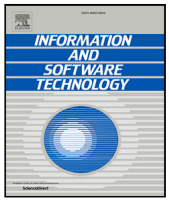
Information and Software Technology 144 (2022) 106783

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



How far are we from reproducible research on code smell detection? A systematic literature review

Tomasz Lewowski *, Lech Madeyski

Wroclaw University of Science and Technology, Department of Applied Informatics, Poland

ARTICLE INFO

Keywords:
Software engineering
Code smells
Reproducibility
Reproducible research

ABSTRACT

Context: Code smells are symptoms of wrong design decisions or coding shortcuts that may increase defect rate and decrease maintainability. Research on code smells is accelerating, focusing on code smell detection and using code smells as defect predictors. Recent research shows that even between software developers, agreement on what constitutes a code smell is low, but several publications claim the high performance of detection algorithms—which seems counterintuitive, considering that algorithms should be taught on data labeled by developers.

Objective: This paper aims to investigate the possible reasons for the inconsistencies between studies in the performance of applied machine learning algorithms compared to developers. It focuses on the reproducibility of existing studies.

Methods: A systematic literature review was performed among conference and journal articles published between 1999 and 2020 to assess the state of reproducibility of the research performed in those papers. A quasi-gold standard procedure was used to validate the search. Modeling process descriptions, reproduction scripts, data sets, and techniques used for their creation were analyzed.

Results: We obtained data from 46 publications. 22 of them contained a detailed description of the modeling process, 17 included any reproduction data (data set, results, or scripts) and 15 used existing data sets. In most of the publications, analyzed projects were hand-picked by the researchers.

Conclusion: Most studies do not include any form of an online reproduction package, although this has started to change recently—8% of analyzed studies published before 2018 included a full reproduction package, compared to 22% in years 2018–2019. Ones that do include a package usually use a research group website or even a personal one. Dedicated archives are still rarely used for data packages. We recommend that researchers include complete reproduction packages for their studies and use well-established research data archives instead of their own websites.

1. Introduction

Code smells are usually understood as patterns in the source code which make it less comprehensible and more error-prone (sometimes they are called antipatterns). According to several recent literature reviews, for example, the one by Santos et al. [36] or the one by Sharma and Spinellis [37], the amount of research related to code smells grows steadily. However, a review by Azeem et al. [2] reports that the agreement between studies is very low. Our internal research for an industrial partner, code quest (http://codequest.com), that develops the codebeat (http://codebeat.co) automated code review platform integrated with GitHub, has brought us to the same conclusions. While the sheer number of papers on the subject grows, the results that they achieve differ vastly, even for papers that use the same algorithms.

We believe that there are two possible causes for this divergence: either there is a difference in understanding of the subject matter between researchers (which would mean that the "code smell" label represents multiple concepts—which would fall in line with findings from [15]) or published research has substantial problems with external validity, i.e., the conclusions and methods are either not valid outside of the narrow problem definition tackled in each of the papers or not described in sufficient detail (which would be a problem similar to the one described in [33] for the SZZ algorithm). It is also possible that both interpretations contribute to the divergence.

This study was conducted to assess whether the latter problem – the external validity – may be a substantial problem in current research on machine learning methods of code smell detection.

* Corresponding author.
E-mail addresses: tomasz.lewowski@pwr.edu.pl (T. Lewowski), lech.madeyski@pwr.edu.pl (L. Madeyski).

https://doi.org/10.1016/j.infsof.2021.106783
Received 27 February 2021; Received in revised form 6 November 2021; Accepted 8 November 2021
Available online 25 November 2021

Problems with external validity may be discovered in reproduction studies with slightly modified experimental setups. However, to create a *slightly* modified experimental setup, one needs to exactly understand the details of the original setup. In software engineering in general, and in code smell detection in particular, it is not possible to summarize the environment by only giving some of its core parameters—simply because it is not well-known which parameters matter. As a result, exact experimental setups in software engineering research may include hundreds of projects with thousands of files, which itself poses a substantial challenge for reproducibility. Another easily forgotten issue is related to the used software—while in many areas stable and supported programs are the core research tools, in software engineering, we often rely on open source tools with best-effort support or external services that only offer access to data via API calls. It is not uncommon for such tools to become inaccessible or unsupported, or simply to break backward compatibility, especially years after the original research is published. The same case applies to API providers—maintaining data and compatibility is associated with certain costs, and every business needs to decide whether they bring in enough benefit to bear them.

In this paper, we focus on *reproducibility*. A closely related term used by some researchers is *replicability*. We follow the distinction set in the paper by Madeyski and Kitchenham [21] and National Academy of Sciences, Engineering and Medicine guidelines [27]—*reproducibility* means obtaining consistent computational results using the same input data, computational steps, methods, code, and conditions of analysis, while *replicability* means obtaining consistent results across studies aimed at answering the same scientific question, each of which has obtained its own data. We claim that providing a sufficient level of reproducibility will be the key to understanding why results achieved by various researchers vary. This paper is a systematic review based on data gathered during the aforementioned research project, extended with information about study reproducibility. The main contribution of this paper is the assessment of the current state of reproducibility in research related to machine learning in code smell detection.[1] We did focus solely on methods that use machine learning, because the meaning of "code smell" concept should be extracted from developers' knowledge, and this is what machine learning methods do.

The rest of the paper is structured as follows: in Section 2 we discuss other literature reviews that consider machine learning in the code smell detection area and publications that emphasize reproducibility, in Section 3 we describe our research goals and research questions. Next, in Section 4 we explain the technical details of our research setup, methods and assumptions. Then we present results in Section 5, discuss the study results and setup in Section 6 and conclude the paper in Section 7.

2. Related work

Several literature reviews on code smell detection using machine learning techniques have been recently published. Most of them – works by Al-Shaaby et al. [1], Caram et al. [4], Sharma and Spinellis [37], Singh and Kaur [39], Rasool and Arshad [32] – focus on the studied code smells, machine learning methods, predictors, data sets and tools. One, by Santos et al. [36], focuses on the general themes of studies investigating code smells (such as whether they focus on detection, correlation with issues or human aspects of code smells). An important review by Azeem et al. [2] reports low agreement between the conclusions of various researchers and the paper by Caram et al. [4] notes the incomparability of the studies due to the usage of different data sets, but none of the ones we found focus on the reproducibility of applied research methods. In particular, no literature review on code smells that we know of is dedicated to analysis of the reproducibility of study results and their relevance to the industry.

According to Rozier and Rozier [35], reproducibility is a substantial problem in all areas of science nowadays. This also includes software engineering, where reproduction is challenging and, according to Shepperd et al. [38], attempted relatively infrequently. Research results can be non-reproducible for a plethora of reasons, starting from unique population selection, through unclear methods and imprecise results up to data fabrication, as reported by Carlisle [5]. While the scientific community puts much effort into addressing those issues, the consequences of misconduct can be critical. Sometimes, as reported by Mousavi and Abdollahi [26], such misconduct can even endanger people's health and life.

Papers exhibiting signs of scientific misconduct are a minority of published works, even though the analysis performed by Hensselman et al. [13] reports that their amount is rising . Unintentional lack of reproducibility seems to be much more common and, at the same time, much harder to address. For example, research done on proprietary data may be perfectly valid and lead to relevant results, but the details of the data set acquisition will likely be covered by a nondisclosure agreement, thus impacting reproducibility. Of course, researchers should strive to reveal as many details as possible (e.g., number of source files, classes, methods, average function complexity, etc.). Nevertheless, reproducibility of the research itself will usually be severely impacted. If the conclusions are not externally valid, it is challenging to distinguish such a publication from a malicious one.

This paper was inspired by several others: two papers by Hozano et al. [14,15], claiming that the agreement regarding code smell assessment is low between developers, a paper by Fontana et al. [9], where multiple machine learning techniques achieve F1-scores above 0.9 and the review by Azeem et al. [2] which reports substantial differences between results from various papers.

A paper addressing the same problem – reproducibility of research methods – in a related domain (usage of SZZ algorithm) was recently published by Rodriguez et al. [33], where the conclusion was that less than 15% of the relevant publications provided a reproduction package[2] and 40% did not provide a sufficient method description.

A paper by Madeyski and Kitchenham [21] points out study reproduction as critical for confirming or rejecting research in the area of software engineering. The state of reproducibility in the area of code smell detection using machine learning techniques remains largely undocumented up to now.

3. Goals

The goal of this paper is to assess the current state of reproducibility in the area of code smell detection using machine learning techniques and, if necessary, provide guidance on how to improve it. We believe that the core aspects that are required to reproduce the study are: the full process (used algorithms, configuration, preprocessing steps) and input data (or its acquisition process). Therefore, we have formulated the following research questions:

RQ1 Is the process of creating code smell machine learning models reproducible in the recent scientific studies on code smell detection using AI/ML methods?

RQ2 Is the process of code smell data set acquisition reproducible in the recent scientific studies on code smell detection using AI/ML methods?

[1] Papers slightly outside the formal boundary of "machine learning" (e.g. in the area of artificial intelligence or soft computing) were not rejected if they matched the subject matter.

[2] Some authors use the term *replication package* instead.

Since the questions may seem vague, we decided to split them into several subquestions that address certain aspects of the original question.

Reproducibility of the research process, which is the concern that **RQ1** deals with, is generally provided by a detailed description of all the steps in the procedure. This description is usually one of the main topics of a research paper. However, complex research processes, such as ones employed in the software engineering area, often heavily depend on the details that cannot be given in the paper, e.g., source code of processing programs. In such case a common solution is to include an appendix to the paper, which substantially improves reproducibility. Therefore we decided to split **RQ1** into following two subquestions:

RQ1.1 Do the authors of recent scientific studies on code smell detection using AI/ML methods describe the modeling process with enough details to reproduce their studies?

RQ1.2 Do the authors of recent scientific studies on code smell detection using AI/ML methods publish reproduction packages for their studies?

This review deals with machine learning models, and those can only be as good as the data that was used to train it. Therefore, considering the lack of result reproducibility among studies and the observations presented by Caram et al. [4], we decided to take a closer look at the training data. Training data acquisition for the code smell detection problem can be divided into three phases:

1. selecting source code repositories (sometimes referred to as *projects*),
2. selecting samples inside the repositories (usually files, classes or methods),
3. classifying samples for training and testing data sets (whether a sample represents a code smell, sometimes also the severity of this smell).

Those three phases are modeled by the following subquestions:

RQ2.1 How do the authors of recent scientific studies on code smell detection using AI/ML methods select projects for their data sets?

RQ2.2 How are samples from projects selected for classification in recent scientific studies on code smell detection using AI/ML methods?

RQ2.3 How are samples assessed for occurrence of code smells in recent scientific studies on code smell detection using AI/ML methods?

It is important to note that those three subquestions may have a single answer when the researcher uses a predefined data set, such as the one provided by Madeyski and Lewowski [22] or by Palomba et al. [30]. However, many researchers decide to use their own data sets, often for good reasons—for example, if their research requires to focus on a specific problem domain, programming language, or paradigm. In such case, each of the steps is important and should be described in detail (although data from a predefined set may be used in some of the phases, e.g., to select projects).

4. Method

This literature review builds upon an earlier systematic literature review [20] conducted with the same search string, acceptance criteria, and filtering procedures. However, the results of the other review were not conclusive. This made us look for the reason for the divergences between various studies, stated as research goals in Section 3. We decided to perform an additional data acquisition phase, this time focused on the processes rather than on the results, for all previously analyzed publications. In this section, we describe the whole data acquisition procedure executed in the original review, as it brought us to the set of publications that were analyzed in detail. We omit the original research questions as irrelevant for this paper, and instead we only focus on the ones discussed here.

4.1. Protocol development

The literature review has started with protocol development. During this phase, a protocol was created that defined the procedures to be used for the search process, study selection, data extraction, and data analysis. The protocol was drafted by one of the authors and double-checked by the other.

