approach because they refer to the degree of automation, manual and computer-aided, respectively.

It seems difficult to classify proposals into a single type of approach. Many authors use more than one diverse technique, method, or algorithm that belong to different approaches, i.e., a combination of different types of approaches to detect similar Design Smells.

Most authors focused their attention on six of these types of approaches: metric-based, logical/rule-based, machine learning-based, search-based, graph-based, and visualization-based, as Section 5.2.1 shows. It is hard to be categorical, but the higher number of papers based on a particular approach could be evidence of better results in detection. In order to measure the effects of a type of approach in detection, we would need the definition of benchmarks with which to obtain different performance indicators.

Design Smell Detection using a metric-based approach combined with rules of knowledge (rule-based, machine learning based) has gained in importance over the period of study. Most researchers tend to work with these approaches. This may be because new metrics, rules, and classifier algorithms can be defined to detect smells in more accurate ways. Therefore, these approaches are combined with each other more than with other approaches.

Regarding the activities related to Design Smell Detection, most of the effort is focused on detection activities. However, as stated above, it is important to increase efforts in the specification in order to help with the standardization of the concepts, which would also allow an increase in detection consistency. Moreover, improving the comprehension of the impact of Design Smells should support the applicability of these results. Impact Analysis should be guided by Quality models, as well as by Technical Debt models. Hence, both Specification and Impact Analysis activities should gain greater importance. Specification should be tackled in a standardized way, and empirical studies should be conducted in this line.

Prioritization deserves special attention. This kind of activity has been observed over the last 5 years. Software projects grow in size and complexity, and therefore, Design Smell Detection reports are huge. Accordingly, these reports should be configured in such a way as to first of all allow an analysis of those smells with a higher priority, i.e., those which, if not removed, could cause such a large technical debt increase as to lead to major problems.

In addition, advances in how to remove the Design Smells are needed to guide refactoring processes.

6.3 RQ3

Several tools/prototypes are proposed for Design Smell Detection in semi-automated or automated ways. Despite the increasing number of proposed tools, the adoption of these tools in the industry is still weak in comparison with refactoring tools.

In our study, we found a group of tools which are cited or used more than others, in particular, DÉCOR and JDeodorant. The reasons might be related to the variety of Design Smells they can detect, their usability, integration with other tools, or the input and output data as major points.

Some of the tools are designed for detecting and refactoring, such as JDeodorant [S49, S216]. Also, some of the detection tools used another metric tool to generate the required metrics for detecting Design Smells or to define and collect their own set of metrics.

We found some limitations related to the existing detection tools that widely affect their adoption for Design Smell Detection and these can be summarized by the following points:

- Most of the tools and prototypes focused on object-oriented language, especially Java.
- We did not find any tool that can analyze source code implemented in more than one programming language in the same system, except the Understand detection tool.
- Only a few tools can analyze very large sized projects (millions of lines of code).
- Most of the tools did not take into account the expert feedback or the influence of the context, characteristics of the organization, project domain, status, etc.
- In order to cover a wide range of Design Smells, several tools should be used.
- The common corpus of Design Smells detected by several tools is very narrow. This hinders tool comparison and, hence, tool choice.

In a production environment, increasing the number of tools that should be used, and therefore integrated into the process workflows and with other tools, increases the complexity and the entropy.

Analyzing the kind of mentions a tool receives in the relevant papers, the fact that "improving" a tool represents just 19% is rather contradictory when facing the huge set of tools/prototypes found and the 17 years of work in this area.

6.4 RQ4

Empirical studies, mainly based on case studies and experiments, are the preferred approaches for validating the proposed tools/prototypes. There is a lack of validation using experts in this field. QA experts from industry and higher education (academic) are needed to assess the results of the tools in terms of detecting false positives and false negatives. It is also important to have expert opinions regarding Smell Prioritization and Smell Impact on product quality and technical debt.

We have observed a set of projects that are frequently used in validating the proposals. Nevertheless, there is a lack of benchmark definitions for Design Smells validated by experts. In fact, only 5% of the works conduct validation based on benchmarks or corpus defined with this purpose. The PROMISES repository is defined as "a research dataset repository specializing in software engineering research datasets." In this repository, two datasets related to Bad Smells can be found. Incorporating Design Smell Detection benchmarks/corpus to this repository should contribute to advances in research on this topic.

Furthermore, the same set of performance indicators should be used to assess the results of the tools/prototypes. We have noticed that the most used indicators are Precision and Recall, False Positives and False Negatives. Usage of Kappa, ROC Area, Specificity, and Sensitivity indicators should be promoted. It is important to measure and compare the degree of agreement (Kappa based) among different tools and between tools and human experts. Moreover, we have observed that it seems easy to obtain Precision values near 1, which is why other indicators should be given to evaluating the goodness of the "diagnostic test."

6.5 RQ5

Different software quality models are found in the literature. Each model defines a set of main software quality features, and each feature includes a group of sub-features that, in total, affect the main feature. These sub-features are common to different models. A few studies identify the relationship between the detected types of Design Smell and the quality features. The nature of the relationship is identified from the point view of the authors and varies from one to another.

Most of the Design Smells, in particular, the group of Design Smells defined by (Brown et al. 1998) and (Fowler et al. 1999) affect more than one quality factor, as we showed before. Therefore, some quality factors will be influenced more than others. The most affected quality factors, such as maintainability, performance, and understandability, will have played a major role in the software maintenance cost. In this case, the set of Design Smells related to these quality factors will have the highest degree of priority for removal from the software.

6.6 Smells wisdom app

The conceptual model obtained was captured in relational database design. The data extraction and collection was materialized in populating the database. The compilation of knowledge was put in the hands of the research community interested in this topic by means of a web application. The reason is twofold: (1) the app should allow the community to contribute with their knowledge, as well as to criticize or comment on the information already collected in the app database; (2) the app should allow updating of the knowledge in a collaborative way for the coming years. For this reason, a social but supervised philosophy was chosen. Users should sign up for the app to give their opinions regarding information already collected and also to contribute with new information. Both opinions and contributions are supervised by the app admins in order to authorize insertions and updates to the app knowledge base with the aim of maintaining the soundness, consistency, and quality of the data.

The web application is available at https://smellswisdom.herokuapp.com/.

7 Threats to validity

In our mapping study, the obtained results might be affected by several factors such as the scope of the study, the coverage of the search, trends on study selection, and the accuracy of the data extraction process. In the following subsections, four types of threat to the validity, according to Wohlin et al. (2012), are discussed, including the mentioned factors.

7.1 Conclusion validity

Conclusion validity regarding a systematic mapping study refers to the extent to which the obtained conclusions can be considered as correct and to how rigorous and repeatable the process to obtain the conclusions was.

In this case, reliability is assured due to the selection of the six databases considered the most efficient and popular according to previous systematic mapping studies in related topics. In addition, the Google Scholar and snow-balling technique were included for completeness, despite leading to repetitions. Moreover, the considered period of 17 years includes all the work done up to the present time in the domain of knowledge regarding Design Smell Detection to enforce the validity of the conclusions. Our work also includes concept definitions and the organization of the terminology in order to avoid ambiguities. The process is repeatable and precisely defined, detailing the protocol we follow to conduct the systematic mapping, as can be read in Section 3 and summarized in Fig. 1.

The quality of the conclusions is enriched in a first stage with a definition of facets and, in a second stage, with the application of domain modeling techniques which include the definition of enumerated types that include and enrich previous facets. The articles examined can be classified using these facets and can be precisely described in terms of the concepts and relationships included in the conceptual model representing this knowledge domain.

The data extraction was accomplished by a PhD student with 3 years' experience in Design Smells and checked by a PhD supervisor with more than 15 years' experience in Design Smells. This process increased the quality of the data extracted. An expert in Empirical Software Engineering supervised and checked each part of the process.

Data integrity was guaranteed by means of conducting a process of data migration from the initial prototype in an Access database to a fully designed MySQL database with integrity restrictions available through a web application. It was a semi-automated process with more than three iterations, ending when a point of no errors in data migration and integrity checking was reached.

7.2 Construct validity

Construct validity is related to the process followed to select the information to be included in the study and related to the suitability of the research questions. In this systematic mapping study, the different types of Design Smell, approaches in detection, detection tools, and quality attributes related to Design Smell Detection in software are studied, defining and limiting the core concepts in the study. All these are reflected in the research questions. The domain model obtained is readable and represents knowledge that allows the research questions to be answered. Moreover, it allows future researchers to elaborate further research questions on the knowledge modeled in this way.

Our experience in the area is a guarantee that considers the concepts are interpreted in the correct way, and all the relevant papers are completely collected and analyzed.

We use both automatic and manual searches to ensure the completeness of the search string. The search string we use is comprehensive in the automatic search, but we use the manual search to discard the papers that are not related to the topic of this study.

7.3 Internal validity

The internal validity focuses on the analysis of the extracted information. The process we follow is detailed in Section 3. The reproducibility of the study is guaranteed due to the detailed specification of the search engines used, but also of the search string and the inclusion and exclusion criteria. Possible limitations of the search results were overcome by including different terms used by different authors for the same and similar concepts. All documents included are available in the full English text, excluding all those that are not relevant (not guaranteeing peer reviewing), etc.

In this study, the descriptive statistical analysis is used to cover all stages of the systematic planning protocol. Also, formal concept analysis (FCA) was used to extract whether there is some underlying structure and relationships in the collected information.

7.4 External validity

External validity is concerned with the representativeness of the selected papers regarding the main goals of the systematic mapping study. The findings of the study regarding the overall knowledge of the Design Smell Detection domain (approaches, tools, types of Design Smell, quality attributes, validation evidence indicators) were considered. The systematic protocol is helpful to obtain a comprehensive representation of the selected papers in the selected study period. Domain modeling techniques and iterations on the process were helpful in obtaining a good representation of the knowledge in the field. Therefore, the classification schema of the selected papers and the conclusions are valid for all studies related to Design Smell Detection and closely similar topics.

8 Conclusion

This systematic mapping corresponds to the interval of study from 2000 to 2017, 18 years of research from the year 2000 where the seminal chapter by Fowler and Beck was published as part of the well-known Refactoring book. We identified 395 articles published in seven well-known electronic databases in the domain of Design Smells Detection. The number of relevant studies in this domain increased through the determined interval of study. In the first 5 years (including the year 2000), we found 16 articles (considering the five from 2001, 27), in the next 5 years more than 75 articles, and in the last 5 years more than 210 articles.

We accomplished a comprehensive process for extracting data which includes 17 facets of each article in the first steps. We extracted important preliminary results, and then applied conceptual modeling techniques to obtain a domain model. The domain model was the base for designing a relational database. A reviewed and enriched version of the data was extracted to be systematized and stored in this database. The database thus populated was included in a web application to make it available for the scientific community. Researchers could exploit, enrich, and suggest modifications to enhance the data in a collaborative way.

Our systematic mapping distinguishes itself from other closer studies, such as (Rattan et al. 2013; Zhang et al. 2011), on the one hand, by defining five research questions that guide the contributions of the present study and, on the other, by tackling the problem in a broader sense.

In this way, we have identified a large number of terms and concepts in the domain of Design Smells that are used without consistency by different authors. According to several classifications, which also differ between them, these terms and concepts are characterized. As a conclusion, the need to standardize such concepts and classifications is clear, for which international collaboration between research teams is essential. This study shows that this kind of collaboration has been weak until now, and an international panel of experts (standard committee) is desirable.

We have classified the detection approaches in 18 possible categories, for which we show their use over time, identifying those that have reached more interest in recent years and highlighting 6 of them as the most used. As a result of our study, we cannot establish any correlation between the techniques used and the efficiency of the results, among other issues, due to the frequency in which these are not used independently but combined among them. Promising combinations can be identified, such as jointly working with Historical information and Context/Feedback-aware-based approaches with others of the most used.

Despite the high number of Design Smells studied (662), most of the works focus on a dozen of them. In fact, an important problem we have found is the absence of an extensive Design Smell Corpus available in common to several detection tools. Comparing with the refactoring area, most of the refactoring tools implement a common corpus of well-known refactoring, some tools introduce small variations in some common operations and add particular refactoring operations. However, the same does not occur regarding Design Smell Detection tools. As a consequence, this hinders adoption in the industry because it is necessary to incorporate a set of different tools in the production workflows to allow the detection of multiple Design Smells or to work with different languages, which significantly complicates the production environment. The emergence of tools such as SonarQube, which allows the execution and reporting of several tools in the same environment, helps to minimize this problem, which has facilitated its success.

We highlight DÉCOR and JDeodorant from the tools used or proposed for Design Smell Detection whose citations stand out from the others. However, in general, there is a lack of maturity in the tools identified with limitations that restrict their use and adoption by the industry. Despite this, there are few studies focused on improving existing tools.

In all cases, the tools offer only a binary response to the presence of Design Smell, but the current context, with increasingly large projects where all corrections are not possible, makes it necessary to evolve towards classification in a set of values (fuzzy classification) that allow the prioritization of the detected Design Smells, based on their impact on quality or the technical debt generated. In particular, neither is there enough work on the impact of Design Smells on quality attributes. It may be important to take the impact of Design Smells on quality attributes into account when considering prioritization.

In the current context, it is very complicated to compare the results obtained by different proposals in the absence of benchmarks and the lack of homogeneity in the performance indicators. Although it is true, that in many cases, a common set of projects is used, mostly from open repositories such as SourceForge or GitHub. However, the comparative studies we found in the study analyze an average of 4 tools and, what is worse, an average of 4 Design Smells in comparisons.

Validation of tools and prototypes is mainly based on "Empirical Studies," "Case Studies," "Experiments," or all of them. To improve validation processes, it would be appropriate to have a reference repository labeled by experts. However, there are many difficulties that arise from this task, such as the absence of a common corpus of Design Smells in tools or the need to evaluate a significant volume of information by highly specialized staff. These reasons also justify the lack of validation by human experts in most of the works. We encourage the community to drive the research in the next few years in these directions.

