Preprint: Tomasz Lewowski and Lech Madeyski, "Code Smells Detection Using Artificial Intelligence Techniques: A Business-Driven Systematic Review", pp. 285–319. Cham: Springer International Publishing, 2022. DOI: 10.1007/978-3-030-77916-0_12

Preprint: https://madeyski.e-informatyka.pl/download/LewowskiMadeyski22.pdf

Code Smells Detection Using Artificial Intelligence Techniques: A Business-Driven Systematic Review

Tomasz Lewowski and Lech Madeyski

Abstract *Context:* Code smells in the software systems are indications that usually correspond to deeper problems that can negatively influence software quality characteristics. This review is a part of a R&D project aiming to improve the existing *codebeat* platform that help developers to avoid code smells and deliver quality code. *Objective:* This study aims to identify and investigate the current state of the art with respect to: (1) predictors used in prediction models to detect code smells, (2) machine learning/artificial intelligence (ML/AI) methods used in prediction models to detect code smells, (3) code smells analyzed in scientific literature. Our secondary objectives were to identify (4) data sets and projects used in research papers to predict code smells, (5) performance measures used to assess prediction models and (6) improvement ideas with regard to code smell detection using ML/AI. *Method:* We conducted a systematic review using a database search in Scopus and evaluated it using the quasi-gold standard procedure to identify relevant studies. In the data sheet used to obtain data from publications we factor research questions into finer-grained ones, which are then answered on a per-publication basis. Those are then merged over a set of publications using an automated script to obtain answers to the posed research questions. *Results:* We have identified 45 primary studies relevant to the primary objectives of this research. The results show the prediction capability of the ML/AI techniques for predicting code smells. *Conclusion:* Only a few smells—Blob, Feature Envy, Long Method and Data Class—have received the vast majority of interest in research community. The usage of deep learning techniques is increasing. Most researchers still use source code metrics as predictors. Precision, recall and F-measure are the go-to performance metrics. There seems to be a need for modern reference data/projects sets that reflect modern constructs of programming

Tomasz Lewowski
Department of Applied Informatics, Wroclaw University of Science and Technology, Poland,
e-mail: tomasz.lewowski@pwr.edu.pl, ORCID: 0000-0003-4897-1263

Lech Madeyski
Department of Applied Informatics, Wroclaw University of Science and Technology, Poland,
e-mail: lech.madeyski@pwr.edu.pl, ORCID: 0000-0003-3907-3357

languages. We identified various promising paths of research that have the potential to advance the state of the art in the area of code smells prediction.

1 Introduction

Software industry is a huge business with worldwide revenue totaled $407.3 billion in 2013 [13]. According to World Quality Report (2016-2017), in average the industry was spending over 30% of the IT budget on Quality Assurance (QA) and Testing, while the report study participants predicted an upward move to 40% by 2019 [3]. Hence, precise detection of quality issues in code (issues that make the code hard to maintain and evolve, and thus need to be fixed/refactored), is of great importance.

At the end of the previous century, the term "code smells" has been coined by Beck and Fowler [12] in the context of identifying quality issues in code that can be refactored. Since then a lot of researchers investigated the smell metaphor in software engineering describing a wide range of smells that can be detected, techniques that can be used to predict (detect) smells, as well as metrics that can serve as predictors of bad smells.

The aim of this systematic review is to summarize a large body of knowledge in the aforementioned areas. However, it is worth mentioning that this review was conducted as a preliminary step of a research & development (R&D) project funded by NCBiR (POIR.01.01.01-00-0792/16) conducted in the *code quest* software development company[1]. The company develops a platform, called *codebeat*[2], for automated code review for mobile and web, supporting developers in detection of code smells. As the company wants to know what is the state of the art in prediction of code smells using artificial intelligence (AI) in general, and machine learning (ML) in particular, this review is in fact a business driven literature review with the goal to present the state of the art in code smells detection with research questions presented in Section 2.1.