The following subsections are based on the protocol. If at any step our actual process diverges from the one defined in the protocol, we report the actual process.

4.1.1. Search process

We have selected Scopus as our publication source, due to its wide coverage of academic papers. Our search was originally made as part of an industrial project, so we were more interested in reproducible findings than in a high recall rate. Nevertheless, we decided to verify the search strategy using a quasi-gold standard as proposed by Zhang et al. [44] and later by Kitchenham et al. [18] by manually inspecting a specified set of relevant journals and proceedings from conferences published in 2017. Detailed results are reported later in Section 4.2.

Our search process can be summarized as follows:

1. identify initial set of publications using Scopus automated search,
2. evaluate the papers for inclusions and exclusions.

4.1.2. Initial study selection process

Major terms were derived from our initial research goals, i.e., finding papers that describe approaches to machine learning in the area of code smell detection. Then we identified alternative spellings and synonyms and constructed Scopus search strings. We decided to cover the range from 1998, when the concept of code smell was widely published by Fowler [11].

Our initial search string was:

```
TITLE-ABS-KEY (
  ( "code smell" OR "bad smell"
    OR antipattern OR anti-pattern
    OR "anti pattern" )
  AND ( "machine learning" OR predict* )
) AND PUBYEAR > 1998
```

The first search was performed on February 21, 2018. 88 papers were returned from Scopus and then archived in BibTeX format.

After analysis of this preliminary data set, several corrections to the search string were introduced. The final search string was:

```
TITLE-ABS-KEY (
  ( "code smell" OR "bad smell"
    OR antipattern OR anti-pattern
    OR "anti pattern" )
  AND ( "machine learning" OR predict*
    OR {detect} OR {detection}
    OR heuristic*
  ) AND software
) AND PUBYEAR > 1998
```

This search string was first used to retrieve a publication list on March 21, 2018, and it returned a list of 424 papers, which were then archived for further analysis in BibTeX. Due to the fact that the project took a long time, the search was rerun on June 05, 2020, yielding a total of 607 papers, which were then archived the same way as the original set.

The biggest contributor to this increase is the new term “detect”, which is fairly often used without the term “predict”.

Those terms are also applicable to the current study, as it operates on the same population of publications (ones that tackle detecting code smells using machine learning techniques), but examines different properties of the papers.

Table 1
Results of the study selection process.

	Phase 1	Phase 2
# of publications	607	169
# of relevant publications	169	45
Precision	27.8%	26.6%
Relevant publications in QGS only	0	0
Relevant publications in set and QGS	5 ([31] [6], [28] [10][29])	0
Recall	100%	N/A

The initial plan contained also backward and forward snowballing as described by Wohlin [42], but since the number of found primary studies exceeded the numbers of studies found in other systematic reviews in the area of code smell detection using machine learning techniques (e.g., [1,4,32]), and reviewed papers already included a number of various methods and techniques, we decided to stick to the identified studies.

4.1.3. Inclusion and exclusion criteria

To simplify automated processing, the criteria were formulated as a checklist. A paper would be further processed (i.e., relevant data would be extracted from it) only if all of the following statements would be true regarding it:

1. The entry is a single journal paper, chapter of a book or conference proceedings paper which requires peer review (i.e., it is not an editorial, abstract, technical report etc.).
2. The paper is written in English.
3. The paper was published in 1999 or later.
4. Title or abstract of the paper indicates that it is related to software engineering.
5. Title or abstract of the paper indicates that at least one code smell/anti-pattern plays an important part of the study.
6. Title or abstract of the paper indicates that it might use machine learning techniques.
7. Abstract or full text of the paper indicates that it focuses on code smells/anti-patterns in programming languages.
8. Abstract or full text of the paper indicates that it focuses on code smells/anti-patterns detection using source code.
9. The paper does not focus on techniques for resolving code smells/anti-patterns.
10. The paper does not focus on using code smells/anti-patterns as predictors of other code or project traits.
11. The paper focuses on detection/prediction of code smells/anti-patterns.
12. If the paper is a chapter of a book or conference proceedings publication, its authors have not published a study under the same title in a journal (we want to include the paper once and it may be expected that the journal version includes more details).
13. Full text of the paper is available.

These criteria were applied in two phases: first, on the basis of titles and abstracts and second, on the basis of full texts. The second phase was run concurrently with study quality assessment and data gathering, but nevertheless full result table is presented in the appendix. The results of those phases together with the results of the search quality check are presented in Table 1.

In the case of uncertainty, we took an inclusive approach, i.e., we continued to analyze the paper. For example, if a paper was found that focused equally on both detection and resolution or detection and prediction based on the smells, it would be included in the study. We did not encounter such a paper during the review—papers that were focused on resolution or prediction based on smells and only performed detection as a necessary intermediate phase were rejected, because in most cases they used existing tools to mark code fragments as smells, without assessing the correctness of this step.

We were aware of one more publication relevant to the study which was not present in the search results—[SLR1]. Therefore, we added the aforementioned to the initial publication set manually and it has gone through the same process as all the other papers qualified for the study. Hence, finally 46 papers were included. It is worth mentioning that the aforementioned paper was not found, because it lacked "software" keyword in title and abstract. However, we decided to include this keyword in the search string, because not doing so would introduce a substantial amount of noise (e.g., the increased number of papers about bacteria and odor detectors). This paper would also have been found if snowballing procedure was done after its publication (we started the study before its publication date and later updated the data set with the results of the same query, as described in Section 4.1.2).

4.2. Quality check

Two of the key criteria given by Dieste et al. [7] and Zhang et al. [44] for verifying the search are *precision* and *recall* (also known as sensitivity). They can be calculated as follows:

$$Precision = \frac{R_{found}}{N_{total}} \tag{1}$$

$$Recall = \frac{R_{found}}{R_{total}} \tag{2}$$

where R_{found} is the number of relevant studies found, R_{total} is the total number of relevant studies and N_{total} is the total number of studies found. In practice, R_{total} is not known. To estimate recall in real-life setups, a "quasi-gold standard" (QGS) procedure for assessing search performance was proposed by Zhang et al. [44] and the original results were promising.

The procedure consists of selecting several topic-specific venues (in our case journals and conferences dedicated to software engineering) and performing an exhaustive search on this sample (treating it as population). Since the venues should be topic-specific, it is then assumed that if a sufficient portion of the relevant papers from those venues are discovered by the initial search (Zhang et al. [44] propose this threshold to be between 70 and 80%), then the search strings are built well enough. Otherwise, another iteration is required.

In the case of our review, the validation process (presented with details in Fig. 1) was as follows:

1. identify topic-specific venues,
2. perform exhaustive search on all publications from those venues to establish a quasi-gold standard,
3. if the initial search evaluation does not pass the threshold (we decided to set it to 75% inspired by Zhang et al. [44]), revise the search string.

It diverges slightly from the original proposal by Zhang et al. [44] in that the initial search string is not derived from the QGS results but created using the authors' domain knowledge. There are two reasons for this:

1. independence of initial search string formulation and selection of venues enhances reliability of the procedure by decoupling search string and validation data set,
2. lack of the aforementioned dependency enables parallel work on quasi-gold standard data set and automated search data set, thus it allowed us to deliver business value earlier and stakeholders to review their expectations.

The results in the form of quasi-sensitivity and quasi-precision can be calculated using Eqs. (1) and (2) and publications from venues selected for QGS as the whole population.

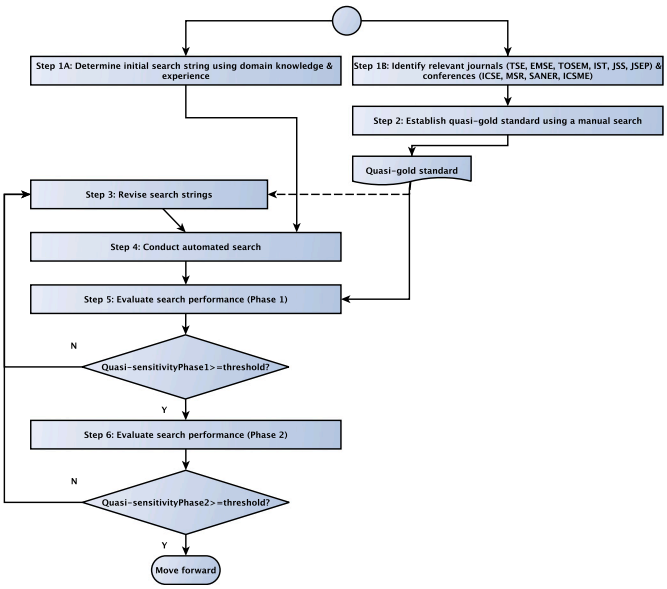


Fig. 1. Workflow of the systematic search process (inspired by [44]).

4.2.1. Selected venues

Establishing a QGS requires the identification of relevant journals and conferences. We only cover the main full paper research tracks of the conferences, ignoring collocated conferences or workshops (with an exception for SEIP track on ICSE). Venues are selected using the authors' expertise and domain knowledge.

The following venues were selected:

- Journal - IEEE Transactions on Software Engineering (TSE)
- Journal - Empirical Software Engineering (EMSE)
- Journal - ACM Transactions on Software Engineering Methodology (TOSEM)
- Journal - Information and Software Technology (IST)
- Journal - Journal of Systems and Software (JSS)
- Journal - Journal of Software: Evolution and Process (JSEP)
- Conference - International Conference on Software Engineering (ICSE + SEIP)
- Conference - Mining Software Repositories (MSR)
- Conference - IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)
- Conference - International Conference on Software Maintenance and Evolution (ICSME)

4.2.2. Quality check results

Publications from those venues were extracted from Scopus on April 23, 2018, using the search strings presented in Table 2. To keep the table clear, we omit `AND PUBYEAR = 2017`, present at the end of every search string. Number of extracted publications and established QGS publications are presented in Table 3

It is worth mentioning that limiting PUBYEAR in Scopus to 2017 does not exclude papers accepted in this year but not yet assigned to a specific issue (article in press). Therefore, more papers may have been analyzed (papers from 2017, but possibly also some from 2018), but this does not affect the results of the procedure.

To verify why our QGS procedure performed so bad, we have investigated at which venues were the accepted papers published. Their DOIs, venue names and CORE rankings are presented in Table 4. For conferences we used CORE2014. We used this specific version as it is the last release of the list before the submission date for papers published in 2017, so the authors might have been using this list to choose a venue to publish their research. We also checked CORE2017

Table 2
Search strings used to extract publications from Scopus to establish quasi-gold standard.

Venue	Search string
TOSEM	SRCTITLE ("ACM Transactions on Software Engineering and Methodology")
TSE	SRCTITLE ("ieee transactions on software engineering")
EMSE	SRCTITLE("Empirical Software Engineering") AND (LIMIT-TO (EXACTSRCTITLE , "Empirical Software Engineering"))
IST	SRCTITLE ("Information and Software Technology")
JSS	SRCTITLE ("Journal of Systems and Software")
JSEP	SRCTITLE ("Journal of Software Evolution and Process")
ICSE	SRCTITLE ("International Conference on Software Engineering") AND (LIMIT-TO (EXACTSRCTITLE , "Proceedings 2017 IEEE ACM 39th International Conference On Software Engineering ICSE 2017") OR LIMIT-TO (EXACTSRCTITLE , "Proceedings International Conference On Software Engineering") OR LIMIT-TO (EXACTSRCTITLE , "Proceedings 2017 IEEE ACM 39th International Conference On Software Engineering Software Engineering In Practice Track ICSE Seip 2017"))
MSR	SRCTITLE ("Mining Software Repositories")
SANER	SRCTITLE ("IEEE International Conference on Software Analysis, Evolution and Reengineering")
ICSME	SRCTITLE ("International Conference on Software Maintenance and Evolution")

Table 3
Results of QGS search and selection.