Acknowledgements This work has received financial support from the Consellera de Cultura, Educacin e Ordenacin Universitaria (accreditation 2016-2019, ED431G/08) and the European Regional Development Fund (ERDF). The authors would like to thank Irene Lavin from Universidad de Valladolid for her work in validating the database, checking integrity, and in the implementation of the web app (smellswisdom).

Appendix. Relevant work included in the systematic mapping

[S1] Abílio, R., Padilha, J., Figueiredo, E., & Costa, H. (2015, April). Detecting code smells in software product lines—an exploratory study. In Information Technology-New Generations (ITNG), 2015 12th International Conference on(pp. 433–438). IEEE.

[S2] Ahmed, I., Mannan, U. A., Gopinath, R., & Jensen, C. (2015, October). An empirical study of design degradation: how software projects get worse over time. In Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on (pp. 1–10). IEEE.

[S3] Almeida, D., Campos, J. C., Saraiva, J., & Silva, J. C. (2015, April). Towards a catalog of usability smells. In Proceedings of the 30th Annual ACM Symposium on Applied Computing (pp. 175–181). ACM.

[S4] Alves, P., Santana, D., & Figueiredo, E. (2012, June). ConcernReCS: finding code smells in software aspectization. In Software Engineering (ICSE), 2012 34th International Conference on (pp. 1463–1464). IEEE.

[S5] Amorim, L., Costa, E., Antunes, N., Fonseca, B., & Ribeiro, M. (2015, November). Experience report: evaluating the effectiveness of decision trees for detecting code smells. In Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on (pp. 261–269). IEEE.

[S6] Arnaoudova, V. (2010, October). Improving source code quality through the definition of linguistic antipatterns. In 17th Working Conference on Reverse Engineering (pp. 285–288).

[S7] Bavota, G., De Lucia, A., Di Penta, M., Oliveto, R., & Palomba, F. (2015). An experimental investigation on the innate relationship between quality and refactoring. Journal of Systems and Software, 107, 1–14.

[S8] Bertran, I. M. (2011, May). Detecting architecturally-relevant code smells in evolving software systems. In Proceedings of the 33rd International Conference on Software Engineering(pp. 1090–1093). ACM.

[S9] Macia, I., Garcia, A., & von Staa, A. (2010, September). Defining and applying detection strategies for aspect-oriented code smells. In Software Engineering (SBES), 2010 Brazilian Symposium on (pp. 60–69). IEEE.

[S10] Bhattacharrya, A., & Fuhrer, R. (2004, October). Smell detection for Eclipse. In Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (pp. 22–22). ACM.

[S11] Boussaa, M., Kessentini, W., Kessentini, M., Bechikh, S., & Chikha, S. B. (2013, August). Competitive coevolutionary code-smells detection. In International Symposium on Search Based Software Engineering (pp. 50–65). Springer, Berlin, Heidelberg.

[S12] Bryton, S., e Abreu, F. B., & Monteiro, M. (2010, September). Reducing subjectivity in code smells detection: experimenting with the Long Method. In Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the (pp. 337–342). IEEE.

[S13] Silva, J. C., Campos, J. C., Saraiva, J., & Silva, J. L. (2014). An approach for graphical user interface external bad smells detection. In New Perspectives in Information Systems and Technologies, Volume 2 (pp. 199–205). Springer, Cham.

[S14] Cardoso, B., & Figueiredo, E. (2015, May). Co-occurrence of design patterns and bad smells in software systems: an exploratory study. In Proceedings of the annual conference on Brazilian symposium on information systems: information systems: a computer socio-technical perspective (pp. 347–354).

[S15] Charalampidou, S., Ampatzoglou, A., & Avgeriou, P. (2015, October). Size and cohesion metrics as indicators of the Long Method bad smell: an empirical study. In Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering (p. 8). ACM.

[S16] Chatzigeorgiou, A., & Manakos, A. (2014). Investigating the evolution of code smells in object-oriented systems. Innovations in Systems and Software Engineering, 10(1), 3–18.

[S17] Chaudron, M. R., Katumba, B., & Ran, X. (2014, August). Automated prioritization of metrics-based design flaws in UML class diagrams. In Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on(pp. 369–376). IEEE.

[S18] Chis, A. E. (2008, October). Automatic detection of memory anti-patterns. In Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications (pp. 925–926). ACM.

[S19] Chivers, H., & Fletcher, M. (2005). Applying security design analysis to a service-based system. Software: Practice and Experience, 35(9), 873–897.

[S20] Choinzon, M., & Ueda, Y. (2006, May). Detecting defects in object oriented designs using design metrics. In Proceedings of the 2006 conference on Knowledge-Based Software Engineering: Proceedings of the Seventh Joint Conference on Knowledge-Based Software Engineering (pp. 61–72). IOS Press.

[S21] Christopoulou, A., Giakoumakis, E. A., Zafeiris, V. E., & Soukara, V. (2012). Automated refactoring to the Strategy design pattern. Information and Software Technology, 54(11), 1202–1214.

[S22] Conceicao, C. F. R., de Figueiredo Carneiro, G., & e Abreu, F. B. (2014, September). Streamlining code smells: using collective intelligence and visualization. In Quality of Information and Communications Technology (QUATIC), 2014 9th International Conference on the (pp. 306–311). IEEE.

[S23] Cortellessa, V., Di Marco, A., & Trubiani, C. (2014). An approach for modeling and detecting software performance antipatterns based on first-order logics. Software & Systems Modeling, 13(1), 391–432.

[S24] Cortellessa, V., Martens, A., Reussner, R., & Trubiani, C. (2010, March). A process to effectively identify "guilty" performance antipatterns. In International Conference on Fundamental Approaches to Software Engineering (pp. 368–382). Springer, Berlin, Heidelberg.

[S25] Counsell, S., & Mendes, E. (2007, October). Size and frequency of class change from a refactoring perspective. In Software Evolvability, 2007 Third International IEEE Workshop on (pp. 23–28). IEEE.

[S26] Crespo, Y., López, C., & Marticorena, R. (1996). Relative thresholds: case study to incorporate metrics in the detection of bad smells. In Proceedings of 10th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering.–Lugano: Universita della Svizzera italiana Press(pp. 109–118).

[S27] Czibula, G., Marian, Z., & Czibula, I. G. (2015). Detecting software design defects using relational association rule mining. Knowledge and Information Systems, 42(3), 545–577.

[S28] Dąbrowski, R., Stencel, K., & Timoszuk, G. (2011, September). Software is a directed multigraph. In European Conference on Software Architecture (pp. 360–369). Springer, Berlin, Heidelberg.

[S29] Danphitsanuphan, P., & Suwantada, T. (2012, May). Code smell detecting tool and code smell-structure bug relationship. In Engineering and Technology (S-CET), 2012 Spring Congress on (pp. 1–5). IEEE.

[S30] von Detten, M., & Becker, S. (2011, June). Combining clustering and pattern detection for the reengineering of component-based software systems. In Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS (pp. 23–32). ACM.

[S31] Dexun, J., Peijun, M., Xiaohong, S., & Tiantian, W. (2012, December). Detecting bad smells with weight based distance metrics theory. In Instrumentation, Measurement, Computer, Communication and Control (IMCCC), 2012 Second International Conference on (pp. 299–304). IEEE.

[S32] Dexun, J., Peijun, M., Xiaohong, S., Tiantian, W. (2013). Distribution rule based bad smell detection and refactoring scheme. International Journal of Software Engineering & Applications (IJSEA) 4(5), 1677–1684.

[S33] Dietrich, J. (2012, October). Upload your program, share your model. In Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity (pp. 21–22). ACM.

[S34] Dijkman, R., Gfeller, B., Küster, J., & Völzer, H. (2011). Identifying refactoring opportunities in process model repositories. Information and Software Technology, 53(9), 937–948.

[S35] Din, J., Al-Badareen, A. B., & Jusoh, Y. Y. (2012, December). Antipatterns detection approaches in object-oriented design: a literature review. In Computing and Convergence Technology (ICCCT), 2012 7th International Conference on (pp. 926–931). IEEE.

[S36] Do Vale, G. A., & Figueiredo, E. M. L. (2015, September). A method to derive metric thresholds for software product lines. In Software Engineering (SBES), 2015 29th Brazilian Symposium on (pp. 110–119). IEEE.

[S37] Dobrzański, Ł., & Kuźniarz, L. (2006, April). An approach to refactoring of executable UML models. In Proceedings of the 2006 ACM symposium on Applied computing (pp. 1273–1279). ACM.

[S38] Drozdz, M., Kourie, D. G., Watson, B. W., & Boake, A. (2006). Refactoring tools and complementary techniques. AICCSA, 6, 685–688.

[S39] Ebert, C., Dumke, R., Bundschuh, M., & Schmietendorf, A. (2005). Best practices in software measurement: how to use metrics to improve project and process performance. Springer Science & Business Media.

[S40] Van Emden, E., & Moonen, L. (2012, October). Assuring software quality by code smell detection. In Reverse Engineering (WCRE), 2012 19th Working Conference on. IEEE.

[S41] Fard, A. M., & Mesbah, A. (2013, September). Jsnose: detecting JavaScript code smells. In Source Code Analysis and Manipulation (SCAM), 2013 IEEE 13th International Working Conference on (pp. 116–125). IEEE.

[S42] Feng, T., Zhang, J., Wang, H., & Wang, X. (2004, September). Software design improvement through anti-patterns identification. In Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on (p. 524). IEEE.

[S43] Fenske, W. (2015, September). Code smells in highly configurable software. In Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on (pp. 602–605). IEEE.

[S44] Fenske, W., & Schulze, S. (2015, January). Code smells revisited: a variability perspective. In Proceedings of the Ninth International Workshop on Variability Modelling of Software-intensive Systems (p. 3). ACM.

[S45] Fenske, W., Schulze, S., Meyer, D., & Saake, G. (2015, September). When code smells twice as much: metric-based detection of variability-aware code smells. In Source Code Analysis and Manipulation (SCAM), 2015 IEEE 15th International Working Conference on (pp. 171–180). IEEE.

[S46] Fernandes, E., Oliveira, J., Vale, G., Paiva, T., & Figueiredo, E. (2016, June). A review-based comparative study of bad smell detection tools. In Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (p. 18). ACM.

[S47] de Figueiredo Carneiro, G., & de Mendonça Neto, M. G. (2013, July). Sourceminer—a multi-perspective software visualization environment. In International Conference on Enterprise Information Systems pp. 25–36). Springer, Cham.

[S48] Carneiro, G. D. F., Silva, M., Mara, L., Figueiredo, E., Sant'Anna, C., Garcia, A., & Mendonca, M. (2010, September). Identifying code smells with multiple concern views. In Software Engineering (SBES), 2010 Brazilian Symposium on (pp. 128–137). IEEE.

[S49] Fokaefs, M., Tsantalis, N., & Chatzigeorgiou, A. (2007, October). Jdeodorant: identification and removal of feature envy bad smells. In Software Maintenance, 2007. ICSM 2007. IEEE International Conference on (pp. 519–520). IEEE.

[S50] Fontana, F. A., Braione, P., & Zanoni, M. (2012). Automatic detection of bad smells in code: an experimental assessment. Journal of Object Technology, 11(2), 5–1.

[S51] Fontana, F. A., Dietrich, J., Walter, B., Yamashita, A., & Zanoni, M. (2016, March). Antipattern and code smell false positives: preliminary conceptualization and classification. In Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on (Vol. 1, pp. 609–613). IEEE.

[S52] Fontana, F. A., Ferme, V., Marino, A., Walter, B., & Martenka, P. (2013, September). Investigating the impact of code smells on system's quality: an empirical study on systems of different application domains. In Software Maintenance (ICSM), 2013 29th IEEE International Conference on (pp. 260–269). IEEE.

[S53] Fontana, F. A., Ferme, V., & Zanoni, M. (2015, May). Filtering code smells detection results. In Proceedings of the 37th International Conference on Software Engineering-Volume 2(pp. 803–804). IEEE Press.

[S54] Fontana, F. A., Ferme, V., & Zanoni, M. (2015, May). Towards assessing software architecture quality by exploiting code smell relations. In Proceedings of the Second International Workshop on Software Architecture and Metrics (pp. 1–7). IEEE Press.

[S55] Fontana, F. A., Ferme, V., Zanoni, M., & Yamashita, A. (2015, May). Automatic metric thresholds derivation for code smell detection. In Emerging Trends in Software Metrics (WETSoM), 2015 IEEE/ACM 6th International Workshop on (pp. 44–53). IEEE.

[S56] Fontana, F. A., Mangiacavalli, M., Pochiero, D., & Zanoni, M. (2015, May). On experimenting refactoring tools to remove code smells. In Scientific Workshop Proceedings of the XP2015 (p. 7). ACM.

[S57] Fontana, F. A., Mäntylä, M. V., Zanoni, M., & Marino, A. (2016). Comparing and experimenting machine learning techniques for code smell detection. Empirical Software Engineering, 21(3), 1143–1191.

[S58] Fontana, F. A., Mariani, E., Mornioli, A., Sormani, R., & Tonello, A. (2011, March). An experience report on using code smells detection tools. In Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on (pp. 450–457). IEEE.

[S59] Fontana, F. A., & Spinelli, S. (2011, May). Impact of refactoring on quality code evaluation. In Proceedings of the 4th Workshop on Refactoring Tools (pp. 37–40). ACM.

[S60] Fontana, F. A., Zanoni, M., Marino, A., & Mantyla, M. V. (2013, September). Code smell detection: towards a machine learning-based approach. In Software Maintenance (ICSM), 2013 29th IEEE International Conference on (pp. 396–399). IEEE.

[S61] Fourati, R., Bouassida, N., & Abdallah, H. B. (2011). A metric-based approach for anti-pattern detection in UML designs. In Computer and Information Science 2011 (pp. 17–33). Springer, Berlin, Heidelberg.