1.1 Related work

This is not the first systematic review of literature regarding code smells. An earlier review by Zhang et al. [30] focused on more meta-research questions: which code smells were researched at the time, what were the aims of studies on code smells, what techniques were used in these studies and whether there is an actual evidence of usefulness of the code smell concept. Their study was focused on the original 22 code smells introduced by Fowler [12], and they discovered that relatively few smells attract most research, and most of the smells were not thoroughly analyzed. Their

[1] `codequest.com`

[2] `codebeat.co`

research included publications published between 2000 and 2009, which means that a lot of recent research is simply not present there.

Singh and Kaur [24] incorporated data up to September 2015. They focused on refactoring of code smells and anti-patterns, but some of the research questions are related strictly to code smell detection. Authors analyzed a wide range of techniques and tools, but their focus is on brief presentation of all tools and techniques (not only automated, but also semi-automated and manual) used for code smell detection rather than on analyzing and comparing performance of ML/AI methods.

Five other systematic studies on code smells were done in the last three years: [23, 22, 2, 4, 1]. In the first one, by Sharma and Spinellis [23], authors cover broad range of code smell-related issues, such as what do they actually represent, how do they get introduced into software systems, what is their effect on processes, artifacts and people and what are their detection methods. Due to the broad scope, the paper only briefly lists categories of smell detection techniques, without going into specific techniques in the category or achieved results. This paper covers publications published from year 1999 to 2016.

A review by Santos et al. [22] covers a slightly different area—it focuses on the „themes" of studies investigating code smells, their experimental settings and convergence of their findings. It covers a total of 65 publications from years 2002 to 2017. The „theme" in the context of this study is kind of a context classification—themes are „Detection", „Programming", „Human aspects", „Correlation with development issues". This study found that there is a multitude of inconsistencies among researchers and, as of now, no known detection and removal techniques are adequate, including human evaluation.

An extensive literature review conducted by Azeem et al. [2] included research questions on used machine learning techniques, independent variables, machine learning algorithms, data sets, evaluation techniques and impact of those factors on final model performance. The paper findings include limited support for code smells detection using machine learning techniques, the fact that current papers focus mostly on using code metrics, problem of code smell intensity is under-researched and the impact of each of these factors cannot be easily determined. The study analyzed papers published between 2000 and 2017.

Caram et al. [4] discussed code smells addressed in the literature, used machine learning techniques and frequency of their usage as well as performance of various techniques. The study analyzed 26 papers published between 1999 and 2016. The conclusion was that all machine learning techniques perform comparably, with Decision Tree, Random Forest, Semi-supervised and Nearest Neighbor having a slightly better overall performance. One of the issues raised by the paper is incomparability of the studies, mostly due to using different data sets.

A more recent study in the area by Al-Shaaby et al. [1] focuses on papers published between 2005 and 2018. 17 studies were deemed relevant and sufficiently precise quality-wise. The study addresses several research questions: used machine learning techniques, code smells that researchers attempt to identify, used performance measures, data sets and tools used for modelling.

None of these studies included any appendix with intermediate results, such as full initial list of considered publications, reasons for rejection for each of them or data elements extracted from each, which would simplify reproduction and improve reviewability.

1.2 Contributions of this study

In response to the above mentioned needs we conducted a systematic review on literature concerning prediction of code smells using ML/AI methods. As a result, the review makes the following contributions to the field:

1. Presents the state-of-the-art in the current code smells detection research including predictors and ML/AI methods used in prediction models, as well as the range of code smells analyzed in the scientific literature.
2. Identifies performance metrics used by researchers today.
3. Identifies data sets or software projects being their origin (including their size and other characteristics) used to create code smell prediction models.
4. Identifies research ideas to advance the domain of code smells prediction on which other researchers and tool vendors may build upon, as well as factors that influence the predictive performance of code smells prediction models.

We present the details of our research methods in Section 2, the results of our systematic review in Section 3, the discussion of the results in Section 4, and we conclude with Section 5.

2 Methods

We performed this systematic review (SR) according to the guidelines by Kitchenham et al [14]. The processes we adopted are specified in this section.