Title	Total	# in Phase 1	# in Phase 2
TSE	108	2	0
EMSE	131	0	0
TOSEM	12	0	0
IST	111	3	0
JSS	225	4	0
JSEP	68	0	0
ICSE	152	1	0
MSR	67	1	0
SANER	84	2	0
ICSME	175	5	0
Total	1133	18	0

and the results were the same. Only one of those papers was published in a CORE A venue. Since there are 6 conferences on the CORE2014 list with "Software Engineering" in the title (not to mention others with keywords like "Software Assessment", "Software Maintenance" or "Software Analysis") with ranking CORE A or higher, we were not able to verify all of them, instead focusing on the ones we believed were the most promising. As for the journals, *Knowledge-Based Systems* is classified by Scopus to categories "Business, Management and Accounting" and "Management Information Systems", while *International Journal of Electrical and Computer Engineering* is classified as "Computer Science: General Computer Science" and "Engineering: Electrical and Electronic Engineering", neither of which is strictly related to software engineering.

The original QGS procedure by Zhang et al. [44] assumed having prior knowledge of some publications that would be included in the review, which would decrease reliability of the procedure—since the author knows which venues will be used to verify the search, it is easy to tune the search string for those venues, even if this is done subconsciously. To address this issue, we selected several well-established software engineering venues without prior knowledge of accepted publications. However, this approach also has its limitations—in our case it turned out that no accepted paper was published in the

Table 4
Publications published in 2017 accepted for analysis.

DOI	Venue name	CORE
10.1109/ASE.2017.8115667	ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering	A
10.1016/j.knosys.2017.04.014	Knowledge-Based Systems	–
10.1109/MOBILESoft.2017.29	MOBILESoft 2017 - Proceedings of the 2017 IEEE/ACM 4th Int. Conference on Mobile Software Engineering and Systems	None
10.1109/MLDS.2017.8	2017 International Conference on Machine Learning and Data Science (MLDS)	None
10.11591/ijece.v7i6.pp3613-3621	International Journal of Electrical and Computer Engineering	–
10.5220/0063388 4740482	ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems	C

aforementioned venues within the selected time frame in spite of the fact that we analyzed 1133 papers from a wide range of sources, see Table 3.

Since the QGS procedure did not yield satisfying results, we did an informal post-hoc check with some of the other systematic reviews about machine learning in code smell detection and confirmed that our study includes a majority of papers included in the other ones.

4.3. Data extraction process

During a single pass, we acquired both data related to assessing study quality and to answer our research questions. To assess study quality, we applied a quality checklist inspired by Dybå and Dingsøyr [8]. As noted by Kitchenham et al. [18], this checklist can be used across multiple study types using machine learning, e.g. effort estimation [41] or software fault prediction [24].

Following the practice set by Wen et al. [41] and Malhotra [24], each question has three possible answers: "Yes" (scored 1), "Partly" (scored 0.5) and "No" (scored 0). The final score for a publication is the sum of scores for all items on the checklist. Each study scores between 0 and 7, but this score does not affect its inclusion to the study. Below is a complete list of quality check questions.

Q1 Are the aims of the research clearly defined?

Yes goals of the paper are explicitly defined and presented

Partly goals of the paper are briefly mentioned (perhaps as part of introduction)

No the paper goes straight to the proposed concept, without discussing its goals

Q2 Are the performance measures used to assess the models clearly defined?

Yes paper either explicitly refers to papers defining the performance measures or defines them itself

Partly paper uses well-known measures, like precision and recall, but without defining or referencing them

No paper uses non-standard performance measures without defining them or does not perform performance evaluation

Q3 Are the performance measures used to assess the models considered credible?

Yes performance measures use all quadrants of the confusion matrix, e.g. MCC

Partly performance measures use some quadrants of the confusion matrix, e.g. precision and recall use together three out of four quadrants of confusion matrix or use entirely different mechanisms (e.g. correlation with defects)

No no performance measurement done

Q4 Are the limitations or threats to validity of the study specified?

Yes a detailed analysis of threats is done in the paper

Partly only brief threat analysis is performed

No no analysis of threats is given in the paper

Q5 Is the proposed method or methods compared with other methods and/or baselines?

Yes well-defined baseline is used and uses same performance measures and data sets as proposed methods

Partly baseline is used, but it is not entirely representative (e.g. results reported on different data sets by other researchers are used as baseline)

No no baseline is given

Q6 Are the findings of the study clearly stated and supported by reported results?

Yes findings are presented and fully and unambiguously supported by reported results

Partly findings are presented and reported results can be reasonably interpreted as supporting the findings

No findings are either not presented or are contradictory to reported results (or at least not supported by them)

Q7 Does the study provide convincing arguments about additional value given to academia or industry community?

Yes earlier research is described and new contributions are clearly stated

Partly new contributions are stated without or with limited reference to earlier research

No new contributions are not stated or are not new

A low study quality score did not cause excluding the study from the review, but we gathered the values and reported the statistics in Table 5 and in the attached data set, while the total counts of papers with each score, split by quality question is shown in Fig. 2. The lowest number of papers got a "Yes" assessment for Q3 (performance metrics). Most papers included metrics like precision, recall, and F-score. Those metrics include data from three out of four quadrants from the confusion matrix which is problematic as described by Yao and Shepperd [43]. The highest number of "No" assessments is for Q4 (limitations) and Q5 (baseline inclusion).

For this set of selected publications, we decided to follow the same procedure as in the original review, only with a modified set of Publication Questions (PQs)[3]:

PQ1 Was the modeling process explained in publication text?

[3] We define **Publication Question** as an item in a data collection sheet, which is answered separately for each publication and which are then merged to form an answer for the Research Question.

Table 5
Statistics for quality assessment scores.

Statistic name	Statistic value
Max achievable score	7
Max achieved score	7
Average score	4.5
Median score	5
Min score	0.5
Total number of publications	46

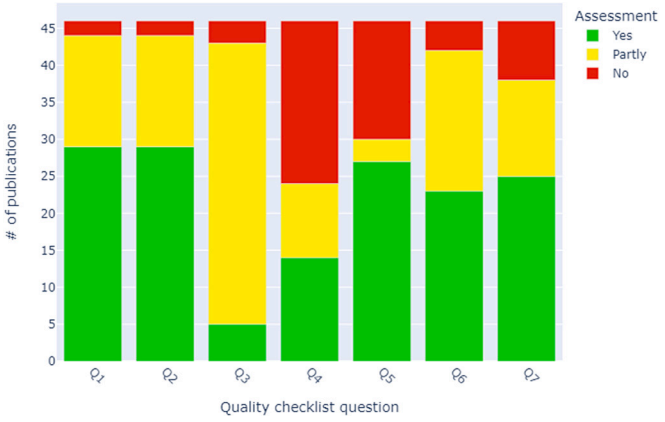


Fig. 2. Number of publications with each score for each question.

Detailed all relevant details seem to be present, i.e., the paper contains both description of the method and parameters used during evaluation.

Moderate many details are present, but some seem to be lacking, i.e., the description did contain the main elements of the method, but lacked some details (e.g., parameters, initial values).

Brief only a general description is given, i.e., the description was clearly incomplete (e.g., contained only the name of the algorithm).

PQ2 Was a reproduction package published?

Yes package was published and contained both used data and scripts.

Scripts only package was published with only scripts or programs used in the study.

Data only package was published with only data used during the study or study results.

No package was not published.

PQ3 How were projects selected? (free text answer)

PQ4 How were samples from projects for classification selected? (free text answer)

PQ5 How were code smells assessed? (free text answer)

In Section 5, we present answers to RQs derived from PQ3, PQ4, and PQ5 as categorized into several groups. The classifications in the answers to these research questions are created post hoc, i.e., we first gathered data as open-ended responses and created those classifications only after data analysis. Details of the classifications are presented together with answers to respective research questions.

The data from 46 publications was initially gathered by one of the authors, while the other did a two-phase cross-check. In the first phase, 11 of 46 papers (≈25% of the whole data set) were analyzed. In the case of 5 papers there was complete agreement from the beginning, in the case of 2 papers there were minor disagreements (such as the level of detail with which answers were gathered). In the case of 4 papers, there was a strong disagreement. All four strong disagreement cases were related to PQ2, one of them also had disagreement regarding the source of the data set. Two of the disagreement cases were related to a link being unavailable for one of the authors, while available for the other. In one of the cases, the reproduction link was valid in PDF but not valid on the publication HTML page (`wcre` from PDF translated to `were` in HTML in a paper by Maiga et al. [SLR2]). In the other case (in the paper by [SLR3]), the link was valid on the publication HTML page, but not in the PDF — this being caused by the tilde character in HTML being represented as UTF "tilde (U+007E)", while the tilde in PDF was represented by UTF "tilde operator (U+223C)" (there is also a few other very similar characters in UTF). Visually those characters vary a little in width and decoration, but it can only be seen by direct comparison, and they are unlikely to be identified at a first glance. Browsers are also able to display most Unicode characters in the address bar nowadays, so there is virtually no option to distinguish between those two, if one is not aware of the existence of such a problem—but only one of them will be properly resolved by a HTTP server, thus leading to researchers being sure that a link is broken despite it being correct. The remaining two disagreement cases – [SLR4,SLR5] – were caused by a different technique of referring to the reproduction data set, which went unspotted by one of the authors on the first read (most papers add a hyperlink, while those two cited the reproduction package).

Table 6
Inter-rater agreement (Cohen's Kappa) for data acquisition.

PQ1	PQ2	PQ3	PQ4	PQ5
0.89	0.85	0.89	0.97	1.00

Due to substantial disagreement on PQ2, the remaining 35 publications were once again verified and one additional change was made to the data set (also caused by the invalid link in some of the sources in [SLR6]). Finally, we were not able to reach two of the links: http://essere.disco.unimib.it/wiki/research/mlcsd referred to by [SLR7] and http://www.rcost.unisannio.it/mdipenta/papers/ase2013. The second one is referred to by several publications (most notably [SLR8]), so we concluded that this is the correct address, but the links are no longer functional. This is unfortunate, since this data set is used in several other studies as well.

After this additional verification for PQ2, the second phase of the cross-check was carried out by the other author for the remaining 35 publications. This additional cross-check yielded seven disagreements on seven publications that affected the final results—three of those were related to PQ1 (*Brief* vs *Moderate* explanation of the modeling process), two were related to PQ3 (*Manual - criteria given* vs *Automated*), one was related to PQ2 (*None* vs *Scripts*) and one was related to PQ4 (*Algorithm run* vs *Unknown*). To calculate Cohen's kappa metric for inter-rater agreement for PQ3, PQ4 and PQ5, we had to first preprocess the free-text answers given by the authors into post-hoc classes that are presented in Section 5. The artifacts of the preprocessing are included in the reproduction package. This metric was calculated after the second cross-check and does not include the disagreements encountered during the first one. The final inter-rater agreement in form of Cohen's kappa for each of the PQs is reported in Table 6. In each case, the agreement was 0.85 or higher.

All disagreements were resolved by discussion. As a result, each entry in the data set was verified by both authors. The full data collection sheet is available as part of the attached data package described briefly in Section 4.4 and in detail in the data sheet distributed together with the package.

Combined agreement rate for all PQs from both phases was 72%.