[S62] Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (2000). Refactoring: improving the design of existing code. Addison-Wesley Professional.

[S63] Fu, S., & Shen, B. (2015, October). Code Bad Smell Detection through evolutionary data mining. In Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on (pp. 1–9). IEEE.

[S64] Gabel, M., & Su, Z. (2010, May). Online inference and enforcement of temporal properties. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1 (pp. 15–24). ACM.

[S65] Ganea, G., & Marinescu, R. (2015, September). Modeling design flaw evolution using complex systems. In Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2015 17th International Symposium on (pp. 433–436). IEEE.

[S66] Garcia, J., Popescu, D., Edwards, G., & Medvidovic, N. (2009, March). Identifying architectural bad smells. In Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on (pp. 255–258). IEEE.

[S67] Gauthier, F., & Merlo, E. (2013, May). Semantic smells and errors in access control models: a case study in PHP. In Software Engineering (ICSE), 2013 35th International Conference on (pp. 1169–1172). IEEE.

[S68] Ge, X., Taneja, K., Xie, T., & Tillmann, N. (2011, May). DyTa: dynamic symbolic execution guided with static verification results. In Software Engineering (ICSE), 2011 33rd International Conference on (pp. 992–994). IEEE.

[S69] Ghannem, A., El Boussaidi, G., & Kessentini, M. (2016). On the use of design defect examples to detect model refactoring opportunities. Software Quality Journal, 24(4), 947–965.

[S70] Ghannem, A., Kessentini, M., & El Boussaidi, G. (2011, November). Detecting model refactoring opportunities using heuristic search. In Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research (pp. 175–187). IBM Corp..

[S71] Gîrba, T., Ducasse, S., Kuhn, A., Marinescu, R., & Daniel, R. (2007, September). Using concept analysis to detect co-change patterns. In Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting (pp. 83–89). ACM.

[S72] Guéhéneuc, Y. G., & Albin-Amiot, H. (2001). Using design patterns and constraints to automate the detection and correction of inter-class design defects. In Technology of Object-

Oriented Languages and Systems, 2001. TOOLS 39. 39th International Conference and Exhibition on (pp. 296–305). IEEE.

[S73] Guerra, E., Alves, F., Kulesza, U., & Fernandes, C. (2013). A reference architecture for organizing the internal structure of metadata-based frameworks. Journal of Systems and Software, 86(5), 1239–1256.

[S74] Guerrouj, L., Kermansaravi, Z., Arnaoudova, V., Fung, B. C., Khomh, F., Antoniol, G., & Guéhéneuc, Y. G. (2017). Investigating the relation between lexical smells and change-and fault-proneness: an empirical study. Software Quality Journal, 25(3), 641–670.

[S75] Guo, L. Q., Hsu, K. H., & Tsai, C. Y. (2015, October). A study of the definition and identification of bad smells in aspect oriented programming. In e-Business Engineering (ICEBE), 2015 IEEE 12th International Conference on (pp. 303–310). IEEE.

[S76] Gupta, V., Kapur, P. K., & Kumar, D. (2016). Modelling and measuring code smells in enterprise applications using TISM and two-way assessment. International Journal of System Assurance Engineering and Management, 7(3), 332–340.

[S77] Hallal, H. H., Alikacem, E., Tunney, W. P., Boroday, S., & Petrenko, A. (2004, September). Antipattern-based detection of deficiencies in Java multithreaded software. In Quality Software, 2004. QSIC 2004. Proceedings. Fourth International Conference on (pp. 258–267). IEEE.

[S78] Han, Z., Gong, P., Zhang, L., Ling, J., & Huang, W. (2013, July). Definition and detection of control-flow anti-patterns in process models. In Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual (pp. 433–438). IEEE.

[S79] Hassaine, S., Khomh, F., Guéhéneuc, Y. G., & Hamel, S. (2010, September). IDS: an immune-inspired approach for the detection of software Design Smells. In Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the (pp. 343–348). IEEE.

[S80] Hecht, G. (2015, May). An approach to detect Android antipatterns. In Proceedings of the 37th International Conference on Software Engineering-Volume 2 (pp. 766–768). IEEE Press.

[S81] Hecht, G., Benomar, O., Rouvoy, R., Moha, N., & Duchien, L. (2015, November). Tracking the software quality of android applications along their evolution (t). In Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on (pp. 236–247). IEEE.

[S82] Hecht, G., Rouvoy, R., Moha, N., & Duchien, L. (2015, May). Detecting antipatterns in android apps. In Mobile Software Engineering and Systems (MOBILESoft), 2015 2nd ACM International Conference on (pp. 148–149). IEEE.

[S83] Holzmann, G. J., & Joshi, R. (2008). Reliable software systems design: defect prevention, detection, and containment. In Verified Software: Theories, Tools, Experiments (pp. 237–244). Springer, Berlin, Heidelberg.

[S84] Hozano, M., Ferreira, H., Silva, I., Fonseca, B., & Costa, E. (2015, April). Using developers' feedback to improve code smell detection. In Proceedings of the 30th Annual ACM Symposium on Applied Computing (pp. 1661–1661). ACM.

[S85] Huang, J., Carminati, F., Betev, L., Zhu, J., & Luzzi, C. (2011, October). EXTRACTOR: an extensible framework for identifying Aspect-oriented refactoring opportunities. In System Science, Engineering Design and Manufacturing Informatization (ICSEM), 2011 International Conference on(Vol. 2, pp. 222–226). IEEE.

[S86] Hussain, S. (2016, April). Threshold analysis of design metrics to detect design flaws: student research abstract. In Proceedings of the 31st Annual ACM Symposium on Applied Computing (pp. 1584–1585). ACM.

[S87] Irwanto, D. (2010, December). Visual indicator component software to show component design quality and characteristic. In Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on (pp. 50–54). IEEE.

[S88] Ito, Y., Hazeyama, A., Morimoto, Y., Kaminaga, H., Nakamura, S., & Miyadera, Y. (2015). A method for detecting bad smells and its application to software engineering education. International Journal of Software Innovation (IJSI), 3(2), 13–23.

[S89] Jaafar, F., Guéhéneuc, Y. G., Hamel, S., Khomh, F., & Zulkernine, M. (2016). Evaluating the impact of design pattern and anti-pattern dependencies on changes and faults. Empirical Software Engineering, 21(3), 896–931.

[S90] Jalbani, A. A., Grabowski, J., Neukirchen, H., & Zeiss, B. (2009, September). Towards an integrated quality assessment and improvement approach for UML models. In International SDL Forum (pp. 63–81). Springer, Berlin, Heidelberg.

[S91] Jiang, D., Ma, P., Su, X., & Wang, T. (2014). Distance metric based divergent change bad smell detection and refactoring scheme analysis. Proc. of the 26th Int. Journal of Innovative Computing, Information and Control (ICIC'2014), 10(4).

[S92] Kaur, H., & Kaur, P. J. (2014, September). A GUI based unit testing technique for antipattern identification. In Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference- (pp. 779–782). IEEE.

[S93] Kaur, M., & Kumar, P. (2014, August). Spotting the phenomenon of bad smells in MobileMedia product line architecture. In Contemporary Computing (IC3), 2014 Seventh International Conference on (pp. 357–363). IEEE.

[S94] Kessentini, M., Kessentini, W., Sahraoui, H., Boukadoum, M., & Ouni, A. (2011, June). Design defects detection and correction by example. In Program Comprehension (ICPC), 2011 IEEE 19th International Conference on (pp. 81–90). IEEE.

[S95] Kessentini, M., Sahraoui, H., Boukadoum, M., & Wimmer, M. (2011, March). Design defect detection rules generation: a music metaphor. In Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on (pp. 241–248). IEEE.

[S96] Kessentini, M., Sahraoui, H., Boukadoum, M., & Wimmer, M. (2011, March). Search-based design defects detection by example. In International Conference on Fundamental Approaches to Software Engineering (pp. 401–415). Springer, Berlin, Heidelberg.

[S97] Kessentini, M., Vaucher, S., & Sahraoui, H. (2010, September). Deviance from perfection is a better criterion than closeness to evil when identifying risky code. In Proceedings of the IEEE/ACM international conference on Automated software engineering (pp. 113–122). ACM.

[S98] Kessentini, W., Kessentini, M., Sahraoui, H., Bechikh, S., & Ouni, A. (2014). A cooperative parallel search-based software engineering approach for code-smells detection. IEEE Transactions on Software Engineering, 40(9), 841–861.

[S99] Khomh, F., Vaucher, S., Guéhéneuc, Y. G., & Sahraoui, H. (2009, August). A Bayesian approach for the detection of code and Design Smells. In Quality Software, 2009. QSIC'09. 9th International Conference on (pp. 305–314). IEEE.

[S100] Khomh, F., Vaucher, S., Guéhéneuc, Y. G., & Sahraoui, H. (2011). BDTEX: a GQM-based Bayesian approach for the detection of antipatterns. Journal of Systems and Software, 84(4), 559–572.

[S101] Kiefer, C., Bernstein, A., & Tappolet, J. (2007, May). Mining software repositories with isparol and a software evolution ontology. In Proceedings of the Fourth International Workshop on Mining Software Repositories (p. 10). IEEE Computer Society.

[S102] Kim, T. W., Kim, T. G., & Seu, J. H. (2013). Specification and automated detection of code smells using OCL. International Journal of Software Engineering and Its Applications, 7(4), 35–44.

[S103] Kreimer, J. (2005). Adaptive detection of design flaws. Electronic Notes in Theoretical Computer Science, 141(4), 117–136.

[S104] Kumar, S., & Chhabra, J. K. (2014, September). Two level dynamic approach for Feature Envy detection. In Computer and Communication Technology (ICCCT), 2014 International Conference on (pp. 41–46). IEEE.

[S105] Lanza, M., & Marinescu, R. (2007). Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer Science & Business Media.

[S106] Lee, S. J., Lo, L. H., Chen, Y. C., & Shen, S. M. (2016). Co-changing code volume prediction through association rule mining and linear regression model. Expert Systems with Applications, 45, 185–194.

[S107] Lelli, V., Blouin, A., Baudry, B., Coulon, F., & Beaudoux, O. (2016, June). Automatic detection of GUI Design Smells: the case of blob listener. In Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems(pp. 263–274). ACM.

[S108] Ligu, E., Chatzigeorgiou, A., Chaikalis, T., & Ygeionomakis, N. (2013, September). Identification of refused bequest code smells. In Software Maintenance (ICSM), 2013 29th IEEE International Conference on (pp. 392–395). IEEE.

[S109] Liska, P., & Polasek, I. (2011, September). Design Smell detection with similarity scoring and fingerprinting: preliminary study. In Engineering of Computer Based Systems (ECBS-EERC), 2011 2nd Eastern European Regional Conference on the (pp. 163–164). IEEE.

[S110] Liu, H., Guo, X., & Shao, W. (2013). Monitor-based instant software refactoring. IEEE Transactions on Software Engineering, 39(8), 1112–1126.

[S111] Liu, H., Liu, Q., Niu, Z., & Liu, Y. (2016). Dynamic and automatic feedback-based threshold adaptation for code smell detection. IEEE Transactions on Software Engineering, 42(6), 544–558.

[S112] Liu, H., Ma, Z., Shao, W., & Niu, Z. (2012). Schedule of bad smell detection and resolution: a new way to save effort. IEEE Transactions on Software Engineering, 38(1), 220–235.

[S113] Lozano, A., Mens, K., & Portugal, J. (2015, March). Analyzing code evolution to uncover relations. In Patterns Promotion and Anti-patterns Prevention (PPAP), 2015 IEEE 2nd Workshop on (pp. 1–4). IEEE.

[S114] Ma, H., Ji, Z., Shao, W., & Zhang, L. (2005, September). Towards the uml evaluation using taxonomic patterns on meta-classes. In Quality Software, 2005.(QSIC 2005). Fifth International Conference on (pp. 37–44). IEEE.

[S115] Macia, I., Garcia, A., Chavez, C., & von Staa, A. (2013, March). Enhancing the detection of code anomalies with architecture-sensitive strategies. In Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on(pp. 177–186). IEEE.

[S116] Macia, I., Garcia, J., Popescu, D., Garcia, A., Medvidovic, N., & von Staa, A. (2012, March). Are automatically-detected code anomalies relevant to architectural modularity?: an exploratory analysis of evolving systems. In Proceedings of the 11th annual international conference on Aspect-oriented Software Development (pp. 167–178). ACM.

[S117] Maddeh, M., & Ayouni, S. (2015, May). Extracting and modeling design defects using gradual rules and UML profile. In IFIP International Conference on Computer Science and its Applications_x000D_ (pp. 574–583). Springer, Cham.

[S118] Mahouachi, R., Kessentini, M., & Ghedira, K. (2012, March). A new design defects classification: marrying detection and correction. In International Conference on Fundamental Approaches to Software Engineering (pp. 455–470). Springer, Berlin, Heidelberg.

[S119] Maiga, A., Ali, N., Bhattacharya, N., Sabane, A., Gueheneuc, Y. G., & Aimeur, E. (2012, October). SMURF: a SVM-based incremental anti-pattern detection approach. In Reverse engineering (WCRE), 2012 19th working conference on (pp. 466–475). IEEE.

[S120] Maiga, A., Ali, N., Bhattacharya, N., Sabané, A., Guéhéneuc, Y. G., Antoniol, G., & Aïmeur, E. (2012, September). Support vector machines for anti-pattern detection. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (pp. 278–281). ACM.

[S121] Mannan, U. A., Ahmed, I., Almurshed, R. A. M., Dig, D., & Jensen, C. (2016, May). Understanding code smells in android applications. In Proceedings of the International Workshop on Mobile Software Engineering and Systems (pp. 225–234). ACM.

[S122] Mansoor, U., Kessentini, M., Maxim, B. R., & Deb, K. (2017). Multi-objective code-smells detection using good and bad design examples. Software Quality Journal, 25(2), 529–552.