2.1 Research questions

The research questions relating to our review are as follows:

RQ1 Which predictors are used in prediction models to detect code smells?
RQ2 Which ML/AI methods are used in prediction models to detect code smells and which methods are considered the best?
RQ3 Which code smells are analyzed in scientific literature?
RQ4 What data sets and projects, and of what characteristics are used in research papers to predict code smells?
RQ5 Which performance metrics are most commonly used in the literature?

RQ6 What are the ideas, in the existing research, upon which code smell prediction using machine learning may be built?

2.2 Protocol development

Initially a protocol was created to define the procedures we intended to use for the systematic review including the search process, the primary study selection process, the data extraction process and the data analysis process. It also identifies the main tasks of all the co-authors. The protocol was initially drafted by the second author and double-checked by the first author.

The following sections are based on the processes defined in the protocol. Any divergences report our actual processes, as opposed to the planned processes are described in the protocol.

2.3 Search process

We intend to search for papers that will help us to answer research questions posed in Section 2.1 by following our search strategy described in Section 2.3.1.

2.3.1 Search Strategy

Our main search process will be an automated search using Scopus because of its wide coverage. From the point of view of the research project we are involved in and the *code quest* company, finding all of the relevant papers is not critical, but to be on the safe side we will validate the Scopus search using a quasi-gold standard [29, 14] performing a manual search across a limited set of topic-specific journals and conference proceedings over a restricted time period (year 2017). The results of this process are reported in subsequent sections.

2.3.2 Search strings

We derived major terms from research questions, identified alternative spellings or synonyms for major terms using OR, used AND to connect the major terms, checked the search terms in relevant publications we already had, and followed the rules to construct search strings in Scopus[3].

As a result, our initial search string in Scopus was as follows:

[3] More details can be found at `https://service.elsevier.com/app/answers/detail/a_id/11213/supporthub/scopus/#tips` and `https://service.elsevier.com/app/answers/detail/a_id/11236/kw/all%20fields/supporthub/scopus/`

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

which translates into the following URL:

`https://www.scopus.com/results/results.uri?sort=plf-f&src=s&sid=a9ac162c765cdd97d420c650614d365f&sot=a&sdt=a&sl=155&s=TITLE-ABS-KEY+%28+%28+%22code+smell%22+OR+%22bad+smell%22+OR+antipattern+OR+anti-pattern+OR+%22anti+pattern%22+%29+AND+%28%22machine+learning%22+OR+predict*+%29+%29+AND+PUBYEAR+%3e+1998&origin=searchadvanced&editSaveSearch=&txGid=272750ded430629599c4d74aae98f0e3`

It is worth mentioning that Beck coined the term "code smell" in the context of identifying quality issues in code that can be refactored to improve the maintainability of a software in 1999 [12]. Hence, the time period to be covered by the review is limited by `PUBYEAR > 1998` in the search string.

Madeyski performed the search in Scopus on February 21, 2018. In total, 88 papers were returned from Scopus. All of the results were saved in BibTeX (and CSV) format.

After analysis of the aforementioned preliminary set of 88 papers necessary corrections to our 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`

while URL was:

`https://www.scopus.com/results/results.uri?sort=plf-f&src=s&sid=9d6009998ae2e22265826addfe46ebd6&sot=a&sdt=a&sl=210&s=TITLE-ABS-KEY+%28+%28+%22code+smell%22+OR+%22bad+smell%22+OR+antipattern+OR+anti-pattern+OR+%22anti+pattern%22+%29+AND+%28+%22machine+learning%22+OR+predict*+OR+%7bdetect%7d+OR+%7bdetection%7d+OR+heuristic*+%29+AND+software+%29+AND+PUBYEAR+%3e+1998&origin=searchadvanced&editSaveSearch=&txGid=70e3ad5814b99ef9e407ff18c23a8c07`

Madeyski performed the final search in Scopus on March 21, 2018. In total, 424 papers were returned from Scopus. All of the results were saved in BibTeX (and CSV) format for further analysis. Due to the fact that work on this paper tool substantial amount of time, the same search was re-run by Lewowski on June 05, 2020. This search yield a total of 607 papers.