Nowadays, there are archive providers such as Zenodo, Figshare or Dryad that allow researchers to submit even large data sets and

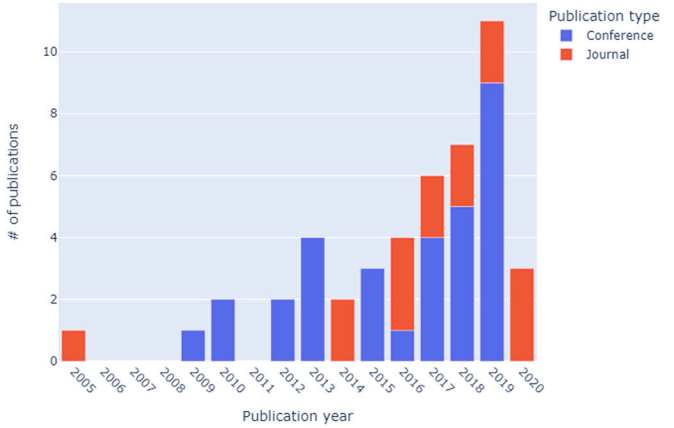


Fig. 3. Number of publications from each year included in the study divided by its type.

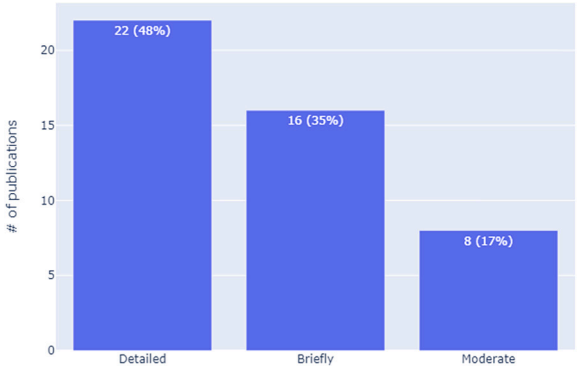


Fig. 4. Level of detail included in the process of modeling.

guarantee their accessibility for a prolonged period of time. However, only one publication – [SLR9] – used these services as the target location for the reproduction package—others used websites of research groups or even single researchers. We believe this to be a suboptimal solution, as those ad hoc locations are usually not thought of as a persistent repository, not to mention the additional maintenance effort that needs to be put into the project. Two of those packages, referred from [SLR8] and [SLR7] are already not accessible, some data from [SLR10] and package provided by [SLR11] are also no longer accessible. We encountered also problems when accessing the reproduction package from [SLR12], although this one was caused by server downtime (which is usually lower for dedicated services), since in the end we were able to access the package.

4.4. Data package

Data gathered during this study is published on Zenodo: https://doi.org/10.5281/zenodo.5647093. The data set partly overlaps with the one that was published for [20] (quasi-gold standard, initial publication filtering and part of the quality check), but we decided to include it separately here as since merging data from two separate data sets is inconvenient for the reader.

The package contains:

- a file with results of initial publication screening,
- a file with data collection results,
- a datasheet for both those data sets based on the template provided by Gebru et al. [12],
- files with preprocessed answers for Cohen's kappa calculation,
- scripts used to generate charts.

Details of the data, including structure, maintenance, acquisition methods, and other properties can be found in the datasheet present in the reproduction package.

5. Results

We obtained a total of 46 publications, listed with DOIs in the Systematic Literature Review References at the end of the paper. Fifteen of those were journal papers, while the other 31 were conference proceedings. Their distributions by publication date and type are presented in Fig. 3. The number of relevant publications is nondecreasing since 2014 (we do not take into account 2020, as the data for this year was not yet complete when we acquired it).

All reviewed papers refer to specifically named code smells, not to the more general concept of "code, that requires additional reviewing"—the latter group was usually described as "design flaws", "anomalies" or "inconsistencies" not "code smells".

5.1. RQ1: Is the process of creating code smell machine learning models reproducible in the recent scientific studies on code smell detection using AI/ML methods?

5.1.1. RQ1.1: Do the authors of recent scientific studies on code smell detection using AI/ML methods describe the modeling process with enough details to reproduce their studies?

Out of 46 analyzed publications, 22 described the modeling process with details sufficient to reproduce the study, eight contained some details but not enough to reproduce basing on the paper alone, while sixteen contained only a brief description, such as a listing of used algorithms without any parameters or detailed architecture. Those results are presented in Fig. 4, while a more detailed view, including publication year, is presented in Fig. 5. In recent years, the number of papers describing the process in moderate detail has increased, which can be associated with an increased interest in deep learning techniques. Those models are incredibly complex, so it may be preferable to leave some details out of the paper, assuming that a reproduction package is provided to resolve all ambiguities.

We did not attempt to reproduce all the studies based on the description. Our assessment is based on a reproduction plan that was created for each of them. In the case of *Brief* description, we were not able to create such a plan. In the case of *Moderate*, we were able to draft a plan, but it lacked some details. In the case of *Detailed* description, we were able to create a plan how to reproduce the study. As such, it is possible that we missed some parameters or steps that were actually necessary for reproduction, but we omitted them in the reproduction plan. Thus, the number of *Detailed* studies should be treated as an upper bound and some of those studies will still lack some detail.

Categories of methods (decision tree, SVM) are present in every study, but many studies lack details of configuration, e.g., type of SVM kernel, architecture of a neural network, number of layers or neurons in a neural network.

5.1.2. RQ1.2: Do the authors of recent scientific studies on code smell detection using AI/ML methods publish reproduction packages for their studies?

The results are presented in Fig. 6. 26 publications do not mention any sort of reproduction package. Three publications ([SLR11,SLR8, SLR7]) provide links which we were not able to resolve and one publication ([SLR13]) mentions providing a reproduction package, but we were unable to find any reference to the said package in the rest of the paper (no hyperlink or citation). Of the remaining fifteen publications, seven ([SLR14,SLR10,SLR15,SLR16,SLR17,SLR5,SLR4]) contained both scripts and data in the reproduction package, six ([SLR6,SLR2,SLR18, SLR9,SLR19,SLR1]) contained only data for replications and the last two – [SLR3,SLR12] – contained only reproduction scripts.

Some of those papers cannot provide any reproduction scripts, simply because there was no script running—if a tool was simply invoked

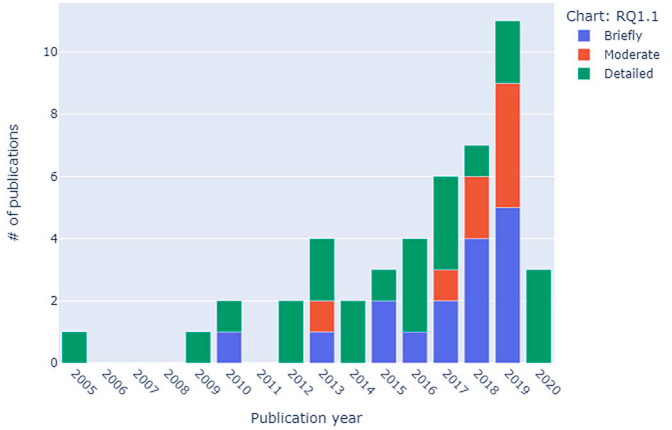


Fig. 5. Number of publications from each year with given level of detail in model description.

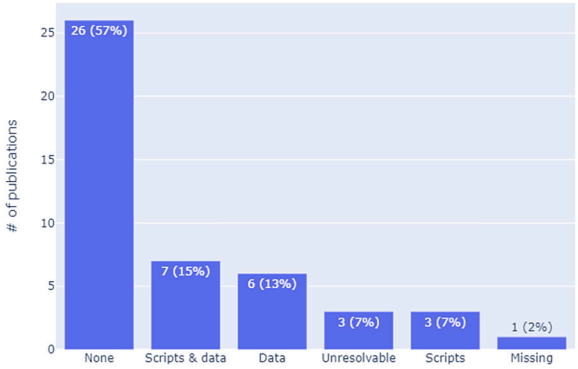


Fig. 6. Reproduction packages in publications.

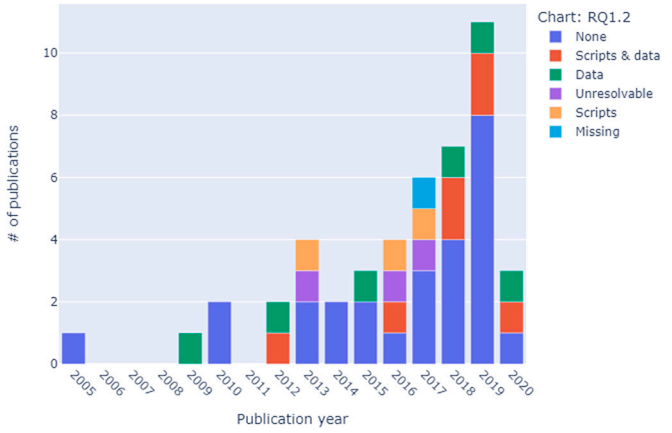


Fig. 7. Number of publications from each year with reproduction package.

with its default parameters, this may have been done manually. In such case, sufficient reproducibility should be provided by describing the process in detail in the paper. For more complex manual workflows, it may be reasonable to use a tool such as DVC [16] to keep track of the research models and their evolution.

A more detailed view, including publication year, is presented in Fig. 7. While the majority of papers do not include any form of reproduction package, it is slightly reassuring that this trend is declining and recently more authors start to include such a package in their research.

5.1.3. RQ1 summary

Overall, as of today, the process of creating models is, with a few exceptions, not reproducible.

> The share of studies with definitely insufficient model description rises, from 28% of the studies before 2018 to 43% in years 2018–2019.

This effect is likely due to increase in the sheer number of papers—from two relevant papers in 2012 to eleven in 2019.

> The share of studies with a full reproduction package (data and scripts) while still low, also rises—from 8% of the studies before 2018 to 22% of the studies in years 2018–2019.

This effect can be attributed to maturing of the domain—since there is more and more research on the subject, authors encounter the same problems that we did and try to mitigate that in their subsequent papers.

5.2. RQ2: Is the process of code smell data set acquisition reproducible in the recent scientific studies on code smell detection using AI/ML methods?

Fifteen publications ([SLR6,SLR2,SLR9,SLR1,SLR17,SLR20,SLR21, SLR22,SLR23,SLR24,SLR25,SLR26,SLR27,SLR28,SLR29]) use predefined data sets, either without applying any modifications or with only cosmetic changes (such as merging two data sets), thus the rest of this section only discusses the remaining 31 publications (although charts show all papers to provide scale).

5.2.1. RQ2.1: How do the authors of recent scientific studies on code smell detection using AI/ML methods select projects for their data sets?

We categorized free-text answers to the applicable question into seven categories: **Predefined** (Using the same data set as another study), **Manual - no criteria** (Manual selection of projects, no selection criteria described in the paper), **Manual - criteria given** (Manual selection of projects, selection criteria were described in free text), **Existing corpus subset** (Projects were selected from an existing corpus, but no selection criteria were given), **Earlier studies** (Selection was inspired by earlier studies, but did not use same data set), **Unknown** (No description of selection process) and **Automated** (Selection was done automatically by a script).

Out of the remaining 31 publications, 22 used manual project selection. In half of the cases selection criteria were not given at all, while in the other half they were briefly mentioned. Criteria were usually related to the projects being used earlier in the research or projects being well-known, nontrivial, known to authors, or diverse. Out of the remaining nine publications, four used a subset of a well-known corpus (Qualitas Corpus [40]), two used projects used in earlier studies, and two publications did not disclose the selection method. In one publication, [SLR14], the authors used an automated project selection method like the one described by Lewowski and Madeyski [19], using the highest number of stars on GitHub as the sole criterion.

The results are visualized in Fig. 8 and, split by publication year, in Fig. 9. Unfortunately, the charts do not show any clear trend.

5.2.2. RQ2.2: How are samples from projects selected for classification in recent scientific studies on code smell detection using AI/ML methods?