[S123] Mäntylä, M. V., & Lassenius, C. (2006). Subjective evaluation of software evolvability using code smells: an empirical study. Empirical Software Engineering, 11(3), 395–431.

[S124] Mara, L., Honorato, G., Medeiros, F. D., Garcia, A., & Lucena, C. (2011, March). Hist-Inspect: a tool for history-sensitive detection of code smells. In Proceedings of the tenth international conference on Aspect-oriented software development companion (pp. 65–66). ACM.

[S125] Marinescu, C., Marinescu, R., Mihancea, P.F., Ratiu, D., Wettel, R. (2005). iplasma: an integrated platform for quality assessment of object-oriented design. In International Conference Software Maintenance ICSM (pp. 77–80). IEEE.

[S126] Marinescu, R. (2004, September). Detection strategies: metrics-based rules for detecting design flaws. In Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on (pp. 350–359). IEEE.

[S127] Mathur, N., & Reddy, Y. R. (2015). Correctness of semantic code smell detection tools. In QuASoQ/WAWSE/CMCE@ APSEC (pp. 17–22).

[S128] Mekruksavanich, S., Yupapin, P. P., & Muenchaisri, P. (2012). Analytical learning based on a meta-programming approach for the detection of object-oriented design defects. Information Technology Journal, 11(12), 1677.

[S129] Mihancea, P. F., & Marinescu, R. (2005, March). Towards the optimization of automatic detection of design flaws in object-oriented software systems. In Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on (pp. 92–101). IEEE.

[S130] Moha, N. (2007, October). Detection and correction of design defects in object-oriented designs. In Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion (pp. 949–950). ACM.

[S131] Moha, N., & Guéhéneuc, Y. G. (2007, November). Decor: a tool for the detection of design defects. In Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering (pp. 527–528). ACM.

[S132] Moha, N., & Guéhéneuc, Y. G. (2007, October). P TIDEJ and DÉCOR: identification of design patterns and design defects. In Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion (pp. 868–869). ACM.

[S133] Moha, N., Gueheneuc, Y. G., & Leduc, P. (2006, September). Automatic generation of detection algorithms for design defects. In Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on (pp. 297–300). IEEE.

[S134] Moha, N., Guéhéneuc, Y. G., Le Meur, A. F., & Duchien, L. (2008, March). A domain analysis to specify design defects and generate detection algorithms. In international conference on fundamental approaches to software engineering (pp. 276–291). Springer, Berlin, Heidelberg.

[S135] Moha, N., Guéhéneuc, Y. G., Le Meur, A. F., Duchien, L., & Tiberghien, A. (2010). From a domain analysis to the specification and detection of code and Design Smells. Formal Aspects of Computing, 22(3–4), 345–361.

[S136] Moha, N., Gueheneuc, Y. G., Duchien, L., & Le Meur, A. F. (2010). Decor: A method for the specification and detection of code and Design Smells. IEEE Transactions on Software Engineering, 36(1), 20–36.

[S137] Moha, N., Palma, F., Nayrolles, M., Conseil, B. J., Guéhéneuc, Y. G., Baudry, B., & Jézéquel, J. M. (2012, November). Specification and detection of SOA antipatterns. In International Conference on Service-Oriented Computing (pp. 1–16). Springer, Berlin, Heidelberg.

[S138] Müller, S., Würsch, M., Fritz, T., & Gall, H. C. (2012, June). An approach for collaborative code reviews using multi-touch technology. In Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop on(pp. 93–99). IEEE.

[S139] Munro, M. J. (2005, September). Product metrics for automatic identification of " bad smell" design problems in java source-code. In Software Metrics, 2005. 11th IEEE International Symposium (pp. 15–15). IEEE.

[S140] Murphy-Hill, E., & Black, A. P. (2008, November). Seven habits of a highly effective smell detector. In Proceedings of the 2008 international workshop on Recommendation systems for software engineering (pp. 36–40). ACM.

[S141] Nayrolles, M., Palma, F., Moha, N., & Guéhéneuc, Y. G. (2012, November). Soda: a tool support for the detection of SOA antipatterns. In International Conference on Service-Oriented Computing (pp. 451–455). Springer, Berlin, Heidelberg.

[S142] Neukirchen, H., & Bisanz, M. (2007). Utilising code smells to detect quality problems in TTCN-3 test suites. In Testing of Software and Communicating Systems (pp. 228–243). Springer, Berlin, Heidelberg.

[S143] Neukirchen, H., Zeiss, B., & Grabowski, J. (2008). An approach to quality engineering of TTCN-3 test specifications. International Journal on Software Tools for Technology Transfer, 10(4), 309.

[S144] Nguyen, A. C., & Khoo, S. C. (2011, October). Extracting significant specifications from mining through mutation testing. In International Conference on Formal Engineering Methods(pp. 472–488). Springer, Berlin, Heidelberg.

[S145] Nguyen, H. V., Nguyen, H. A., Nguyen, T. T., Nguyen, A. T., & Nguyen, T. N. (2012, September). Detection of embedded code smells in dynamic web applications. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (pp. 282–285). ACM.

[S146] Nguyen, T., & Pooley, R. (2009, September). Effective recognition of patterns in object-oriented designs. In Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on (pp. 320–325). IEEE.

[S147] Nguyen, T. T., Nguyen, H. A., Pham, N. H., Al-Kofahi, J. M., & Nguyen, T. N. (2009, August). Graph-based mining of multiple object usage patterns. In Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (pp. 383–392). ACM.

[S148] Nödler, J., Neukirchen, H., & Grabowski, J. (2009, April). A flexible framework for quality assurance of software artefacts with applications to java, uml, and ttcn-3 test specifications. In Software Testing Verification and Validation, 2009. ICST'09. International Conference on (pp. 101–110). IEEE.

[S149] Nongpong, K. (2015, January). Feature envy factor: a metric for automatic feature envy detection. In Knowledge and Smart Technology (KST), 2015 7th International Conference on (pp. 7–12). IEEE.

[S150] Oliveira, R. (2016, May). When more heads are better than one? Understanding and improving collaborative identification of code smells. In Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on(pp. 879–882). IEEE.

[S151] Oliveto, R., Gethers, M., Bavota, G., Poshyvanyk, D., & De Lucia, A. (2011, May). Identifying method friendships to remove the feature envy bad smell (NIER track). In Proceedings of the 33rd International Conference on Software Engineering (pp. 820–823). ACM.

[S152] Ouni, A., Kessentini, M., Bechikh, S., & Sahraoui, H. (2015). Prioritizing code-smells correction tasks using chemical reaction optimization. Software Quality Journal, 23(2), 323–361.

[S153] Ouni, A., Kessentini, M., Sahraoui, H., & Boukadoum, M. (2013). Maintainability defects detection and correction: a multi-objective approach. Automated Software Engineering, 20(1), 47–79.

[S154] Ouni, A., Kessentini, M., Sahraoui, H., Inoue, K., & Deb, K. (2016). Multi-criteria code refactoring using search-based software engineering: an industrial case study. ACM Transactions on Software Engineering and Methodology (TOSEM), 25(3), 23.

[S155] Ouni, A., Kessentini, M., Sahraoui, H., Inoue, K., & Hamdi, M. S. (2015). Improving multi-objective code-smells correction using development history. Journal of Systems and Software, 105, 18–39.

[S156] Palma, F. (2012, November). Detection of SOA antipatterns. In International Conference on Service-Oriented Computing(pp. 412–418). Springer, Berlin, Heidelberg.

[S157] Palomba, F. (2015, May). Textual analysis for code smell detection. In Proceedings of the 37th International Conference on Software Engineering-Volume 2 (pp. 769–771). IEEE Press.

[S158] Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., & De Lucia, A. (2014, September). Do they really smell bad? a study on developers' perception of bad code smells. In Software maintenance and evolution (ICSME), 2014 IEEE international conference on (pp. 101–110). IEEE.

[S159] Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A., & Poshyvanyk, D. (2013, November). Detecting bad smells in source code using change history information. In Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (pp. 268–278). IEEE Press.

[S160] Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., Poshyvanyk, D., & De Lucia, A. (2015). Mining version histories for detecting code smells. IEEE Transactions on Software Engineering, 41(5), 462–489.

[S161] Palomba, F., Di Nucci, D., Tufano, M., Bavota, G., Oliveto, R., Poshyvanyk, D., & De Lucia, A. (2015, May). Landfill: an open dataset of code smells with public evaluation. In Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on (pp. 482–485). IEEE.

[S162] Palomba, F., De Lucia, A., Bavota, G., & Oliveto, R. (2014). Anti-pattern detection: methods, challenges, and open issues. In Advances in computers (Vol. 95, pp. 201–238). Elsevier.

[S163] Palomba, F., Panichella, A., De Lucia, A., Oliveto, R., & Zaidman, A. (2016, May). A textual-based technique for smell detection. In Program Comprehension (ICPC), 2016 IEEE 24th International Conference on (pp. 1–10). IEEE.

[S164] Parsons, T. (2004, October). A framework for detecting, assessing and visualizing performance antipatterns in component based systems. In Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (pp. 316–317). ACM.

[S165] Parsons, T., Murphy, J. (2008). Detecting performance antipatterns in component based enterprise systems. Journal of Object Technology, 7(3), 55–90.

[S166] Peiris, M., & Hill, J. H. (2014). Towards detecting software performance anti-patterns using classification techniques. ACM SIGSOFT Software Engineering Notes, 39(1), 1–4.

[S167] Pietrzak, B., & Walter, B. (2006, June). Leveraging code smell detection with inter-smell relations. In International Conference on Extreme Programming and Agile Processes in Software Engineering (pp. 75–84). Springer, Berlin, Heidelberg.

[S168] Piveta, E. K., Hecht, M., Pimenta, M. S., & Price, R. T. (2006). Detecting Bad Smells in AspectJ. Journal of Universal Computer Science, 12(7), 811–827.

[S169] Polášek, I., Snopko, S., & Kapustík, I. (2012, September). Automatic identification of the anti-patterns using the rule-based approach. In Intelligent Systems and Informatics (SISY), 2012 IEEE 10th Jubilee International Symposium on(pp. 283–286). IEEE.

[S170] Polášek, I., Líška, P., Kelemen, J., & Lang, J. (2012, June). On extended similarity scoring and bit-vector algorithms for Design Smell detection. In Intelligent Engineering Systems (INES), 2012 IEEE 16th International Conference on (pp. 115–120). IEEE.

[S171] Pradel, M., Jaspan, C., Aldrich, J., & Gross, T. R. (2012, June). Statically checking API protocol conformance with mined multi-object specifications. In Proceedings of the 34th International Conference on Software Engineering (pp. 925–935). IEEE Press.

[S172] Putro, H. P., & Liem, I. (2011, July). Xml representations of program code. In Electrical Engineering and Informatics (ICEEI), 2011 International Conference on (pp. 1–6). IEEE.

[S173] Qingji, X. (2010). Study on the detection and correction of software based on UML. In International Conference on E-Health Networking, Digital Ecosystems and Technologies (pp. 268–271). IEEE.

[S174] Rao, A.A., Reddy, K.N. (2008). Detecting bad smells in object-oriented design using design change propagation probability matrix. In Proceedings of the International Multi-Conference of Engineers and Computer Scientists, Hong Kong, vol. I..

[S175] Rasool, G., & Arshad, Z. (2015). A review of code smell mining techniques. Journal of Software: Evolution and Process, 27(11), 867–895.

[S176] Rasool, G., & Arshad, Z. (2017). A lightweight approach for detection of code smells. Arabian Journal for Science and Engineering, 42(2), 483–506.

[S177] Rapu, D., Ducasse, S., Gîrba, T., & Marinescu, R. (2004, March). Using history information to improve design flaws detection. In Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on(pp. 223–232). IEEE.

[S178] Ratzinger, J., Fischer, M., Gall, H.C (2005, May). Improving evolvability through refactoring. In Proceedings of the 2005 International Workshop on Mining Software Repositories, MSR 2005.

[S179] Reddy, K. R., & Rao, A. A. (2009). Dependency oriented complexity metrics to detect rippling related design defects. ACM SIGSOFT Software Engineering Notes, 34(4), 1–7.

[S180] Rodriguez, J. M., Crasso, M., & Zunino, A. (2013). An approach for web service discoverability anti-pattern detection for journal of web engineering. Journal of Web Engineering, 12(1–2), 131–158.

[S181] Rodriguez, J. M., Crasso, M., Zunino, A., & Campo, M. (2010). Improving web service descriptions for effective service discovery. Science of Computer Programming, 75(11), 1001–1021.

[S182] Rongviriyapanish, S., Karunlanchakorn, N., & Meananeatra, P. (2015, June). Automatic code locations identification for replacing temporary variable with query method. In Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2015 12th International Conference on (pp. 1–6). IEEE.

[S183] Rosenfeld, M., Fernández, A., & Díaz, A. (2010, June). Semantic wiki refactoring. a strategy to assist semantic wiki evolution. In Fifth Workshop on Semantic Wikis Linking Data and People 7th Extended Semantic Web Conference Hersonissos, Crete, Greece (p. 132).

[S184] Roussey, C., Corcho, O., & Vilches-Blázquez, L. M. (2009, September). A catalogue of OWL ontology antipatterns. In Proceedings of the fifth international conference on Knowledge capture (pp. 205–206). ACM.

[S185] Roussey, C., Corcho, Ó., Sváb-Zamazal, O., Schar e, F., Bernard, S (2012). Antipattern detection in web ontologies: an experiment using SPARQL queries. In Extraction et gestion des connaissances (pp. 321–326).

[S186] Sae-Lim, N., Hayashi, S., & Saeki, M. (2016, May). Context-based code smells prioritization for prefactoring. In Program Comprehension (ICPC), 2016 IEEE 24th International Conference on (pp. 1–10). IEEE.

[S187] Sahin, D., Kessentini, M., Bechikh, S., & Deb, K. (2014). Code-smell detection as a bilevel problem. ACM Transactions on Software Engineering and Methodology (TOSEM), 24(1), 6.