It is worth mentioning that now our search string includes now not only terms “predict” and “machine learning”, but also “detect” and “heuristic*”. Especially the word “detect” was sometimes used in the abstracts of interesting papers without the word “predict”.

Checking the accuracy of the search string, Madeyski found that an important review paper by Sharma and Spinellis was missing in Scopus for unknown reason (other papers in JSS journal vol. 138 are indexed). Fortunately, the paper was indexed in Scopus later.

Our search procedure may be summarized as follows:

- Identify a tentative set of papers via automated search in Scopus.
- Evaluate the papers for inclusions and exclusions.
- Validate (and perhaps correct) the search strategy.

Initially, we also planned to perform the snowballing procedure using both backward and forward snowballing as described by Wohlin [28], but the number of found, read and included primary studies was so large that we decided to stick to the already identified and accepted set of papers if it passes the validation step described in the next section.

2.3.3 Validating the search strategy

Two key criteria for assessing the automated search are *recall* (also termed sensitivity) and *precision* [7, 29] that can be calculated as follows:

$$Recall = \frac{R_{found}}{R_{total}} \tag{1}$$

$$Precision = \frac{R_{found}}{N_{total}} \tag{2}$$

where:
R_{total} is the total number of relevant studies
N_{total} is the total number of studies found
R_{found} is the number of relevant studies found

Unfortunately, the practical problem in calculating recall is that R_{total} is not known. Hence, our strategy to validate the search process will be through the construction of a "quasi-gold standard" (QGS). We will incorporate the QGS concept, which consists of collection of known studies, and corresponding "quasi-sensitivity" into the search process for evaluating search performance as described by Zhang et al [29]. The quasi-gold standard will be determined by performing a manual search across a limited set of topic-specific journals and conference proceedings over a restricted time period (year 2017). The approach has been originally evaluated through two participant-observer case studies with promising results. QGS will be applied only to publications relevant to primary RQs (1-4) and publications relevant only to secondary RQs(5-6) will not be included.

Our validation of the search strategy will be conducted in the following steps, similar to what was recommended by Zhang et al [29]:

Step 1A Determine initial search string using domain knowledge and experience.
Step 1B Identify relevant journals and conferences.
Step 2 Establish quasi-gold standard using manual search
Step 3* Revise search string using QGS results[4]

[4] Apply only if recall does not reach required threshold.

The review team screen all papers in the selected sources and apply the inclusion and exclusion criteria, defined in advance. Screening can be applied initially to the title and abstract of a paper (in Phase 1) or to the whole paper (in Phase 2).

Step 4 Conduct automated search

Step 5 Evaluate search performance
If we do not reach quasi-sensitivity threshold of 75% then we go to Step 3 and revise search strings[5].

The workflow of the validation of the search strategy is presented in Figure 2.3.3. The workflow diverges from what was proposed by Zhang et al. [29] in that initial