We categorized free-text answers to the applicable question into eight categories: **Predefined** (Using the same data set as another study), **Advisors** (Samples for examination were recommended by an existing tool, e.g., JDeodorant. Any such tool is later on referred to as an Advisor), **Exhaustive search** (All possible occurrences were examined), **Unknown** (No selection details were present in the paper), **Bug seeding** (Samples were created by artificially modifying the code),

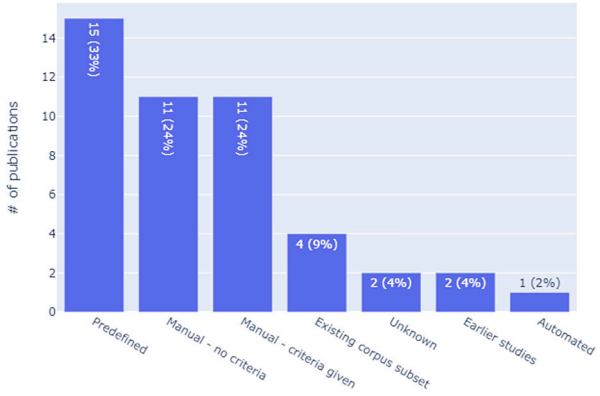


Fig. 8. Project selection methods.

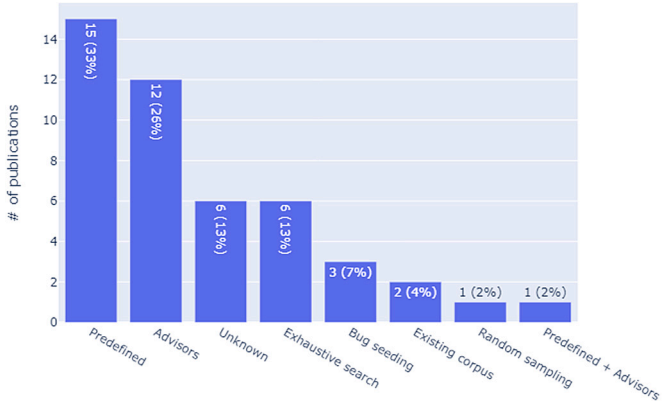


Fig. 10. Sample selection methods.

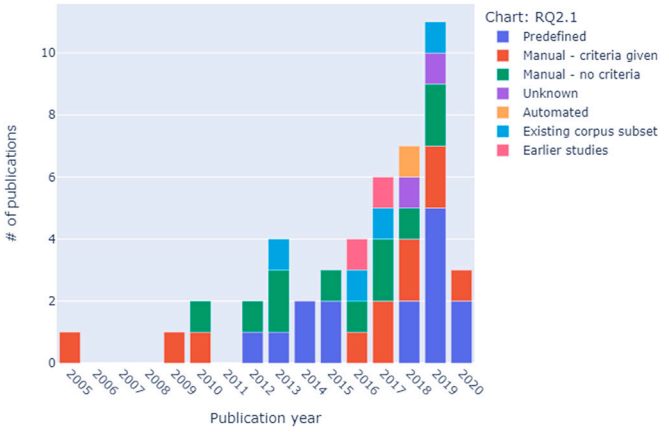


Fig. 9. Number of publications from each year split by project selection method.

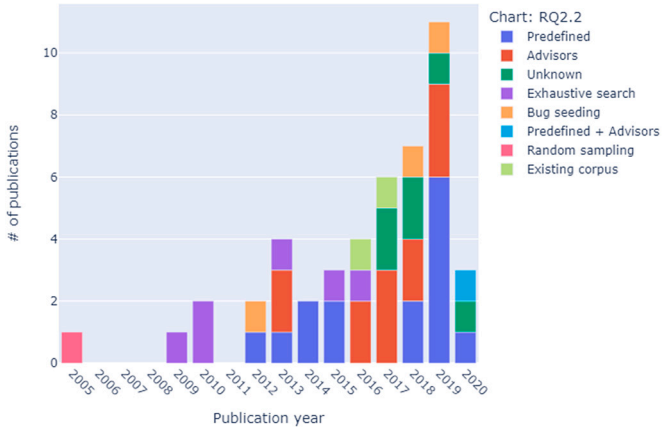


Fig. 11. Number of publications from each year split by sample selection method.

Existing corpus (Samples were taken from existing code smell corpus and examined again), **Predefined + advisors** (Samples were taken from the existing data set, but it was additionally examined using existing programs) and **Random sampling** (Samples were picked randomly from all available ones).

Out of the 31 publications that used custom data sets, six used an exhaustive search (all entities were verified for smell occurrence), three used smell-introducing refactoring or bug seeding, one used random sampling, eleven used advisors (one of them used advisors for one smell and an existing data set for another, so the numbers do not sum up), two used an existing corpus of samples and the method used in six publications remains unknown.

The results are visualized in Fig. 10 and, split by publication year, in Fig. 11. Unfortunately, the charts do not show any clear trend.

5.2.3. RQ2.3: How are samples assessed for occurrence of code smells in recent scientific studies on code smell detection using AI/ML methods?

We categorized free-text answers to the applicable question into six categories: **Predefined** (Using the same data set as another study), **Students** (Assessment was done by graduate or post-graduate students), **Automated** (Other tools are used for assessment), **Unknown** (No assessment details were present in the paper), **Authors** (Study authors were doing the assessment), **Developers** (Assessment was done by software developers). Some studies used multiple categories at the same time, which resulted in three additional categories—**Authors & students**, **Predefined + Students & developers** (predefined samples additionally reviewed) and **Students & developers**.

As for sample assessment, it is usually done either by authors (seven publications - [SLR11,SLR15,SLR30,SLR31,SLR32,SLR33,SLR34]) or students (twelve publications - [SLR14,SLR11,SLR10,SLR12,SLR35,SLR8, SLR18,SLR16,SLR7,SLR36,SLR37,SLR38]). Only three publications ([14, SLR16,SLR36]) mention support from developers or engineers. Only in three publications there were more than ten people involved in data set creation. As for the remaining eleven publications, in the case of seven ([SLR19,SLR39,SLR40,SLR41,SLR42,SLR43,SLR44]) the assessment method is unknown, while in the last four ([SLR5,SLR4, SLR45,SLR46]) samples are graded automatically—either because they are seeded or taken from a bug database or because an automated advisor is configured as automatically accepted.

The results are visualized in Fig. 12 and, split by publication year, in Fig. 13. Unfortunately, the charts do not show any clear trend.

As for predefined data sets, two are predominantly used—one provided in [SLR10] using Qualitas Corpus [40] as project and sample source, and another one that contains data from Azureus, ArgoUML and Xerces-J. Other used data sets include the one published in [SLR8] and another published by Khomh et al. [17].

5.2.4. RQ2 summary

Creating a code smell data set itself is not an easy endeavor, as described in [22].

> The most commonly employed strategy (33% of reviewed publications) is to use an existing data set. This is also a strategy that we recommend, as it lets the researchers focus on the modeling and not on creating an ad hoc data set.

As this study has shown, in the overwhelming majority of papers where authors decide to create a new data set, the projects are

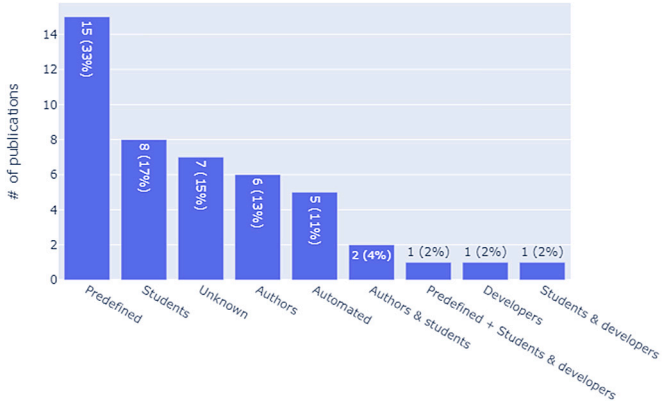


Fig. 12. Sample assessment methods.

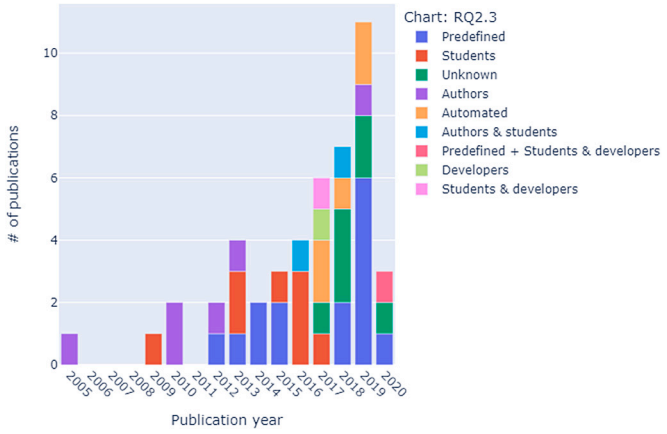


Fig. 13. Number of publications from each year split by sample assessment method.

picked manually, which makes it impossible to automate (since detailed selection rules are not known). After project selection, samples for assessment are usually picked by advisors (such as JDeodorant), which in turn skews the data set in the direction of the advisors' "understanding". Finally, code smell assessment is usually done by students or authors, who typically are not experienced developers. It is also fairly common (ca. 10% of reviewed studies) to use automated assessment by another tool, e.g., JDeodorant, which renders the whole machine learning procedure useless, since there is already an automated algorithm to perform the task. Overall, as of today, the process of code smell sample acquisition is generally not reproducible.

6. Discussion and study implications

Overall reproducibility. We found out that the current state of the studies on code smell detection methods leaves a lot to be desired. In particular, less than 20% of the studies include a full reproduction package (scripts and data), less than 45% of the studies mention any package (e.g., only used data or created models) and in less than 40% of the cases the package is accessible. A third of all included studies contains a description of a modeling process that is insufficient to allow reproduction without access to additional information. Most of those papers also do not include any sort of reproduction package, which means that their results are unlikely to be reproduced. While Madeyski and Kitchenham [21] raised awareness of the problems caused by unreproducible research in software engineering in general, in this paper we focus on one active research area, code smell detection, and show the details, as well as scale of the problem in that particular area. Despite the fact that our study is limited to just one of the research areas in software engineering, similar results were obtained also by Rodriguez [33], so it is unlikely that this area is unique regarding to reproducibility problems, while the implications of these problems are serious.

> **Implications:** The lack of reproducibility in scientific research implicates that the results are unlikely to directly impact:
>
> - the industry (as, e.g., it is difficult to trust, invest in and apply in practice ideas or findings that are not possible to reproduce) or
> - academia (as, e.g., it is much more difficult to verify the findings, as well as build upon previous research, i.e., *build upon the shoulders of giants* [3], when there is no data and/or scripts).
>
> Thus, it lowers scientific output efficiency, slows down scientific progress, and leads to wasted time and money.

Reproduction packages. We encountered a number of studies describing increasingly complex algorithms (such as deep learning ones). Those algorithms often include an enormous amount of parameters, and it is not possible to include them all in the publication when they are not the core research subject of the paper and the paper has to adhere to strict size limits.

> **Implications:** Papers, including complex (e.g., deep learning) algorithms in particular, should provide reproduction scripts, since the implicit parameters, unknown to other researchers, may be crucial to the obtained results. We urge the researchers to include reproduction scripts not only because it substantially improves study reproducibility, but also because it impacts validity of the results. The latter effect is caused by the fact that the researcher has to once again get familiar with details of the applied procedure when preparing a reproduction package. The original thought path for the procedure might have been forgotten by this moment, so the researcher is more likely to spot defects or inconsistencies.

An example to support the above implication: when preparing the reproduction data set for this publication, we noticed that two of the publications accepted in the initial screening were omitted in the data collection sheet, and, as such, were not actually included in the study. Those publications did not change the study conclusions substantially, but the sole act of providing a reproduction package allowed us to improve the paper. It is unlikely that our research is unique in that matter, thus we strongly recommend providing a reproduction package.