[S188] Santos, J. A. M., & de Mendonça, M. G. (2015, April). Exploring decision drivers on god class detection in three controlled experiments. In Proceedings of the 30th Annual ACM Symposium on Applied Computing (pp. 1472–1479). ACM.

[S189] Santos, J. A. M., de Mendonça, M. G., Dos Santos, C. P., & Novais, R. L. (2014). The problem of conceptualization in god class detection: agreement, strategies and decision drivers. Journal of Software Engineering Research and Development, 2(1), 11.

[S190] Santos, J. A., de Mendonça, M. G., & Silva, C. V. (2013, April). An exploratory study to investigate the impact of conceptualization in god class detection. In Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (pp. 48–59). ACM.

[S191] dos Santos Neto, B. F., Ribeiro, M., Da Silva, V. T., Braga, C., De Lucena, C. J. P., & de Barros Costa, E. (2015). AutoRefactoring: a platform to build refactoring agents. Expert Systems with Applications, 42(3), 1652–1664.

[S192] AyshwaryaLakshmi, S., Mary, S. S. A., & Vadivu, S. S. (2013, July). Agent based tool for topologically sorting badsmells and refactoring by analyzing complexities in source code. In Computing, Communications and Networking Technologies (ICCCNT), 2013 Fourth International Conference on (pp. 1–7). IEEE.

[S193] Schumacher, J., Zazworka, N., Shull, F., Seaman, C., & Shaw, M. (2010, September). Building empirical support for automated code smell detection. In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (p. 8). ACM.

[S194] Settas, D., Cerone, A., & Fenz, S. (2012). Enhancing ontology-based antipattern detection using Bayesian networks. Expert Systems with Applications, 39(10), 9041–9053.

[S195] Settas, D., Meditskos, G., Bassiliades, N., & Stamelos, I. G. (2011, June). Detecting antipatterns using a web-based collaborative antipattern ontology knowledge base. In International Conference on Advanced Information Systems Engineering (pp. 478–488). Springer, Berlin, Heidelberg.

[S196] Settas, D. L., Sowe, S. K., & Stamelos, I. G. (2009). Detecting similarities in antipattern ontologies using semantic social networks: implications for software project management. The Knowledge Engineering Review, 24(3), 287–307.

[S197] Sfayhi, A., & Sahraoui, H. (2011, September). What you see is what you asked for: an effort-based transformation of code analysis tasks into interactive visualization scenarios. In Source Code Analysis and Manipulation (SCAM), 2011 11th IEEE International Working Conference on (pp. 195–203). IEEE.

[S198] Shaikh, M., & Lee, C. G. (2016). Aspect oriented re-engineering of legacy software using cross-cutting concern characterization and significant code smells detection. International Journal of Software Engineering and Knowledge Engineering, 26(03), 513–536.

[S199] Singh, S., & Kahlon, K. S. (2011). Effectiveness of encapsulation and object-oriented metrics to refactor code and identify error prone classes using bad smells. ACM SIGSOFT Software Engineering Notes, 36(5), 1–10.

[S200] Singh, S., & Kahlon, K. S. (2012). Effectiveness of refactoring metrics model to identify smelly and error prone classes in open source software. ACM SIGSOFT Software Engineering Notes, 37(2), 1–11.

[S201] Smith, C. U., & Williams, L. G. (2001, December). Software performance antipatterns; common performance problems and their solutions. In Int. CMG Conference (pp. 797–806).

[S202] Speicher, D., Jancke, S. (2010). Smell detection in context. Softwaretechnik-Trends, 30(2).

[S203] Srivisut, K., & Muenchaisri, P. (2007, July). Defining and detecting bad smells of aspect-oriented software. In Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International (Vol. 1, pp. 65–70). IEEE.

[S204] Stoianov, A., & Şora, I. (2010, May). Detecting patterns and antipatterns in software using Prolog rules. In Computational Cybernetics and Technical Informatics (ICCC-CONTI), 2010 International Joint Conference on (pp. 253–258). IEEE.

[S205] Stolee, K. T., & Elbaum, S. (2011, May). Refactoring pipe-like mashups for end-user programmers. In Proceedings of the 33rd International Conference on Software Engineering (pp. 81–90). ACM.

[S206] Tahmid, A., Nahar, N., & Sakib, K. (2016, March). Understanding the evolution of code smells by observing code smell clusters. In Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on (Vol. 4, pp. 8–11). IEEE.

[S207] Tahvildari, L., & Kontogiannis, K. (2003, March). A metric-based approach to enhance design quality through meta-pattern transformations. In Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on (pp. 183–192). IEEE.

[S208] Tahvildar, L., & Kontogiannis, K. (2004). Improving design quality using meta-pattern transformations: a metric-based approach. Journal of Software: Evolution and Process, 16(4–5), 331–361.

[S209] Tamrawi, A., Nguyen, H. A., Nguyen, H. V., & Nguyen, T. N. (2012, June). Build code analysis with symbolic evaluation. In Proceedings of the 34th International Conference on Software Engineering (pp. 650–660). IEEE Press.

[S210] Tamrawi, A., Nguyen, H. A., Nguyen, H. V., & Nguyen, T. N. (2012, September). SYMake: a build code analysis and refactoring tool for makefiles. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (pp. 366–369). ACM.

[S211] Tekin, U., & Buzluca, F. (2014). A graph mining approach for detecting identical design structures in object-oriented design models. Science of Computer Programming, 95, 406–425.

[S212] Tourwé, T., & Mens, T. (2003, March). Identifying refactoring opportunities using logic meta programming. In Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on (pp. 91–100). IEEE.

[S213] Trifu, A., & Marinescu, R. (2005, November). Diagnosing design problems in object oriented systems. In Reverse Engineering, 12th Working Conference on (pp. 10-pp). IEEE.

[S214] Trubiani, C., & Koziolek, A. (2011, March). Detection and solution of software performance antipatterns in palladio architectural models. In ACM SIGSOFT Software Engineering Notes (Vol. 36, No. 5, pp. 19–30). ACM.

[S215] Trucchia, F., & Romei, J. (2010). Finding “Bad Smells” in code. In Pro PHP Refactoring (pp. 5–24). Apress.

[S216] Tsantalis, N., Chaikalis, T., & Chatzigeorgiou, A. (2008, April). JDeodorant: identification and removal of type-checking bad smells. In Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on (pp. 329–331). IEEE.

[S217] Tsantalis, N., & Chatzigeorgiou, A. (2011, March). Ranking refactoring suggestions based on historical volatility. In Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on (pp. 25–34). IEEE.

[S218] Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., & Poshyvanyk, D. (2015, May). When and why your code starts to smell bad. In Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on (Vol. 1, pp. 403–414). IEEE.

[S219] Vale, G., Figueiredo, E., Abílio, R., & Costa, H. (2014, September). Bad smells in software product lines: a systematic review. In Software Components, Architectures and Reuse (SBCARS), 2014 Eighth Brazilian Symposium on(pp. 84–94). IEEE.

[S220] Van Emden, E., & Moonen, L. (2002). Java quality assurance by detecting code smells. In Reverse Engineering, 2002. Proceedings. Ninth Working Conference on (pp. 97–106). IEEE.

[S221] Verebi, I. (2015, September). A model-based approach to software refactoring. In Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on (pp. 606–609). IEEE.

[S222] Vidal, S., Vazquez, H., Diaz-Pace, J. A., Marcos, C., Garcia, A., & Oizumi, W. (2015, November). JSpIRIT: a flexible tool for the analysis of code smells. In Chilean Computer Science Society (SCCC), 2015 34th International Conference of the(pp. 1–6). IEEE.

[S223] Vidal, S. A., Marcos, C., & Díaz-Pace, J. A. (2016). An approach to prioritize code smells for refactoring. Automated Software Engineering, 23(3), 501–532.

[S224] Štolc, M., & Polášek, I. (2010, January). A visual based framework for the model refactoring techniques. In Applied Machine Intelligence and Informatics (SAMI), 2010 IEEE 8th International Symposium on (pp. 72–82). IEEE.

[S225] Walter, B., & Alkhaeir, T. (2016). The relationship between design patterns and code smells: an exploratory study. Information and Software Technology, 74, 127–142.

[S226] Walter, B., Matuszyk, B., & Fontana, F. A. (2015, May). Including structural factors into the metrics-based code smells detection. In Scientific Workshop Proceedings of the XP2015(p. 11). ACM.

[S227] Walter, B., & Pietrzak, B. (2005, June). Multi-criteria detection of bad smells in code with UTA method. In International Conference on Extreme Programming and Agile Processes in Software Engineering (pp. 154–161). Springer, Berlin, Heidelberg.

[S228] Wang, C., Hirasawa, S., Takizawa, H., & Kobayashi, H. (2014, May). A platform-specific code smell alert system for high performance computing applications. In Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International (pp. 652–661). IEEE.

[S229] Wasylkowski, A., Zeller, A., & Lindig, C. (2007, September). Detecting object usage anomalies. In Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (pp. 35–44). ACM.

[S230] Winkler, D., & Biffl, S. (2006, June). An empirical study on design quality improvement from best-practice inspection and pair programming. In International Conference on Product Focused Software Process Improvement (pp. 319–333). Springer, Berlin, Heidelberg.

[S231] Yamashita, A. (2014). Assessing the capability of code smells to explain maintenance problems: an empirical study combining quantitative and qualitative data. Empirical Software Engineering, 19(4), 1111–1143.

[S232] Yamashita, A., & Counsell, S. (2013). Code smells as system-level indicators of maintainability: an empirical study. Journal of Systems and Software, 86(10), 2639–2653.

[S233] Yamashita, A., & Moonen, L. (2013). To what extent can maintenance problems be predicted by code smell detection?—an empirical study. Information and Software Technology, 55(12), 2223–2242.

[S234] Yamashita, A., Zanoni, M., Fontana, F. A., & Walter, B. (2015, September). Inter-smell relations in industrial and open source systems: A replication and comparative analysis. In Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on (pp. 121–130). IEEE.

[S235] Yamashita, A., & Moonen, L. (2013, October). Do developers care about code smells? an exploratory survey. In Reverse Engineering (WCRE), 2013 20th Working Conference on (pp. 242–251). IEEE.

[S236] Yamashita, A., & Moonen, L. (2013, May). Exploring the impact of inter-smell relations on software maintainability: an empirical study. In Software Engineering (ICSE), 2013 35th International Conference on (pp. 682–691). IEEE.

[S237] Yoshida, N., Saika, T., Choi, E., Ouni, A., & Inoue, K. (2016, May). Revisiting the relationship between code smells and refactoring. In Program Comprehension (ICPC), 2016 IEEE 24th International Conference on (pp. 1–4). IEEE.

[S238] Zamani, B., Butler, G. (2009). Smell detection in UML designs which utilize pattern languages. Iranian Journal of Electrical and Computer Engineer-ing (IJECE), 8(1), 47–52.

[S239] Zazworka, N., Seaman, C., & Shull, F. (2011, May). Prioritizing design debt investment opportunities. In Proceedings of the 2nd Workshop on Managing Technical Debt (pp. 39–42). ACM.

[S240] Zazworka, N., Shaw, M. A., Shull, F., & Seaman, C. (2011, May). Investigating the impact of design debt on software quality. In Proceedings of the 2nd Workshop on Managing Technical Debt (pp. 17–23). ACM.

[S241] Zhang, L., Sun, Y., Song, H., Wang, W., & Huang, G. (2012, October). Detecting anti-patterns in java ee runtime system model. In Proceedings of the Fourth Asia-Pacific Symposium on Internetware (p. 21). ACM.

[S242] Zhang, M., Baddoo, N., Wernick, P., & Hall, T. (2008, October). Improving the precision of fowler's definitions of bad smells. In Software Engineering Workshop, 2008. SEW'08. 32nd Annual IEEE (pp. 161–166). IEEE.

[S243] Abdelmoez, W., Kosba, E., & Iesa, A. F. (2014, January). Risk-based code smells detection tool. In The International Conference on Computing Technology and Information Management (ICCTIM) (p. 148). Society of Digital Information and Wireless Communication.

[S244] Abilio, R., Vale, G., Oliveira, J., Figueiredo, E., & Costa, H. (2014). Code Smell Detection Tool for Compositional-based Software Product Lines. In Proceedings of 21th Brazilian Conference on Software, Tools Session (pp. 109–116).

[S245] Linares-Vásquez, M., Klock, S., McMillan, C., Sabané, A., Poshyvanyk, D., & Guéhéneuc, Y. G. (2014, June). Domain matters: bringing further evidence of the relationships among anti-patterns, application domains, and quality-related metrics in Java mobile apps. In Proceedings of the 22nd International Conference on Program Comprehension (pp. 232–243). ACM.

[S246] Adnan, M., & Afzal, M. (2016, October). A novel approach to super quality software development using workflows. In Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), IEEE Annual (pp. 1–3). IEEE.

[S247] Ahmed, I., Brindescu, C., Mannan, U. A., Jensen, C., & Sarma, A. (2017, November). An empirical examination of the relationship between code smells and merge conflicts. In Empirical Software Engineering and Measurement (ESEM), 2017 ACM/IEEE International Symposium on (pp. 58–67). IEEE.

[S248] Aivaloglou, E., & Hermans, F. (2016, August). How kids code and how we know: an exploratory study on the Scratch repository. In Proceedings of the 2016 ACM Conference on International Computing Education Research (pp. 53–61). ACM.

[S249] Aniche, M., Bavota, G., Treude, C., Gerosa, M. A., & van Deursen, A. (2017). Code smells for model-view-controller architectures. Empirical Software Engineering, 1–37.

[S250] Aniche, M., Bavota, G., Treude, C., Van Deursen, A., & Gerosa, M. A. (2016, October). A validated set of smells in model-view-controller architectures. In Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on (pp. 233–243). IEEE.

[S251] Aras, M. T., & Selçuk, Y. E. (2016, July). Metric and rule based automated detection of antipatterns in object-oriented software systems. In Computer Science and Information Technology (CSIT), 2016 7th International Conference on (pp. 1–6). IEEE.