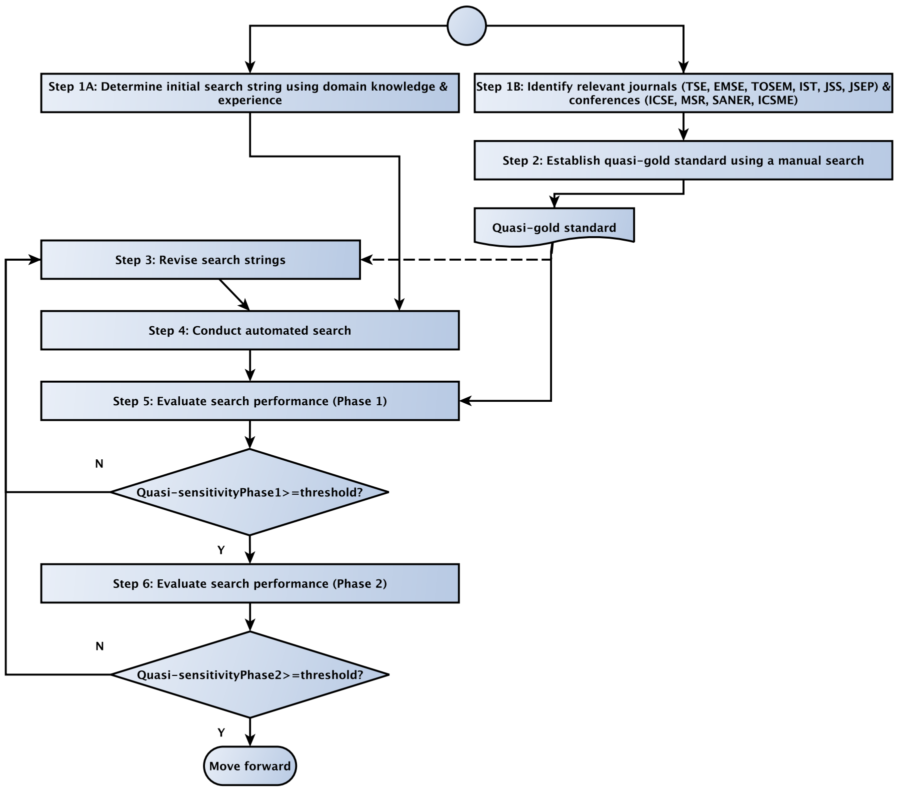
Step 1A: Determine initial search string using domain knowledge & experience
Step 1B: Identify relevant journals (TSE, EMSE, TOSEM, IST, JSS, JSEP) & conferences (ICSE, MSR, SANER, ICSME)
Step 2: Establish quasi-gold standard using a manual search
Quasi-gold standard
Step 3: Revise search strings
Step 4: Conduct automated search
Step 5: Evaluate search performance (Phase 1)
N
Quasi-sensitivityPhase1>=threshold?
Y
Step 6: Evaluate search performance (Phase 2)
N
Quasi-sensitivityPhase2>=threshold?
Y
Move forward

Fig. 1 Workflow of the systematic search process (inspired by [29])

search string is not determined using QGS results but using authors' domain knowledge and experience. There are two causes for this divergence (taking advantage of the fact that there is more than one author of the review):

[5] Zhang et al. [29] suggest that a sensitivity (recall) threshold (i.e., a completeness target) of between 70% and 80% might be used to decide whether to go to Step 3 (and to refine the search terms) or whether to proceed to the next stage of the review.

1. Independence of initial search string formulation and selection of venues for QGS enhances reliability of the validation procedure by removing coupling between the search string and the validation data set.
2. Lack of the aforementioned dependency enables parallel work on steps 2 and 4, thus making the process shorter and delivering business value earlier. This allows stakeholders to review their expectations (for example by refining/adding/removing research questions or providing additional guidance), which in turn increases business relevance of the study, even at the cost of additional search string revision.

The results of the automated search are compared to the results of the manual search (the quasi-gold standard) and quasi-sensitivity ($Recall = \frac{R_{found}}{R_{total}}$), as well as quasi-precision ($Precision = \frac{R_{found}}{N_{total}}$) can be calculated, as on a basis of:

- R_{found} is the number of relevant studies found by the automated search (Step 4) that are published in the venues used in Step 2 (the manual search) during the time period covered by the manual search.
- R_{total} (the total number of relevant studies for the selected venues and time period) is the number of relevant papers found by the manual search (Step 2).
- N_{total} is the total number of papers found by the automated search (Step 4).

Validation of our search strategy requires (in Step 1B) to identify relevant journals and conferences[6]. We decided to search for papers published in 2017 in the following top software engineering journals:

- IEEE Transactions on Software Engineering (TSE)
- Empirical Software Engineering (EMSE)
- ACM Transactions on Software Engineering Methodology (TOSEM)
- Information and Software Technology (IST)
- Journal of Systems and Software (JSS)
- Journal of Software: Evolution and Process (JSEP)

and conference proceedings:

- International Conference on Software Engineering (ICSE)
- Mining Software Repositories (MSR)
- IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)
- International Conference on Software Maintenance and Evolution (ICSME)

Publications from a venue were extracted using Scopus on April 23, 2018 using search strings presented in Table 1. Number of extracted publications and established QGS publications are presented in Table 2.