Data set creation. Our study found out that only a third of the analyzed studies used a predefined data set, while others introduced their own data sets, usually without a detailed description. This finding supports the claim made by Caram et al. [4]—data sets often are created independently and researchers do not share the details of their data set creation techniques. Since there are only few clues as to how to acquire a similar data set and the data sets are rarely published, the research, even if valid, becomes irreproducible.

While it is understandable that many researchers will need to create their own data sets – they may be interested in various languages or application types – the description of the data set creation should be explicit. For selecting software projects, we recommend an automated approach, for example, by following the process described by Lewowski and Madeyski [19]. The data sets should also be described in detail, for instance using a data sheet similar to one published by Gebru et al. [12].

Implications: Researchers who decide to create a new data set and do not share the details of how were the data gathered (e.g., in form of a data sheet) severely hinder reproducibility of their research.

Invalid link translation. A moderately common problem encountered in our study (two out of twenty publications with package links) is the poor quality of linking in research papers—for example, URLs containing a tilde character (~) can either contain an ASCII tilde (~) or one of eight other UTF tilde-like characters (e.g. ~, used by LaTeX in Math environments). Depending on the font, they may be either the same or extremely similar symbol and even a skilled reader is unlikely to be able to distinguish between them. However, the requests made by the browser will differ, and only one of them will lead to the right location (HTTP server will reject the others, usually with `404: Not Found` or `403: Forbidden` status code). In the two described cases ([SLR2,SLR6]) the problem lies somewhere inside the mechanism used for translation between document formats (PDF and HTML), where particular characters were misread. Both those papers are published by IEEE and are accessible on the IEEE Xplore website.

Implications: Researchers and practitioners should be aware that such a problem in translation between various media for same paper may exist and, if a hyperlink from one version of a paper does not point the desired resource, verify the versions present in other formats of said document.

While the publisher is responsible for making sure that all content from the paper is correctly translated into all supported media, occasionally problems with the said translation do arise and researchers have to deal with them on an ad hoc basis.

Online data appendices. We found out that online appendices (reproduction scripts, data sets, prebuilt models) are often published on research group sites or even single researchers' personal sites—only one of analyzed papers, ([SLR9]) used an established data archive, despite those archives being available since at least 2012. While this is a substantial improvement over not publishing them at all, this solution leaves a lot to be desired. First, those sites usually do not offer any permanent linking mechanism (e.g., by DOI). Therefore, any change in organization layout (e.g., renaming of the working group or author changing affiliation) will render those links invalid, even if the data is still valid and still available on the web. Second, they do not offer any structured versioning mechanism, so a reader several years from the first publishing cannot be sure if the version they use is the same version that is referred to in the paper. Third, those sites are often volatile and can disappear from the web together with the data held by them (such a thing happened with packages published for [SLR7] and [SLR8]). Sometimes some of those problems can be resolved by contacting the corresponding author directly, but it is not always the case and it is not a scalable solution for data sharing. Handling of data sets and other digital artifacts is an important matter for the overall quality of science. This has been acknowledged by international authorities, e.g., in the form of the OECD recommendation, US NIH data stewardship organization [34] or UN data policies. While those documents are discussed, reviewed, and sometimes criticized [23], there is little doubt that this is a needed discussion and recommendations for artifact management are crucial for research reproducibility.

Implications: Authors should be aware of the maintenance requirements arising from the use of custom infrastructure and, whenever possible, use a dedicated archive, such as Zenodo, Figshare, Dryad or the infrastructure provided by the publisher, which provides DOIs, versioning, platform maintenance and other features helpful for increasing data visibility and durability.

Online source code appendices. Half of the analyzed papers that included any reproduction package included some sort of scripts or program. In software engineering, it is common that one of the artifacts is the source code of an application—in the case of seven papers the researchers adopted the typical way of doing things in software development and published the code on GitHub or other public source code repository system (GitLab, BitBucket, Savannah), while all the others who chose to publish their scripts have done that on their own, custom sites (either private or research group site). We strongly recommend using version control for source code-related artifacts, both for reproducibility and for possibility to track changes between subsequent research papers. The dominant version control system as of today, at least for open source code, is Git. Git assigns a checksum to each version of the repository, so, technically speaking, giving the reader this checksum is the most precise version information available. However, to adhere to scientific conventions, GitHub offers a possibility of assigning a DOI to a specific repository version. Of course, underneath this is really no different from a specific release tag and less specific than a commit SHA,[4] but may be more readable for researchers from other domains, not familiar with modern software development techniques. Researchers interested mostly in data modeling may prefer a tool such as DVC [16], tailored to the needs of data science and machine learning. For source code artifacts, as of today, GitHub is the only archive we are aware of that makes it possible to assign a DOI to a source code artifact without decoupling it from its history, although other providers are working on this feature as well.

Implications: Software is much more volatile than publications. For reproducibility, it is important to mark (with tag, release, SHA or DOI) a specific version used during the research. Still, further improvements should not be ignored, so public source code repositories should be preferred over research group websites.

Code smell assessment. This study was focused on machine learning techniques in code smell detection and all 46 papers use some machine learning techniques to try to detect code smells. Using machine learning implies that there is no simple, known algorithm for code smell detection and that the concept is understood by developers, who know how to identify it—for example, by using own domain and technical expertise. In this context, it is puzzling that only three of the analyzed papers explicitly mention any support from software developers in code smell detection, while over 10 explicitly mention that data set was obtained by students. We urge creators of the future code smell data sets to use the domain knowledge of the experts in the field (in this case—experienced software developers, as we did in [22]) as the source of assessment.

Implications: Using data from ad hoc trained participants will only cause confusion, as those participants will usually replicate simple rules that they were taught—and if the rules are simple, they can be programmed directly, and there would be no need to apply machine learning to this problem. As of now, as shown by Hozano et al. [15], it is not even clear whether the understanding of code smells is consistent among seasoned developers, not to mention the less experienced ones.

Separate data set creation. Our study discovered that the most widely used predefined data sets [25], [SLR35] are published as part of wider studies on code smell detection. Since the papers that introduce the data sets are focused on code smell detection, it is harder to

[4] SHA is a unique fingerprint, while tags can technically be moved, even if it is discouraged in practice

find information about data set details, including format and detailed creation procedure, thus the origins of the data set may be easier lost.

It has recently been understood that creating reliable data sets is more challenging than one would initially expect. There are separate conferences dedicated to data publishing (e.g., MSR) and data papers are becoming increasingly common. We recommend that the authors of new data sets publish them separately from their main research papers—this way the data set becomes the core artifact and it is possible to avoid introducing study-specific variables to the data itself and to focus effort on curating the data, not only on using it.

> **Implications:** Data sets published as part of a wider study (e.g., related to code smell detection) and not as separate artifacts are at risk of (even inadvertently) being fine-tuned to the specific solution that the researcher is presenting, instead of being a sample of the data from the problem domain. They are also less likely to contain detailed data sheets and descriptions, as they are not the primary focus for researchers and reviewers of a given paper.

Call for standardization. Our research has shown that about two-thirds of research papers use a custom data set. This is understandable, since the number of programming languages, domains, and paradigms is overwhelming. However, the gathering of said data would be substantially simplified if there was a shared tool that would be able to generate standardized outputs and allow the experts to easily reconfigure it and share snapshots. A step in this direction was published with the Landfill project by Palomba et al. [30], but its goal was less ambitious, thus it cannot be easily used to configure multiple grades (some researchers try to quantify the "smelliness" of the source code, others treat it as a binary property) or to analyze languages other than Java.

> **Implications:** Lack of standardized tools for code smell data acquisition makes it more difficult to create reliable and reusable data sets and perform code smell related research.

6.1. Threats to validity

For the purpose of this study, we consider *construct validity*, *internal validity*, *external validity* and *reliability*.

6.1.1. Construct validity

Construct validity is concerned with the degree to which the study measures what it claims it does.

We believe that the main threat to construct validity is related to the chosen publication set—publications were taken from a single database (Scopus). To address this threat, we ran a Quasi-gold standard procedure to verify the search quality, but unfortunately the procedure did not give us definitive results.

The publication set was initially selected to serve a different purpose —assessing state-of-the-art in the area of code smell detection for the *code quest* company that develops *codebeat* code review platform. We believe this does not have any substantial impact on the outcome, since our review was focused on the same specific group of papers. Additionally, our findings are consistent with those from other researchers in the area of software engineering [33].

An additional threat to the construct validity is that we did not perform snowballing. Thus, the risk of missing out papers, and therefore not adequately implementing the construct "all published papers on the topic", is higher. While snowballing might improve the confidence in the selected publication set, the number of analyzed papers was already bigger than in the case of other systematic reviews which dealt with the problem of machine learning for code smell detection. We decided that this was a sufficient reason to skip the effort-consuming process of snowballing and instead focus on curating the data for further use, including replication and verification.

6.1.2. Internal validity

Internal validity is concerned with how the research was conducted.

The data acquisition procedure was manual, so this is a natural place for errors to appear. To address this threat, we performed a cross-check for all gathered data points in the data collection sheet and for a random sample in the initial screening. For one of the publication questions, the agreement rate after cross-checking the first 25% was too low, so we regathered all of the data for it and ran a cross-check for all remaining papers. To further improve reliability, we publish the full data collection sheet created for the purpose of this publication and another sheet with initial publication screening data.

While the data collection cross-check was complete, the one applied to the initial screening was much briefer—only a tiny percentage of initially rejected papers were double-checked for relevance. While we carefully examined the data to make sure that relevant papers are included, it is possible that some were rejected at that stage. However, to address such a threat, we include data from the whole initial screening in the reproduction data package.

6.1.3. External validity

External validity is concerned with the possibility of generalizing the findings. This review was performed only on a selected problem in software engineering—reproducibility of research in the area of code smell detection using machine learning techniques. We do not claim that the same issues can be observed throughout the whole area of software engineering, although there are other studies, such as [33], that obtain similar results in different subdomains.

6.1.4. Reliability

Reliability is concerned with the possibility to reproduce the research and achieve the same results. To guarantee the maximum possibility for reproduction, we describe the research procedure in detail in Section 4 and attach links to the gathered data and processing scripts. For the steps that were performed manually in the reproduction package, we include all created intermediate artifacts.

7. Conclusion

Our findings are consistent with those of other researchers, e.g., [33]—nearly half of the publications did not describe the methodology in detail sufficient for reproduction, over 60% do not include any form of reproduction package and less than 20% provide a full reproduction package (scripts and data sets).

Reproducibility in code smell detection should be treated more seriously. This applies to all procedures—as of today, we are not even sure if we are all discussing the same concepts.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.infsof.2021.106783.

Acknowledgment

This research was partly financed by Polish National Centre for Research and Development, Poland grant POIR.01.01.01-00-0792/16: "Codebeat - wykorzystanie sztucznej inteligencji w statycznej analizie jakości oprogramowania."

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.infsof.2021.106783.

References

[1] A. Al-Shaaby, H. Aljamaan, M. Alshayeb, Bad smell detection using machine learning techniques: A systematic literature review, Arab. J. Sci. Eng. 45 (2020) http://dx.doi.org/10.1007/s13369-019-04311-w.

[2] M.I. Azeem, F. Palomba, L. Shi, Q. Wang, Machine learning techniques for code smell detection: A systematic literature review and meta-analysis, Inf. Softw. Technol. (ISSN: 0950-5849) 108 (2019) 115–138, http://dx.doi.org/10.1016/j.infsof.2018.12.009.

[3] E. Barr, C. Bird, E. Hyatt, T. Menzies, G. Robles, On the shoulders of giants, in: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research, FoSER '10, ACM, ISBN: 978-1-4503-0427-6, 2010, pp. 23–28, http://dx.doi.org/10.1145/1882362.1882368.