[S252] Fontana, F. A., & Zanoni, M. (2017). Code smell severity classification using machine learning techniques. Knowledge-Based Systems, 128, 43–58.

[S253] Beena, R. (2015). A study of code smells in software versioning system using data mining techniques. 3, 37–39.

[S254] Bowes, D., Randall, D., & Hall, T. (2013, May). The inconsistent measurement of message chains. In Emerging Trends in Software Metrics (WETSoM), 2013 4th International Workshop on (pp. 62–68). IEEE.

[S255] Brabra, H., Mtibaa, A., Sliman, L., Gaaloul, W., Benatallah, B., & Gargouri, F. (2016, October). Detecting cloud (anti) patterns: OCCI perspective. In International Conference on Service-Oriented Computing (pp. 202–218). Springer, Cham.

[S256] Carette, A., Younes, M. A. A., Hecht, G., Moha, N., & Rouvoy, R. (2017, February). Investigating the energy impact of android smells. In Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on (pp. 115–126). IEEE.

[S257] Charalampidou, S., Ampatzoglou, A., Chatzigeorgiou, A., & Avgeriou, P. (2017, May). Assessing code smell interest probability: a case study. In Proceedings of the XP2017 Scientific Workshops (p. 5). ACM.

[S258] Chen, W. K., & Wang, J. C. (2012, August). Bad smells and refactoring methods for gui test scripts. In Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on (pp. 289–294). IEEE.

[S259] Chen, Z., Chen, L., Ma, W., & Xu, B. (2016, November). Detecting code smells in Paython programs. In Software Analysis, Testing and Evolution (SATE), International Conference on (pp. 18–23). IEEE.

[S260] Choudhary, A. & Singh, P. (2016). Minimizing refactoring effort through prioritization of classes based on historical, architectural and code smell information. CEUR Workshop Proc. 1771, (pp.76–79).

[S261] Codabux, Z., Sultana, K. Z., & Williams, B. J. (2017). The relationship between traceable code patterns and code smells. In Proc. 29th Int. Conf. Software Engineering and Knowledge Engineering.

[S262] Counsell, S., Hierons, R. M., Hamza, H., Black, S., & Durrand, M. (2010, May). Is a strategy for code smell assessment long overdue?. In Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics (pp. 32–38). ACM.

[S263] da Silva Carvalho, L. P., Novais, R., do Nascimento Salvador, L., & de Mendonça Neto, M. G. (2017). An ontology-based approach to analyzing the occurrence of code smells in software.

[S264] da Silva Sousa, L. (2016, May). Spotting design problems with smell agglomerations. In Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on (pp. 863–866). IEEE.

[S265] D'Ambros, M., Bacchelli, A., & Lanza, M. (2010, July). On the impact of design flaws on software defects. In Quality Software (QSIC), 2010 10th International Conference on (pp. 23–31). IEEE.

[S266] de Andrade, H. S., Almeida, E., & Crnkovic, I. (2014, April). Architectural bad smells in software product lines: an exploratory study. In Proceedings of the WICSA 2014 Companion Volume (p. 12). ACM.

[S267] de Mello, R., Oliveira, R., Sousa, L., & Garcia, A. (2017, May). Towards effective teams for the identification of code smells. In Cooperative and Human Aspects of Software Engineering (CHASE), 2017 IEEE/ACM 10th International Workshop on(pp. 62–65). IEEE.

[S268] Delchev, M., & Harun, M. F. (2015). Investigation of code smells in different software domains. Full-scale Software Engineering, 31.

[S269] Dhaka, G., & Singh, P. (2016, December). An empirical investigation into code smell elimination sequences for energy efficient software. In Software Engineering Conference (APSEC), 2016 23rd Asia-Pacific (pp. 349–352). IEEE.

[S270] Dhillon, P. K., & Sidhu, G. (2012). Can software faults be analyzed using bad code smells?: an empirical study. International Journal of Scientific and Research Publications, 2(10), 1–7.

[S271] dos Reis, J. P., e Abreu, F. B., & Carneiro, G. D. F. (2016, September). Code smells incidence: does it depend on the application domain?. In Quality of Information and Communications Technology (QUATIC), 2016 10th International Conference on the (pp. 172–177). IEEE.

[S272] Ferme, V., Marino, A., & Fontana, F. A. (2013). Is it a Real Code Smell to be Removed or not?. In International Workshop on Refactoring & Testing (RefTest), co-located event with XP 2013 Conference.

[S273] Ferreira, M., Barbosa, E., Macia, I., Arcoverde, R., & Garcia, A. (2014, March). Detecting architecturally-relevant code anomalies: a case study of effectiveness and effort. In Proceedings of the 29th Annual ACM Symposium on Applied Computing (pp. 1158–1163). ACM.

[S274] Fontana, F. A., Ferme, V., & Spinelli, S. (2012, June). Investigating the impact of code smells debt on quality code evaluation. In Proceedings of the Third International Workshop on Managing Technical Debt (pp. 15–22). IEEE Press.

[S275] Fontana, F. A., Ferme, V., Zanoni, M., & Roveda, R. (2015, October). Towards a prioritization of code debt: a code smell intensity index. In Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on (pp. 16–24). IEEE.

[S276] Fontana, F. A., & Maggioni, S. (2011, June). Metrics and antipatterns for software quality evaluation. In Software Engineering Workshop (SEW), 2011 34th IEEE (pp. 48–56). IEEE.

[S277] Fontana, F. A., Pigazzini, I., Roveda, R., Tamburri, D., Zanoni, M., & Di Nitto, E. (2017, April). Arcan: a tool for architectural smells detection. In Software Architecture Workshops (ICSAW), 2017 IEEE International Conference on (pp. 282–285). IEEE.

[S278] Fontana, F. A., Pigazzini, I., Roveda, R., Tamburri, D., Zanoni, M., & Di Nitto, E. (2017, April). Arcan: a tool for architectural smells detection. In Software Architecture Workshops (ICSAW), 2017 IEEE International Conference on (pp. 282–285). IEEE.

[S279] Fontana, F. A., Roveda, R., & Zanoni, M. (2016, April). Tool support for evaluating architectural debt of an existing system: an experience report. In Proceedings of the 31st Annual ACM Symposium on Applied Computing (pp. 1347–1349). ACM.

[S280] Suryanarayana, G., Samarthyam, G., & Sharma, T. (2014). Refactoring for software Design Smells: managing technical debt. Morgan Kaufmann.

[S281] Grigera, J., Garrido, A., & Rivero, J. M. (2014, July). A tool for detecting bad usability smells in an automatic way. In International Conference on Web Engineering (pp. 490–493). Springer, Cham.

[S282] Guimaraes, E., Garcia, A., Figueiredo, E., & Cai, Y. (2013, May). Prioritizing software anomalies with software metrics and architecture blueprints. In Modeling in Software Engineering (MiSE), 2013 5th International Workshop on (pp. 82–88). IEEE.

[S283] Gull, M., Zia, T., & Ilyas, M. (2017). Source code author attribution using author's programming style and code smells. International Journal of Intelligent Systems and Applications, 9(5), 27.

[S284] Guo, Y., Seaman, C., Zazworka, N., & Shull, F. (2010, May). Domain-specific tailoring of code smells: an empirical study. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2 (pp. 167–170). ACM.

[S285] Habchi, S., Hecht, G., Rouvoy, R., & Moha, N. (2017, May). Code smells in iOS apps: how do they compare to Android?. In Proceedings of the 4th International Conference on Mobile Software Engineering and Systems (pp. 110–121). IEEE Press.

[S286] Haendler, T., Sobernig, S., & Strembeck, M. (2017, May). Towards triaging code-smell candidates via runtime scenarios and method-call dependencies. In Proceedings of the XP2017 Scientific Workshops (p. 8). ACM.

[S287] Hall, T., Zhang, M., Bowes, D., & Sun, Y. (2014). Some code smells have a significant but small effect on faults. ACM Transactions on Software Engineering and Methodology (TOSEM), 23(4), 33.

[S288] Hecht, G., Moha, N., & Rouvoy, R. (2016, May). An empirical study of the performance impacts of android code smells. In Proceedings of the International Conference on Mobile Software Engineering and Systems (pp. 59–69). ACM.

[S289] Hermans, F., & Aivaloglou, E. (2016, May). Do code smells hamper novice programming? A controlled experiment on Scratch programs. In Program Comprehension (ICPC), 2016 IEEE 24th International Conference on (pp. 1–10). IEEE.

[S290] Hozano, M., Antunes, N., Fonseca, B. & Costa, E. (2017). Evaluating the accuracy of machine learning algorithms on detecting code smells for different developers. Proc. 19th Int. Conf. Enterp. Inf. Syst. (pp. 474–482).

[S291] Hozano, M., Garcia, A., Antunes, N., Fonseca, B., & Costa, E. (2017, May). Smells are sensitive to developers!: on the efficiency of (un) guided customized detection. In Proceedings of the 25th International Conference on Program Comprehension (pp. 110–120). IEEE Press.

[S292] Husien, H. K., Harun, M. F., & Lichter, H. (2017). Towards a severity and activity based assessment of code smells. Procedia Computer Science, 116, 460–467.

[S293] Jaafar, F., Guéhéneuc, Y. G., Hamel, S., & Khomh, F. (2013, October). Mining the relationship between anti-patterns dependencies and fault-proneness. In Reverse Engineering (WCRE), 2013 20th Working Conference on (pp. 351–360). IEEE.

[S294] Jiang, Y., Li, M., & Zhou, Z. H. (2011). Software defect detection with Rocus. Journal of Computer Science and Technology, 26(2), 328–342.

[S295] Kaur, A., Kaur, K., & Jain, S. (2016, September). Predicting software change-proneness with code smells and class imbalance learning. In Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on (pp. 746–754). IEEE.

[S296] Keck, P., Van Hoorn, A., Okanović, D., Pitakrat, T., & Düllmann, T. F. (2016, October). Antipattern-based problem injection for assessing performance and reliability evaluation techniques. In Software Reliability Engineering Workshops (ISSREW), 2016 IEEE International Symposium on (pp. 64–70). IEEE.

[S297] Kessentini, M., & Ouni, A. (2017, May). Detecting Android smells using multi-objective genetic programming. In Proceedings of the 4th International Conference on Mobile Software Engineering and Systems (pp. 122–132). IEEE Press.

[S298] Khomh, F., Penta, M. D., Gueheneuc, Y. G., & Antoniol, G. (2009). An exploratory study of the impact of antipatterns on software changeability. École Polytechnique de Montréal, Tech. Rep. EPM-RT-2009-02.

[S299] Khomh, F., Di Penta, M., & Gueheneuc, Y. G. (2009, October). An exploratory study of the impact of code smells on software change-proneness. In Reverse Engineering, 2009. WCRE'09. 16th Working Conference on (pp. 75–84). IEEE.

[S300] Khumnin, P., & Senivongse, T. (2017, June). SQL antipatterns detection and database refactoring process. In Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2017 18th IEEE/ACIS International Conference on (pp. 199–205). IEEE.

[S301] Kim, D. K. (2017). Finding bad code smells with neural network models. International Journal of Electrical and Computer Engineering (IJECE), 7(6), 3613–3621

[S302] Krishna, R., Menzies, T., & Layman, L. (2017). Less is more: minimizing code reorganization using XTREE. Information and Software Technology, 88, 53–66.

[S303] AyshwaryaLakshmi, S., Shanmuga Vadivu, S., & Ramachandran, A. (2013). Detecting and scheduling bad smells using Java Agent Development (JADE). International Journal of Computer Applications, 67(10), 29–37.

[S304] Lenhard, J., Hassan, M. M., Blom, M., & Herold, S. (2017, September). Are code smell detection tools suitable for detecting architecture degradation?. In Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings (pp. 138–144). ACM.

[S305] Li, W., & Shatnawi, R. (2007). An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution. Journal of Systems and Software, 80(7), 1120–1128.

[S306] Llano, M. T., & Pooley, R. (2009, September). UML specification and correction of object-oriented anti-patterns. In Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on (pp. 39–44). IEEE.

[S307] Lozano, A., Wermelinger, M., & Nuseibeh, B. (2007, September). Assessing the impact of bad smells using historical information. In Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting (pp. 31–34). ACM.

[S308] Ma, W., Chen, L., Zhou, Y., & Xu, B. (2016, November). Do we have a chance to fix bugs when refactoring code smells?. In Software Analysis, Testing and Evolution (SATE), International Conference on (pp. 24–29). IEEE.

[S309] Macia Bertran, I., Garcia, A., & von Staa, A. (2011, March). An exploratory study of code smells in evolving aspect-oriented systems. In Proceedings of the tenth international conference on Aspect-oriented software development (pp. 203–214). ACM.

[S310] Macia, I., Garcia, J., Popescu, D., Garcia, A., Medvidovic, N., & von Staa, A. (2012, March). Are automatically-detected code anomalies relevant to architectural sustainability? In Proceedings of the 11th annual international conference on Aspect-oriented Software Development (pp. 167–178). ACM.

[S311] Malhotra, R., Chug, A., & Khosla, P. (2015, August). Prioritization of classes for refactoring: a step towards improvement in software quality. In Proceedings of the Third International Symposium on Women in Computing and Informatics (pp. 228–234). ACM.

[S312] Maneerat, N., & Muenchaisri, P. (2011, May). Bad-smell prediction from software design model using machine learning techniques. In Computer Science and Software Engineering (JCSSE), 2011 Eighth International Joint Conference on (pp. 331–336). IEEE.

[S313] Mäntylä, M. V., Vanhanen, J. & Lassenius, C. (2004). Bad smells—humans as code critics. In Proceedings 20th IEEE Int. Conf. Softw. Maintenance (pp. 399–408).