It is worth mentioning that limiting PUBYEAR in Scopus to 2017 does not exclude papers accepted in this year, but still waiting for assigning to a specific issue

[6] We focus on the main full papers research tracks of the conferences, and do not cover collocated conferences or workshops.

Table 1 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") AND PUBYEAR = 2017
TSE	SRCTITLE ("ieee transactions on software engineering") AND PUBYEAR = 2017
EMSE	SRCTITLE ("Empirical Software Engineering") AND PUBYEAR = 2017 AND (LIMIT-TO (EXACTSRCTITLE , "Empirical Software Engineering "))
IST	SRCTITLE ("Information and Software Technology") AND PUBYEAR = 2017
JSS	SRCTITLE ("Journal of Systems and Software") AND PUBYEAR = 2017
JSEP	SRCTITLE ("Journal of Software Evolution and Process") AND PUBYEAR = 2017
ICSE	SRCTITLE ("International Conference on Software Engineering") AND PUBYEAR = 2017 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") AND PUBYEAR = 2017
SANER	SRCTITLE ("IEEE International Conference on Software Analysis, Evolution and Reengineering") AND PUBYEAR = 2017
ICSME	SRCTITLE ("International Conference on Software Maintenance and Evolution") AND PUBYEAR = 2017

Table 2 Results of QGS search and selection

Title	Total	# of publications accepted in Phase 1	# of publications accepted in Phase 2	References
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	1	[SLR36]

(so called articles in press). This means that more papers might have been analyzed (apart from papers published in 2017, we analyzed also some papers that will be published in 2018 and maybe even later), but does not affect the relevance of the procedure.

Established QGS contains only a single publication. To verify why, we have investigated at which venues were the accepted papers published. Their DOIs and venue names are included in Table 3.

Search Process Task Allocation: Madeyski prepared an initial search string and performed initial search in Scopus which returned 88 results. He verified if any known papers are missing, refined the search string, and performed a refined search in Scopus which returned 424 results. He prepared the search evaluation strategy on a basis of quasi-gold standard (QGS). Lewowski performed the manual search required by QGS.

Table 3 Publications published in 2017 accepted for analysis

DOI	Venue name
doi:10.1109/ICSME.2016.26	ICSME 2016 - Proceedings of the 2016 IEEE International Conference on Software Maintenance and Evolution
doi:10.1109/ASE.2017.8115667	ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering
doi:10.1016/j.knosys.2017.04.014	Knowledge-Based Systems
doi:10.1109/MOBILESoft.2017.29	MOBILESoft 2017 - Proceedings of the 2017 IEEE/ACM 4th Int. Conference on Mobile Software Engineering and Systems
doi:10.1007/s11219-016-9309-7	Software Quality Journal
doi:10.5220/0006338804740482	ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems

2.4 Primary Study Selection Process

Primary studies were filtered using a two-phase approach: in the first phase (screening) the paper's title and abstract were verified with a checklist, and if the paper passed this phase, the second one was data acquisition from full text. Since the screening was done using only abstract, some papers were rejected during second phase—for example if their abstract suggested that the paper used machine learning, but in fact it did not.

2.4.1 Inclusion and exclusion criteria

The inclusion criteria for papers are defined as follows:

- The paper reports the use of ML/AI prediction models
- The paper is related to code smell detection

The exclusion criteria:

- The paper was an editorial, abstract, presentation slides, is not peer-reviewed or is not an article or chapter of a book or conference proceedings.
- The paper was written in a language other than English.
- Full text of the paper was not available.
- The paper was published before 1999 when Fowler at al [12] first introduced the concept of code smells.
- The paper did not attempt to use ML/AI for code smell prediction/detection in the context of text-based object-oriented, functional or procedural programming languages or a language being a mix of some or all of them (e.g., Scala is an object-oriented language, but has many features of functional programming languages).
- The same or extended results were already published (only extended results are then included in the study).