[4] F. Caram, B.R. de Oliveira Rodrigues, A. Campanelli, F. Silva Parreiras, Machine learning techniques for code smells detection: A systematic mapping study, Int. J. Softw. Eng. Knowl. Eng. 29 (2019) 285–316, http://dx.doi.org/10.1142/S021819401950013X.

[5] J.B. Carlisle, Data fabrication and other reasons for non-random sampling in 5087 randomized, controlled trials in anaesthetic and general medical journals, Anaesthesia 72 (8) (2017) 944–952, http://dx.doi.org/10.1111/anae.13938.

[6] B. Chen, Z.M.J. Jiang, Characterizing and detecting anti-patterns in the logging code, in: Proceedings of the 39th International Conference on Software Engineering, ICSE '17, IEEE Press, ISBN: 9781538638682, 2017, pp. 71–81, http://dx.doi.org/10.1109/ICSE.2017.15.

[7] O. Dieste, A. Grimán, N. Juristo, Developing search strategies for detecting relevant experiments, Empir. Softw. Eng. 14 (5) (2009) 513–539.

[8] T. Dybå, T. Dingsøyr, Empirical studies of agile software development: A systematic review, Inf. Softw. Technol. 50 (9–10) (2008) 833–859.

[9] F.A. Fontana, M.V. Mäntylä, M. Zanoni, A. Marino, Comparing and experimenting machine learning techniques for code smell detection, Empir. Softw. Eng. 21 (3) (2016) 1143–1191, http://dx.doi.org/10.1007/s10664-015-9378-4.

[10] F.A. Fontana, I. Pigazzini, R. Roveda, M. Zanoni, Automatic detection of instability architectural smells, in: 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME, 2016, pp. 433–437, http://dx.doi.org/10.1109/ICSME.2016.33.

[11] M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts, Refactoring: Improving the Design of Existing Code, Addison-Wesley, Boston, MA, USA, 1999.

[12] T. Gebru, J. Morgenstern, B. Vecchione, J.W. Vaughan, H.M. Wallach, H.D. III, K. Crawford, Datasheets for datasets, 2018, , CoRR abs/1803.09010.

[13] F. Hesselmann, V. Graf, M. Schmidt, M. Reinhart, The visibility of scientific misconduct: A review of the literature on retracted journal articles, Curr. Sociol. 65 (6) (2017) 814–845, http://dx.doi.org/10.1177/0011392116663807.

[14] M. Hozano, N. Antunes, B. Fonseca, E. Costa, Evaluating the accuracy of machine learning algorithms on detecting code smells for different developers, in: Proceedings of the 19th International Conference on Enterprise Information Systems - Volume 2: ICEIS, SciTePress, INSTICC, 2017, pp. 474–482, http://dx.doi.org/10.5220/0006338804740482.

[15] M. Hozano, A. Garcia, N. Antunes, B. Fonseca, E. Costa, Smells are sensitive to developers!: On the efficiency of (un)guided customized detection, in: Proceedings of the 25th International Conference on Program Comprehension, ICPC '17, IEEE Press, Piscataway, NJ, USA, 2017, pp. 110–120.

[16] Iterative.ai, Data version control, http://https://dvc.org/.

[17] F. Khomh, M. Di Penta, Y.-G. Guéhéneuc, G. Antoniol, An exploratory study of the impact of antipatterns on class change- and fault-proneness, Empir. Softw. Eng. 17 (2012) 243–275, http://dx.doi.org/10.1007/s10664-011-9171-y.

[18] B. Kitchenham, D. Budgen, P. Brereton, Evidence-Based Software Engineering and Systematic Reviews, CRC Press, 2016, http://dx.doi.org/10.1201/b19467.

[19] T. Lewowski, L. Madeyski, Creating evolving project data sets in software engineering, in: S. Jarzabek, A. Poniszewska-Marańda, L. Madeyski (Eds.), Integrating Research and Practice in Software Engineering, in: Studies in Computational Intelligence, vol. 851, Springer, Cham, 2020, pp. 1–14, http://dx.doi.org/10.1007/978-3-030-26574-8_1.

[20] T. Lewowski, L. Madeyski, Code smells detection using artificial intelligence techniques: A business-driven systematic review, in: N. Kryvinska, A. Poniszewska-Marańda (Eds.), Developments in Information & Knowledge Management for Business Applications : Volume 3, Springer, Cham, 2022, pp. 285–319, http://dx.doi.org/10.1007/978-3-030-77916-0_12.

[21] L. Madeyski, B. Kitchenham, Would wider adoption of reproducible research be beneficial for empirical software engineering research? J. Intell. Fuzzy Systems 32 (2017) 1509–1521, http://dx.doi.org/10.3233/JIFS-169146.

[22] L. Madeyski, T. Lewowski, MLCQ: INdustry-relevant code smell data set, in: Proceedings of the Evaluation and Assessment in Software Engineering, EASE '20, ACM, New York, NY, USA, 2020, pp. 342–347, http://dx.doi.org/10.1145/3383219.3383264.

[23] L. Madeyski, T. Lewowski, B. Kitchenham, OECD Recommendation's draft concerning access to research data from public funding: A review, Bull. Pol. Acad. Sci.: Tech. Sci. 69 (2021) http://dx.doi.org/10.24425/bpasts.2020.135401.

[24] R. Malhotra, A systematic review of machine learning techniques for software fault prediction, Appl. Soft Comput. 27 (2015) 504–518.

[25] N. Moha, Y.-G. Gueheneuc, L. Duchien, A.-F. Le Meur, DECOR: A Method for the specification and detection of code and design smells, IEEE Trans. Softw. Eng. 36 (1) (2010) 20–36, http://dx.doi.org/10.1109/TSE.2009.50.

[26] T. Mousavi, M. Abdollahi, A review of the current concerns about misconduct in medical sciences publications and the consequences, DARU J. Pharm. Sci. 28 (2020) 1–11, http://dx.doi.org/10.1007/s40199-020-00332-1.

[27] National Academies of Sciences, Engineering, and Medicine, Reproducibility and Replicability in Science, The National Academies Press, Washington, DC, ISBN: 978-0-309-48616-3, 2019, http://dx.doi.org/10.17226/25303, https://www.nap.edu/catalog/25303/reproducibility-and-replicability-in-science.

[28] F. Palomba, Alternative sources of information for code smell detection: Postcards from far away, in: 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME, 2016, pp. 636–640, http://dx.doi.org/10.1109/ICSME.2016.26.

[29] F. Palomba, D. Di Nucci, A. Panichella, A. Zaidman, A. De Lucia, Lightweight detection of android-specific code smells: The adoctor project, in: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER, 2017, pp. 487–491, http://dx.doi.org/10.1109/SANER.2017.7884659.

[30] F. Palomba, D. Di Nucci, M. Tufano, G. Bavota, R. Oliveto, D. Poshyvanyk, A. De Lucia, Landfill: An open dataset of code smells with public evaluation, in: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, 2015, pp. 482–485. http://dx.doi.org/10.1109/MSR.2015.69.

[31] F. Palomba, A. Panichella, A. Zaidman, R. Oliveto, A. De Lucia, The scent of a smell: An extensive comparison between textual and structural smells, IEEE Trans. Softw. Eng. (2017) http://dx.doi.org/10.1109/TSE.2017.2752171.

[32] G. Rasool, Z. Arshad, A review of code smell mining techniques, J. Softw.: Evol. Process 27 (11) (2015) 867–895, http://dx.doi.org/10.1002/smr.1737.

[33] G. Rodríguez-Pérez, G. Robles, J.M. González-Barahona, Reproducibility and credibility in empirical software engineering: A case study based on a systematic literature review of the use of the SZZ algorithm, Inf. Softw. Technol. (ISSN: 0950-5849) 99 (2018) 164–176, http://dx.doi.org/10.1016/j.infsof.2018.03.009.

[34] S. Rosenbaum, Data governance and stewardship: Designing data stewardship entities and advancing data access, Health Serv. Res. 45 (5p2) (2010) 1442–1455, http://dx.doi.org/10.1111/j.1475-6773.2010.01140.x.

[35] K. Rozier, E. Rozier, Reproducibility, correctness, and buildability: the three principles for ethical public dissemination of computer science and engineering research, in: 2014 IEEE International Symposium on Ethics in Science, Technology and Engineering, ETHICS 2014, 2014, http://dx.doi.org/10.1109/ETHICS.2014.6893384.

[36] J.A.M. Santos, J. ao B. Rocha-Junior, L.C.L. Prates, R.S. do Nascimento, M. a Falcão Freitas, M.G. de Mendonça, A systematic review on the code smell effect, J. Syst. Softw. (ISSN: 0164-1212) 144 (2018) 450–477, http://dx.doi.org/10.1016/j.jss.2018.07.035.

[37] T. Sharma, D. Spinellis, A survey on software smells, J. Syst. Softw. (ISSN: 0164-1212) 138 (2018) 158–173, http://dx.doi.org/10.1016/j.jss.2017.12.034.

[38] M. Shepperd, N. Ajienka, S. Counsell, The role and value of replication in empirical software engineering results, Inf. Softw. Technol. (ISSN: 0950-5849) 99 (2018) 120–132, http://dx.doi.org/10.1016/j.infsof.2018.01.006.

[39] S. Singh, S. Kaur, A systematic literature review: Refactoring for disclosing code smells in object oriented software, Ain Shams Eng. J. (ISSN: 2090-4479) (2017) http://dx.doi.org/10.1016/j.asej.2017.03.002.

[40] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, J. Noble, Qualitas corpus: A curated collection of java code for empirical studies, in: 2010 Asia Pacific Software Engineering Conference, APSEC2010, 2010, pp. 336–345, http://dx.doi.org/10.1109/APSEC.2010.46.

[41] J. Wen, S. Li, Z. Lin, Y. Hu, C. Huang, Systematic literature review of machine learning based software development effort estimation models, Inf. Softw. Technol. 54 (1) (2012) 41–59, http://dx.doi.org/10.1016/j.infsof.2011.09.002.

[42] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering EASE'14, 2014. http://dx.doi.org/10.1145/2601248.2601268.

[43] J. Yao, M. Shepperd, The impact of using biased performance metrics on software defect prediction research, Inf. Softw. Technol. (ISSN: 0950-5849) 139 (2021) 106664, http://dx.doi.org/10.1016/j.infsof.2021.106664.

[44] H. Zhang, M.A. Babar, P. Tell, Identifying relevant studies in software engineering, Inf. Softw. Technol. 53 (6) (2011) 625–637.

Systematic Literature Review References

[SLR1] H. Grodzicka, A. Ziobrowski, Z. Łakomiak, M. Kawa, L. Madeyski, Code smell prediction employing machine learning meets emerging java language constructs, in: A. Poniszewska-Marańda, N. Kryvinska, S. Jarzabek, L. Madeyski (Eds.), Data-Centric Business and Applications: Towards Software Development (Volume 4), in: vol. 40 of book series Lecture Notes on Data Engineering and Communications Technologies, Springer International Publishing, Cham, 2020, pp. 137–167, http://dx.doi.org/10.1007/978-3-030-34706-2_8.

[SLR2] A. Maiga, N. Ali, N. Bhattacharya, A. Sabané, Y.-G. Guéhéneuc, E. Aimeur, SMURF: A SVM-based incremental anti-pattern detection approach, in: 2012 19th Working Conference on Reverse Engineering, 2012, pp. 466–475, http://dx.doi.org/10.1109/WCRE.2012.56.

[SLR3] F.S. Ocariza, K. Pattabiraman, A. Mesbah, Detecting unknown inconsistencies in web applications, in: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE, 2017, pp. 566–577, http://dx.doi.org/10.1109/ASE.2017.8115667.