[S314] Marinescu, R. (2001). Detecting design flaws via metrics in object-oriented systems. In Technology of Object-Oriented Languages and Systems, 2001. TOOLS 39. 39th International Conference and Exhibition on (pp. 173–182). IEEE.

[S315] Marinescu, R. (2012). Assessing technical debt by identifying design flaws in software systems. IBM Journal of Research and Development, 56(5), 9–1.

[S316] Mekruksavanich, S. (2017, March). An adaptive approach for automatic design defects detection in object-oriented systems. In Digital Arts, Media and Technology (ICDAMT), International Conference on (pp. 342–346). IEEE.

[S317] Santos, J. A. M., & Mendonça, M. G. (2014). Identifying strategies on god class detection in two controlled experiments. In SEKE (pp. 244–249).

[S318] Mkaouer, M. W. (2016, October). Interactive code smells detection: an initial investigation. In International Symposium on Search Based Software Engineering (pp. 281–287). Springer, Cham.

[S319] Mo, R., Cai, Y., Kazman, R., & Xiao, L. (2015, May). Hotspot patterns: the formal definition and automatic detection of architecture smells. In Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on (pp. 51–60). IEEE.

[S320] Alkharabsheh, K., Almobydeen, S., Taboada, J. A., & Crespo, Y. (2016). Influence of nominal project knowledge in the detection of design smells: an exploratory study with God Class. International Journal of Advanced Studies in Computers, Science and Engineering, 5(11), 120.

[S321] Moha, N., & Guéhéneuc, Y. G. (2005, July). On the automatic detection and correction of software architectural defects in object-oriented designs. In Proceedings of the 4th ECOOP Workshop on Object-Oriented Reengineering.

[S322] Murphy-Hill, E., & Black, A. P. (2010, October). An interactive ambient visualization for code smells. In Proceedings of the 5th international symposium on Software visualization (pp. 5–14). ACM.

[S323] Nahar, N., & Sakib, K. (2016, March). ACDPR: a recommendation system for the creational design patterns using anti-patterns. In Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on (Vol. 4, pp. 4–7). IEEE.

[S324] Nanthaamornphong, A., & Chaisutanon, A. (2016, September). Empirical evaluation of code smells in open source projects: preliminary results. In Proceedings of the 1st International Workshop on Software Refactoring (pp. 5–8). ACM.

[S325] Asano, K., Hayashi, S., & Saeki, M. (2017, November). Detecting bad smells of refinement in goal-oriented requirements analysis. In International Conference on Conceptual Modeling (pp. 122–132). Springer, Cham.

[S326] Dhambri, K., Sahraoui, H., & Poulin, P. (2008, April). Visual detection of design anomalies. In Software maintenance and reengineering, 2008. CSMR 2008. 12th European Conference on (pp. 279–283). IEEE.

[S327] Oizumi, W. N., Garcia, A. F., Colanzi, T. E., Ferreira, M., & Staa, A. V. (2015). On the relationship of code-anomaly agglomerations and architectural problems. Journal of Software Engineering Research and Development, 3(1), 11.

[S328] Oizumi, W., Sousa, L., Garcia, A., Oliveira, R., Oliveira, A., Agbachi, O. I., & Lucena, C. (2017). Revealing design problems in stinky code. In Proceedings of the 11th Brazilian Symposium on Software Components, Architectures, and Reuse (pp. 1–10).

[S329] Olbrich, S. M., Cruzes, D. S., & Sjøberg, D. I. (2010, September). Are all code smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems. In Software Maintenance (ICSM), 2010 IEEE International Conference on (pp. 1–10). IEEE.

[S330] Olbrich, S., Cruzes, D. S., Basili, V., & Zazworka, N. (2009, October). The evolution and impact of code smells: a case study of two open source systems. In Proceedings of the 2009 3rd international symposium on empirical software engineering and measurement (pp. 390–400). IEEE Computer Society.

[S331] de Oliveira, J. A., Fernandes, E. M., & Figueiredo, E. (2015). Evaluation of duplicated code detection tools in cross-project context. In Proceedings of the 3rd Workshop on Software Visualization, Evolution, and Maintenance (pp. 49–56).

[S332] Oliveira, R., Estácio, B., Garcia, A., Marczak, S., Prikladnicki, R., Kalinowski, M., & Lucena, C. (2016, September). Identifying code smells with collaborative practices: a controlled experiment. In Software Components, Architectures and Reuse (SBCARS), 2016 X Brazilian Symposium on (pp. 61–70). IEEE.

[S333] Oliveira, R., Sousa, L., de Mello, R., Valentim, N., Lopes, A., Conte, T., ... & Lucena, C. (2017, May). Collaborative identification of code smells: a multi-case study. In Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), 2017 IEEE/ACM 39th International Conference on (pp. 33–42). IEEE.

[S334] Oliveto, R., Khomh, F., Antoniol, G., & Guéhéneuc, Y. G. (2011, March). Numerical signatures of antipatterns: an approach based on b-splines. In Software maintenance and reengineering (CSMR), 2012 14th European Conference on(pp. 248–251). IEEE.

[S335] Ouni, A., Kessentini, M., Inoue, K., & Cinnéide, M. O. (2017). Search-based web service antipatterns detection. IEEE Transactions on Services Computing, 10(4), 603–617.

[S336] Padilha, J., Pereira, J., Figueiredo, E., Almeida, J., Garcia, A., & Sant'Anna, C. (2014, June). On the effectiveness of concern metrics to detect code smells: an empirical study. In International Conference on Advanced Information Systems Engineering (pp. 656–671). Springer, Cham.

[S337] Paiva, T., Damasceno, A., Figueiredo, E., & Sant'Anna, C. (2017). On the evaluation of code smells and detection tools. Journal of Software Engineering Research and Development, 5(1), 7.

[S338] Paiva, T., Damasceno, A., Padilha, J., Figueiredo, E. & Sant'Anna, C. (2015). Experimental evaluation of code smell detection tools. 3th Work. Softw. Vis. Evol. Maint.

[S339] Palomba, F. (2016, October). Alternative sources of information for code smell detection: postcards from far away. In Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on (pp. 636–640). IEEE.

[S340] Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R., & De Lucia, A. (2017). On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. Empirical Software Engineering, 1–34.

[S341] Palomba, F., Di Nucci, D., Panichella, A., Zaidman, A., & De Lucia, A. (2017, February). Lightweight detection of Android-specific code smells: the aDoctor project. In Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on (pp. 487–491). IEEE.

[S342] Palomba, F., Oliveto, R., & De Lucia, A. (2017, February). Investigating code smell co-occurrences using association rule learning: a replicated study. In Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE), IEEE Workshop on (pp. 8–13). IEEE.

[S343] Palomba, F., Panichella, A., Zaidman, A., Oliveto, R., & De Lucia, A. (2017). The scent of a smell: an extensive comparison between textual and structural smells. IEEE Transactions on Software Engineering.

[S344] Palomba, F., Zanoni, M., Fontana, F. A., De Lucia, A., & Oliveto, R. (2017). Toward a smell-aware bug prediction model. IEEE Transactions on Software Engineering.

[S345] Palomba, F., Zanoni, M., Fontana, F. A., De Lucia, A., & Oliveto, R. (2016, October). Smells like teen spirit: improving bug prediction performance using the intensity of code smells. In Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on (pp. 244–255). IEEE.

[S346] Parnin, C., Görg, C., & Nnadi, O. (2008, September). A catalogue of lightweight visualizations to support code smell inspection. In Proceedings of the 4th ACM symposium on Software visualization (pp. 77–86). ACM.

[S347] Peiris, M., & Hill, J. H. (2016, March). Automatically detecting excessive dynamic memory allocations software performance anti-pattern. In Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering (pp. 237–248). ACM.

[S348] Peldszus, S., Kulcsár, G., Lochau, M., & Schulze, S. (2016, September). Continuous detection of design flaws in evolving object-oriented programs using incremental multi-pattern matching. In Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on (pp. 578–589). IEEE.

[S349] Dos Reis, J. P., e Abreu, F. B., & Carneiro, G. D. F. (2017, June). Code smells detection 2.0: crowdsmelling and visualization. In Information Systems and Technologies (CISTI), 2017 12th Iberian Conference on (pp. 1–4). IEEE.

[S350] Pessoa, T., Abreu, F. B. E., Monteiro, M. P. & Bryton, S. (2011). An Eclipse plugin to support code smells detection. INFORUM 2011- Simpósio de Informática 12.

[S351] Peters, R., & Zaidman, A. (2012, March). Evaluating the lifespan of code smells using software repository mining. In Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on (pp. 411–416). IEEE.

[S352] Rani, A., & Chhabra, J. K. (2017, February). Prioritization of smelly classes: a two phase approach (Reducing refactoring efforts). In Computational Intelligence & Communication Technology (CICT), 2017 3rd International Conference on (pp. 1–6). IEEE.

[S353] Ratiu, D., Marinescu, R., Ducasse, S., & Gırba, T. (2004). Evolution-enriched detection of God Classes. Proc. of the 2nd CAVIS, 3–7.

[S354] Verma, A., Kumar, A., & Kaur, I. (2017). Automatic multiprogramming bad smell detection with refactoring. International Journal of Advanced and Applied Sciences, 4(9), 80–85.

[S355] Rio, A. & e Abreu, F. (2016). Web systems quality evolution. 10th Int. Conf. Qual. Inf. Commun. Technol. 1, (pp. 248–253).

[S356] Rio, A., & e Abreu, F. B. (2017, June). Analyzing web applications quality evolution. In Information Systems and Technologies (CISTI), 2017 12th Iberian Conference on (pp. 1–4). IEEE.

[S357] Romano, D., Raila, P., Pinzger, M., & Khomh, F. (2012, October). Analyzing the impact of antipatterns on change-proneness using fine-grained source code changes. In Reverse Engineering (WCRE), 2012 19th Working Conference on (pp. 437–446). IEEE.

[S358] Sabane, A., Di Penta, M., Antoniol, G., & Guéhéneuc, Y. G. (2013, March). A study on the relation between antipatterns and the cost of class unit testing. In Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on (pp. 167–176). IEEE.

[S359] Saboury, A., Musavi, P., Khomh, F., & Antoniol, G. (2017, February). An empirical study of code smells in JavaScript projects. In Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on (pp. 294–305). IEEE.

[S360] Sae-Lim, N., Hayashi, S., & Saeki, M. (2017, May). Revisiting context-based code smells prioritization: on supporting referred context. In Proceedings of the XP2017 Scientific Workshops (p. 3). ACM.

[S361] Sae-Lim, N., Hayashi, S., & Saeki, M. (2017, September). How do developers select and prioritize code smells? A preliminary study. In Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on (pp. 484–488). IEEE.

[S362] Salehie, M., Li, S., & Tahvildari, L. (2006, June). A metric-based heuristic framework to detect object-oriented design flaws. In Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on (pp. 159–168). IEEE.

[S363] Sangeetha, M., & Sengottuvelan, P. (2017, January). Systematic exhortation of code smell detection using JSmell for Java source code. In Inventive Systems and Control (ICISC), 2017 International Conference on (pp. 1–5). IEEE.

[S364] Santos, J. A. M., Rocha-Junior, J. B., & de Mendonça, M. G. (2017). Investigating factors that affect the human perception on god class detection: an analysis based on a family of four controlled experiments. Journal of Software Engineering Research and Development, 5(1), 8.

[S365] Kaur, K., & Jain, S. (2017). Evaluation of machine learning approaches for change-proneness prediction using code smells. In Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications(pp. 561–572). Springer, Singapore.

[S366] Serikawa, M. A., Landi, A. D. S., Siqueira, B. R., Costa, R. S., Ferrari, F. C., Menotti, R., & de Camargo, V. V. (2016, September). Towards the characterization of monitor smells in adaptive systems. In Software Components, Architectures and Reuse (SBCARS), 2016 X Brazilian Symposium on (pp. 51–60). IEEE.

[S367] Sharma, T., Fragkoulis, M., & Spinellis, D. (2016, May). Does your configuration code smell?. In Mining Software Repositories (MSR), 2016 IEEE/ACM 13th Working Conference on (pp. 189–200). IEEE.

[S368] Taibi, D., Janes, A., & Lenarduzzi, V. (2016, May). Towards a lean approach to reduce code smells injection: an empirical study. In International Conference on Agile Software Development (pp. 300–304). Springer, Cham.

[S369] Shatnawi, R., & Li, W. (2006, April). An investigation of bad smells in object-oriented design. In Information Technology: New Generations, 2006. ITNG 2006. Third International Conference on (pp. 161–165). IEEE.

[S370] Shoenberger, I., Mkaouer, M. W., & Kessentini, M. (2017, April). On the use of smelly examples to detect code smells in JavaScript. In European Conference on the Applications of Evolutionary Computation (pp. 20–34). Springer, Cham.

[S371] Singh, S., & Kaur, S. (2017). A systematic literature review: refactoring for disclosing code smells in object oriented software. Ain Shams Engineering Journal.

[S372] Sirikul, K., & Soomlek, C. (2016, July). Automated detection of code smells caused by null checking conditions in Java programs. In Computer Science and Software Engineering (JCSSE), 2016 13th International Joint Conference on (pp. 1–7). IEEE.

[S373] Sjøberg, D. I., Yamashita, A., Anda, B. C., Mockus, A., & Dybå, T. (2013). Quantifying the effect of code smells on maintenance effort. IEEE Transactions on Software Engineering, 39(8), 1144–1156.

[S374] Soh, Z., Yamashita, A., Khomh, F., & Guéhéneuc, Y. G. (2016, March). Do code smells impact the effort of different maintenance programming activities?. In Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on (Vol. 1, pp. 393–402). IEEE.

[S375] Soltanifar, B., Akbarinasaji, S., Caglayan, B., Bener, A. B., Filiz, A., & Kramer, B. M. (2016, July). Software analytics in practice: a defect prediction model using code smells. In Proceedings of the 20th International Database Engineering & Applications Symposium (pp. 148–155). ACM.