[SLR4] H. Liu, Z. Xu, Y. Zou, Deep learning based feature envy detection, in: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Association for Computing Machinery, New York, NY, USA, 2018, pp. 385–396, http://dx.doi.org/10.1145/3238147.3238166.

[SLR5] H. Liu, J. Jin, Z. Xu, Y. Bu, Y. Zou, L. Zhang, Deep learning based code smell detection, IEEE Trans. Softw. Eng. (2019) http://dx.doi.org/10.1109/TSE.2019.2936376.

[SLR6] L. Amorim, E. Costa, N. Antunes, B. Fonseca, M. Ribeiro, Experience report: Evaluating the effectiveness of decision trees for detecting code smells, in: 2015 IEEE 26th International Symposium on Software Reliability Engineering, ISSRE, 2015, pp. 261–269, http://dx.doi.org/10.1109/ISSRE.2015.7381819.

[SLR7] F.A. Fontana, M. Zanoni, Code smell severity classification using machine learning techniques, Knowl.-Based Syst. 128 (2017) 43–58, http://dx.doi.org/10.1016/j.knosys.2017.04.014.

[SLR8] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, D. Poshyvanyk, Detecting bad smells in source code using change history information, in: 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE, 2013, pp. 268–278, http://dx.doi.org/10.1109/ASE.2013.6693086.

[SLR9] T. Guggulothu, S.A. Moiz, Code smell detection using multi-label classification approach, Softw. Qual. J. (2020) http://dx.doi.org/10.1007/s11219-020-09498-y.

[SLR10] F.A. Fontana, M.V. Mäntylä, M. Zanoni, A. Marino, Comparing and experimenting machine learning techniques for code smell detection, Empir. Softw. Eng. 21 (3) (2016) 1143–1191, http://dx.doi.org/10.1007/s10664-015-9378-4.

[SLR11] U. Mansoor, M. Kessentini, B.R. Maxim, K. Deb, Multi-objective code-smells detection using good and bad design examples, Softw. Qual. J. 25 (2) (2017) 529–552, http://dx.doi.org/10.1007/s11219-016-9309-7.

[SLR12] H. Liu, Q. Liu, Z. Niu, Y. Liu, Dynamic and automatic feedback-based threshold adaptation for code smell detection, IEEE Trans. Softw. Eng. 42 (6) (2016) 544–558, http://dx.doi.org/10.1109/TSE.2015.2503740.

[SLR13] M. Hozano, N. Antunes, B. Fonseca, E. Costa, Evaluating the accuracy of machine learning algorithms on detecting code smells for different developers, in: Proceedings of the 19th International Conference on Enterprise Information Systems - Volume 2: ICEIS, SciTePress, INSTICC, 2017, pp. 474–482, http://dx.doi.org/10.5220/0006338804740482.

[SLR14] Z. Chen, L. Chen, W. Ma, X. Zhou, Y. Zhou, B. Xu, Understanding metric-based detectable smells in Python software: A comparative study, Inf. Softw. Technol. 94 (2018) 14–29, http://dx.doi.org/10.1016/j.infsof.2017.09.011.

[SLR15] M. Pradel, S. Heiniger, T.R. Gross, Static detection of brittle parameter typing, in: Proceedings of the 2012 International Symposium on Software Testing and Analysis, ISSTA 2012, Association for Computing Machinery, New York, NY, USA, 2012, pp. 265–275, http://dx.doi.org/10.1145/2338965.2336785.

[SLR16] A. Barbez, F. Khomh, Y.-G. Guéhéneuc, A machine-learning based ensemble method for anti-patterns detection, J. Syst. Softw. 161 (2020) http://dx.doi.org/10.1016/j.jss.2019.110486.

[SLR17] A. Barbez, F. Khomh, Y.-G. Guéhéneuc, Deep learning anti-patterns from code metrics history, in: 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME, 2019, pp. 114–124, http://dx.doi.org/10.1109/ICSME.2019.00021.

[SLR18] F. Khomh, S. Vaucher, Y.-G. Guéhéneuc, H. Sahraoui, A Bayesian approach for the detection of code and design smells, in: 2009 Ninth International Conference on Quality Software, 2009, pp. 305–314, http://dx.doi.org/10.1109/QSIC.2009.47.

[SLR19] S. Fakhoury, V. Arnaoudova, C. Noiseux, F. Khomh, G. Antoniol, Keep it simple: Is deep learning good for linguistic smell detection? in: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering, SANER, 2018, pp. 602–611, http://dx.doi.org/10.1109/SANER.2018.8330265.

[SLR20] F. Palomba, Textual analysis for code smell detection, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 2, 2015, pp. 769–771, http://dx.doi.org/10.1109/ICSE.2015.244.

[SLR21] W. Kessentini, M. Kessentini, H. Sahraoui, S. Bechikh, A. Ouni, A cooperative parallel search-based software engineering approach for code-smells detection, IEEE Trans. Softw. Eng. 40 (9) (2014) 841–861, http://dx.doi.org/10.1109/TSE.2014.2331057.

[SLR22] D. Sahin, M. Kessentini, S. Bechikh, K. Deb, Code-smell detection as a bilevel problem, ACM Trans. Softw. Eng. Methodol. 24 (1) (2014) http://dx.doi.org/10.1145/2675067.

[SLR23] M. Boussaa, W. Kessentini, M. Kessentini, S. Bechikh, S. Ben Chikha, Competitive coevolutionary code-smells detection, in: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8084, 2013, pp. 50–65, http://dx.doi.org/10.1007/978-3-642-39742-4_6.

[SLR24] X. Guo, C. Shi, H. Jiang, Deep semantic-based feature envy identification, in: Proceedings of the 11th Asia-Pacific Symposium on Internetware, Internetware '19, Association for Computing Machinery, New York, NY, USA, 2019, http://dx.doi.org/10.1145/3361242.3361257.

[SLR25] E.O. Kiyak, D. Birant, K.U. Birant, Comparison of multi-label classification algorithms for code smell detection, in: 2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies, ISMSIT, 2019, pp. 1–6, http://dx.doi.org/10.1109/ISMSIT.2019.8932855.

[SLR26] M. Hadj-Kacem, N. Bouassida, Deep representation learning for code smells detection using variational auto-encoder, in: 2019 International Joint Conference on Neural Networks, IJCNN, 2019, pp. 1–8, http://dx.doi.org/10.1109/IJCNN.2019.8851854.

[SLR27] S. Tummalapalli, L. Kumar, L.B.M. Neti, An empirical framework for web service anti-pattern prediction using machine learning techniques, in: 2019 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference, IEMECON, 2019, pp. 137–143, http://dx.doi.org/10.1109/IEMECONX.2019.8877008.

[SLR28] K. Karađuzović-Hadžiabdić, R. Spahić, Comparison of machine learning methods for code smell detection using reduced features, in: 2018 3rd International Conference on Computer Science and Engineering, UBMK, 2018, pp. 670–672, http://dx.doi.org/10.1109/UBMK.2018.8566561.

[SLR29] M. Hadj-Kacem, N. Bouassida, A hybrid approach to detect code smells using deep learning, in: Proceedings of the 13th International Conference on Evaluation of Novel Approaches To Software Engineering - ENASE, Vol. 2018-March, 2018, pp. 137–146, http://dx.doi.org/10.5220/0006709801370146.

[SLR30] F. Gauthier, E. Merlo, Semantic smells and errors in access control models: A case study in PHP, in: 2013 35th International Conference on Software Engineering, ICSE, 2013, pp. 1169–1172, http://dx.doi.org/10.1109/ICSE.2013.6606670.

[SLR31] S. Hassaine, F. Khomh, Y.-G. Gueheneuc, S. Hamel, IDS: AN immune-inspired approach for the detection of software design smells, in: 2010 Seventh International Conference on the Quality of Information and Communications Technology, 2010, pp. 343–348, http://dx.doi.org/10.1109/QUATIC.2010.61.

[SLR32] S. Bryton, F. Brito e Abreu, M. Monteiro, Reducing subjectivity in code smells detection: Experimenting with the long method, in: 2010 Seventh International Conference on the Quality of Information and Communications Technology, 2010, pp. 337–342, http://dx.doi.org/10.1109/QUATIC.2010.60.

[SLR33] J. Kreimer, Adaptive detection of design flaws, Electron. Notes Theor. Comput. Sci. 141 (4 SPEC. ISS.) (2005) 117–136, http://dx.doi.org/10.1016/j.entcs.2005.02.059.

[SLR34] J. Rubin, A.N. Henniche, N. Moha, M. Bouguessa, N. Bousbia, Sniffing android code smells: An association rules mining-based approach, in: 2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems, MOBILESoft, 2019, pp. 123–127, http://dx.doi.org/10.1109/MOBILESoft.2019.00025.

[SLR35] F.A. Fontana, M. Zanoni, A. Marino, M.V. Mäntylä, Code smell detection: Towards a machine learning-based approach, in: 2013 IEEE International Conference on Software Maintenance, 2013, pp. 396–399, http://dx.doi.org/10.1109/ICSM.2013.56.

[SLR36] M. Kessentini, A. Ouni, Detecting android smells using multi-objective genetic programming, in: 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems, MOBILESoft, 2017, pp. 122–132, http://dx.doi.org/10.1109/MOBILESoft.2017.29.

[SLR37] M.W. Mkaouer, Interactive code smells detection: An initial investigation, Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 9962 (2016) 281–287, http://dx.doi.org/10.1007/978-3-319-47106-8_24.

[SLR38] S. Fu, B. Shen, Code bad smell detection through evolutionary data mining, in: 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM, 2015, pp. 1–9, http://dx.doi.org/10.1109/ESEM.2015.7321194.

[SLR39] A. Kaur, S. Jain, S. Goel, A support vector machine based approach for code smell detection, in: 2017 International Conference on Machine Learning and Data Science, MLDS, 2017, pp. 9–14, http://dx.doi.org/10.1109/MLDS.2017.8.

[SLR40] A. Kaur, S. Jain, S. Goel, SP-J48: a novel optimization and machine-learning-based approach for solving complex problems: special application in software engineering for detecting code smells, Neural Comput. Appl. (2019) http://dx.doi.org/10.1007/s00521-019-04175-z.

[SLR41] B.M. Merzah, Software quality prediction using data mining techniques, in: 2019 International Conference on Information and Communications Technology, ICOIACT, 2019, pp. 394–397, http://dx.doi.org/10.1109/ICOIACT46704.2019.8938487.

[SLR42] P. Sharma, E.A. Kaur, Design of testing framework for code smell detection (OOPS) using BFO algorithm, Int. J. Eng. Technol.(UAE) 7 (2.27 Special Issue 27) (2018) 161–166, http://dx.doi.org/10.14419/ijet.v7i2.27.14635.

[SLR43] H. Gupta, L. Kumar, L.B.M. Neti, An empirical framework for code smell prediction using extreme learning machine, in: 2019 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference, IEMECON, 2019, pp. 189–195, http://dx.doi.org/10.1109/IEMECONX.2019.8877082.

[SLR44] Z. Özkalkan, K.S. Aydin, H.Y. Tetik, R. Belen Saglam, Automatic detection of feature envy using machine learning techniques, in: CEUR Workshop Proceedings, Vol. 2201, 2018, http://ceur-ws.org/Vol-2201/UYMS_2018_paper_80.pdf.

[SLR45] D. Kim, Finding bad code smells with neural network models, Int. J. Electr. Comput. Eng. 7 (6) (2017) 3613–3621, http://dx.doi.org/10.11591/ijece.v7i6.pp3613-3621.

[SLR46] A.K. Das, S. Yadav, S. Dhal, Detecting code smells using deep learning, in: TENCON 2019 - 2019 IEEE Region 10 Conference, TENCON, 2019, pp. 2081–2086, http://dx.doi.org/10.1109/TENCON.2019.8929628.