[S376] Sousa, B. L., Souza, P. P., Fernandes, E. M., Ferreira, K. A., & Bigonha, M. A. (2017, May). FindSmells: flexible composition of bad smell detection strategies. In Program Comprehension (ICPC), 2017 IEEE/ACM 25th International Conference on(pp. 360–363). IEEE.

[S377] Srivisut, K., & Muenchaisri, P. (2007, July). Bad-smell metrics for aspect-oriented software. In Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on (pp. 1060–1065). IEEE.

[S378] Steinbeck, M. (2017, February). An arc-based approach for visualization of code smells. In Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on (pp. 397–401). IEEE.

[S379] Padilha, J., Figueiredo, E., Sant'Anna, C., & Garcia, A. (2013). Detecting god methods with concern metrics: an exploratory study. In Latin-American Workshop on Aspect-Oriented Software Development.

[S380] Taibi, D., Janes, A., & Lenarduzzi, V. (2017). How developers perceive smells in source code: a replicated study. Information and Software Technology, 92, 223–235.

[S381] Techapalokul, P. (2017, March). Sniffing through millions of blocks for bad smells. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (pp. 781–782). ACM.

[S382] Vaucher, S., Khomh, F., Moha, N., & Guéhéneuc, Y. G. (2009, October). Tracking design smells: lessons from a study of God Classes. In Reverse Engineering, 2009. WCRE'09. 16th Working Conference on (pp. 145–154). IEEE.

[S383] Velioğlu, S., & Selçuk, Y. E. (2017, June). An automated code smell and anti-pattern detection approach. In Software Engineering Research, Management and Applications (SERA), 2017 IEEE 15th International Conference on (pp. 271–275). IEEE.

[S384] Vidal, S., Guimaraes, E., Oizumi, W., Garcia, A., Pace, A. D., & Marcos, C. (2016, September). Identifying architectural problems through prioritization of code smells. In Software Components, Architectures and Reuse (SBCARS), 2016 X Brazilian Symposium on (pp. 41–50). IEEE.

[S385] Vidal, S., Guimaraes, E., Oizumi, W., Garcia, A., Pace, A. D., & Marcos, C. (2016, April). On the criteria for prioritizing code anomalies to identify architectural problems. In Proceedings of the 31st Annual ACM Symposium on Applied Computing (pp. 1812–1814). ACM.

[S386] Wettel, R., & Lanza, M. (2008, September). Visually localizing design problems with disharmony maps. In Proceedings of the 4th ACM symposium on Software visualization (pp. 155–164). ACM.

[S387] Wong, S., Cai, Y., & Dalton, M. (2010). Detecting design defects caused by design rule violations. Proc of 18th ESEC/FSE.

[S388] Yamashita, A., Abtahizadeh, S. A., Khomh, F., & Guéhéneuc, Y. G. (2017, May). Software evolution and quality data from controlled, multiple, industrial case studies. In Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on (pp. 507–510). IEEE.

[S389] Yamashita, A., & Moonen, L. (2012, September). Do code smells reflect important maintainability aspects?. In Software Maintenance (ICSM), 2012 28th IEEE International Conference on (pp. 306–315). IEEE.

[S390] Gupta, A., Suri, B., & Misra, S. (2017, July). A systematic literature review: code bad smells in Java source code. In International Conference on Computational Science and Its Applications (pp. 665–682). Springer, Cham.

[S391] Zazworka, N., & Ackermann, C. (2010, September). CodeVizard: a tool to aid the analysis of software evolution. In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (p. 63). ACM.

[S392] Zhang, M., Baddoo, N., Wernick, P., & Hall, T. (2011, March). Prioritising refactoring using code bad smells. In Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on (pp. 458–464). IEEE.

[S393] Zhang, X., Zhou, Y., & Zhu, C. (2017, November). An empirical study of the impact of bad designs on defect proneness. In Software Analysis, Testing and Evolution (SATE), 2017 International Conference on (pp. 1–9). IEEE.

[S394] Zhao, X., Xuan, X., & Li, S. (2015, November). An empirical study of Long Method and God Method in industrial projects. In Automated Software Engineering Workshop (ASEW), 2015 30th IEEE/ACM International Conference on (pp. 109–114). IEEE.

[S395] Frącz, W., & Dajda, J. (2017, July). Experimental validation of source code reviews on mobile devices. In International Conference on Computational Science and Its Applications(pp. 533–547). Springer, Cham.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

References

Alkharabsheh, K., Crespo, Y., Manso, E., Taboada, J. (2016a). Comparación de herramientas de Detección de Design Smells. In *Jornadas de Ingeniería del Software y Bases de Datos, JISBD* (pp. 159–172). Salamanca, Spain.

Alkharabsheh, K., Crespo, Y., Manso, E., Taboada, J. (2016b). Sobre el grado de acuerdo entre evaluadores en la detección de Design Smells. In *Jornadas de Ingeniería del Software y Bases de Datos, JISBD* (pp. 143–157). Salamanca, Spain.

Bavota, G., De Lucia, A., Di Penta, M., Oliveto, R., & Palomba, F. (2015). An experimental investigation on the innate relationship between quality and refactoring. *Journal of Systems and Software, 107*, 1–14.

Brown, W.J., Malveau, R., III, Mc Cormick, H.W., Mowbray, T.J. (1998). Antipatterns refactoring software, architectures and projects in crisis. John Wiley & Sons Inc.

Budgen, D., Bailey, J., Turner, M., Kitchenham, B., Brereton, P., Charters, S. (2008). Lessons from a cross-domain investigation of empirical practices. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE* (pp. 88–99). Italy.

Dyba, T., Dingsoyr, T., Hanssen, G.K. (2007). Applying systematic reviews to diverse study types: an experience report. In *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, ESEM* (pp. 225-234). Madrid, Spain.

Fernandes, E., Oliveira, J., Vale, G., Paiva, T., Figueiredo, E. (2016). A review-based comparative study of bad smell detection tools. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, EASE* (pp. 1-18). New York, NY, USA.

Fokaefs, M., Tsantalis, N. (2007). Jdeodorant: Identification and removal of feature envy bad smells. In *IEEE International Conference on Software Maintenance*, *ICSM* (pp. 519-520). Paris, France.

Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (1999). Refactoring: improving the design of existing code. Addison-Wesley Professional.

Ganter, B., Wille, R. (1999). Formal concept analysis*:* mathematical foundations. Springer, Berlin/Heidelberg.

Gupta, A., Suri, B., & Misra, S. (2017). *A systematic literature review: code bad smells in java source code.* In *International Conference on Computational Science and Its Applications*, *ICCSA* (pp. 665-682). Italy: Trieste.

Hassaine, S., Khomh, F., Guéhéneuc, Y., Hamel, S. (2010). IDS: an immune-inspired approach for the detection of software Design Smells. In *Proceedings of Quality of Information and Communications Technology, 7th International Conference on the Quality of Information and Communications Technology*, *QUATIC* (pp. 343-348). Porto, Portugal.

Jung, H. W., Kim, S. G., & Chung, C. S. (2004). Measuring software product quality: a survey of iso/iec 9126. *IEEE Software Journal, 21*(5), 88–92.

Kitchenham, B., Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. In *Tecnical. Report EBSE-2007-01, Keele University.*

Kitchenham, B., Mendes, E., & Travassos, G. H. (2006). A systematic review of cross-vs. within-company cost estimation studies. *In Proceedings of the 10th international conference on Evaluation and Assessment in Software Engineering* (pp. 81–90).

Laguna, M. A., & Crespo, Y. (2013). A systematic mapping study on software product line evolution: from legacy system reengineering to product line refactoring. *Journal of Science of Computer Programming, 78*(8), 1010–1034.

Lanza, M., Marinescu, R. (2006). Object-oriented metrics in practice—using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer.

Líška, P., Polášek, I. (2011). Design Smell Detection with similarity scoring and fingerprinting: preliminary study. In *Eastern European Regional Conference on the Engineering of Computer Based Systems, ECBS-EERC* (pp. 163–164). Bratislava.

Martin, R.C. (2003). Agile software development: principles, patterns, and practices. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Mens, T., & Tourwé, T. (2004). A survey of software refactoring. *IEEE Transactions on Software Engineering, 30*(2), 126–139.

Moha, N. (2007). Detection and correction of design defects in object-oriented designs. In Companion to the 22Nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion, OOPSLA (pp. 949–950). Montreal, Quebec, Canada.

Moha, N., Guéhéneuc, Y. G., Duchien, L., & Le Meur, A. F. (2010). Decor: a method for the specification and detection of code and Design Smells. *IEEE Transactions on Software Engineering, 36*(1), 20–36.

Novais, R. L., Torres, A., Mendes, T. S., Mendonça, M. G., & Zazworka, N. (2013). Software evolution visualization: a systematic mapping study. *Information & Software Technology, 55*(11), 1860–1883.

Pérez, J., Moha, N., Mens, T. (2011). A classification framework and survey for Design Smell management. In *Technical report*. 2011/01*,* GIRO Research Group*,* Departamento de Informática*,* Universidad de Valladolid.

Pérez-García, F.J. (2011). *Refactoring planning for Design Smell correction in object-oriented software*, Ph.D. thesis. University of Valladolid.

Polásek, I., Lìska, P., Kelemen, J., & Lang, J. (2012). On extended similarity scoring and bitvector algorithms for Design Smell detection. In *IEEE 16th International Conference on Intelligent Engineering Systems* (pp. 115–120). Lisbon, portugal: *INES*.

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

Rattan, D., Bhatia, R. K., & Singh, M. (2013). Software clone detection: a systematic review. *Information & Software Technology, 55*(7), 1165–1199.

Salehie, M., Li, S., & Tahvildari, L. (2006). A metric-based heuristic framework to detect object-oriented design flaws. In *14th IEEE International Conference on Program Comprehension* (pp. 159–168). Athens, Greece: *ICPC*.

Singh, S., Kaur, S. (2017). A systematic literature review: Refactoring for disclosing code smells in object-oriented software. Ain Shams Engineering Journal.

Suryanarayana, G., Samarthyam, G., & Sharma, T. (2014). Refactoring for software Design Smells: managing technical debt. Morgan Kaufmann.

Vasconcellos, F. J., Landre, G. B., Cunha, J. A. O., Oliveira, J. L., Ferreira, R. A., & Vincenzi, A. M. (2017). Approaches to strategic alignment of software process improvement: a systematic literature review. *Journal of Systems and Software, 123*, 45–63.

Wasylkowski, A., Zeller, A., & Lindig, C. (2007). Detecting object usage anomalies. In *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering* (pp. 35–44). Dubrovnik, Croatia: *ESECFSE*.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *In Proceedings of the 18th international conference on evaluation and assessment in software engineering* (p. 38). London, England: *EASE*.

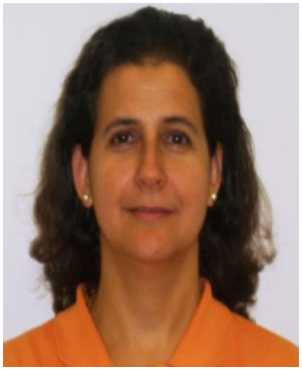
Wohlin, C., Runeson, P., Hst, M., Ohlsson, M., Regnell, B.,Wessln, A. (2012). Experimentation in software engineering. International Series. Springer Science & Business Media.

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

Zazworka, N., Shaw, M. A., Shull, F., & Seaman, C. (2011). Investigating the impact of design debt on software quality. In *In Proceedings of the 2nd Workshop on Managing Technical Debt* (pp. 17–23). Waikiki, Honolulu, HI, USA: MTD.

Zhang, M., Hall, T., & Baddoo, N. (2011). Code bad smells: a review of current knowledge. *Journal of Software Maintenance, 23*(3), 179–202.

Khalid Alkharabsheh is a PhD student at the Department of Electronics and Computer Science of the Universidade de Santiago de Compostela (USC), under the supervision of Prof. Jose Angel Taboada (Universidade de Santiago de Compostela) and Prof. Yania Crespo (Universidade de Valladolid). He is a member of the COGRADE research group and trainee of the Centro Singular de Investigación en Tecnoloxías da Información (CITIUS). He received his bachelor's (BSc) and master's degree (MSc) in computer science from Yarmouk University and Al-Balqa' Applied University respectively, Jordan, in 2005. His research interests include the software quality, software validation and verification, and software evolution.



Yania Crespo is an Associate Professor at the Computer Science Department of the Universidad de Valladolid (UVA). She is a founding member of the GIRO (Grupo de Investigación en Reutilización y Orientación a Objeto) research group of the UVA. She obtained a master's degree (1995) and a PhD (2000) in computer science at the University of Havana and Universidad de Valladolid, respectively. Her current research interests are related to software quality, software refactoring, software product line, software evolution, and object-oriented programming.

M. Esperanza Manso is a Professor at the Computer Science Department of the Universidad de Valladolid. She is a founding member of the GIRO (Grupo de Investigación en Reutilización y Orientación a Objeto) research group of the UVA. Her current research interests are related to software engineering, programming languages, and information science. Their current project is Threshold Prediction.

Jose A. Taboada is an Associate Professor at the Department of Electronics and Computer Science of the Universidade de Santiago de Compostela (USC). Besides, he is a founding partner of a USC's spin-off company, member of the COGRADE (Computer Graphics and Data Engineering) research group of the USC, and he is member of the research staff of the Centro Singular de Investigación en Tecnoloxías da Información (CITIUS). He obtained a PhD (1996) in physics at the Universidade de Santiago de Compostela. His current research interests are in software quality, smart building, and computing languages and systems.

Affiliations

Khalid Alkharabsheh[1] • **Yania Crespo**[2] • **Esperanza Manso**[2] • **José A. Taboada**[1]

Yania Crespo
yania@infor.uva.es

Esperanza Manso
manso@infor.uva.es

José A. Taboada
joseangel.taboada@usc.es

[1] Universidade de Santiago de Compostela, Santiago de Compostela, Spain

[2] Universidade de Valladolid, Valladolid, Spain