- The same authors published a study with the same title in conference proceedings (or book chapter), as well as in a journal (only journal paper, which typically is more thorough, was included).

Phase 1 of selection process was performed using a checklist that contained inclusion and exclusion criteria, split into simple Yes/No statements that can be answered basing only on publication title and abstract. The checklist is designed in such a way that answering "No" results in rejection of the publication (i.e., all checks are represented as inclusion criteria).

Applied checklist contained the following statements:

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 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.

Results after Phase 1 of selection (on the basis of abstracts) are as follows:

- Number of publications: 607.
- Number of relevant publications: 164.
- Precision: 27.0%.
- Number of relevant publications found in QGS, but not found in automated search: 0
- Number of relevant publications found in both QGS and automated search: 5 [20, 5, 10, 18],[SLR36].
- Recall: 100%, i.e., above the assumed threshold 75%.

Results after Phase 2 of selection (on the basis of full texts) are as follows:

- Number of publications: 164.
- Number of relevant publications: 44.
- Precision: 26.8%.
- Number of relevant publications found in QGS, but not found in automated search: 0
- Number of relevant publications found in both QGS and automated search: 1 [SLR36].
- Recall: 100%, i.e., above the assumed threshold 75%.

We were aware of one more publication relevant to the study which was not present in search results, therefore we included the paper by Grodzicka et al [SLR12] to the initial publication set manually. It has gone through the regular checklist and data acquisition phases.

Final list of accepted studies is listed in the end of the publication. References for Systematic Literature Review are prefixed with "SLR".

Task allocation during selection process:

1. Lewowski applied the inclusion and exclusion criteria to the identified studies.
2. Madeyski checked the application of the inclusion/exclusion criteria on randomly selected papers.

Disagreements were resolved by discussion. Agreement rate for results of first phase was 95% (disagreement on [21] - final decision: reject), for results of second phase - 80% (disagreement on [26] - final decision: reject and [SLR31] - final decision: accept).

2.5 Assessing study quality

Quality assessment is about determining the extent to which the results of an empirical study are valid and free from bias. We applied the quality checklist inspired by Dybå and Dingsøyr [8] which, as mentioned by Kitchenham et al. [14], has an advantage that can be used across multiple study types. The same checklist inspired other researchers performing systematic reviews of machine learning techniques in other areas as well [27, 16]. Each question has only three possible answers: "Yes", "Partly", or "No" and these three answers are scored in the following way: "Yes" = 1, "Partly" = 0.5, and "No" = 0 inspired by [27, 16]. The final score is obtained after adding the values assigned to each question. A study could have maximum score of 11 and minimum score of 0. The criteria we take into account are as follows:

1. 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

2. Is there an adequate description of the context in which the research was carried out (analyzed projects, data sets, data collection procedure etc.)?

Yes	paper contains detailed information on analyzed data sets (including version tags or VCS revisions) or valid references to data sets, detailed procedure of data collection (or data collection script) and reproducible analysis procedure (or runnable script)
Partly	paper contains information on analyzed data sets (without version tags or VCS revisions), approximate procedure of data collection, stated analysis goals
No	only rough information on analyzed data sets (e.g. number of data sets), no or rough information on data collection procedure

3. Are the independent variables (predictors) and dependent variable(s) clearly defined?

Yes	paper contains detailed description of all predictors and dependent variables. If predictors are obtained by running a tool, tool version is given (if required, parameters used for running the tool are given as well)
Partly	paper contains rough description of used predictors or reference to them. If predictors are obtained by running a tool, tool version and used parameters are unknown (but the tool is accessible)
No	paper contains only basic information on predictors, like their number and names (if names are not well-known). There is no possibility to acquire information about predictors from used tool (for example because the tool is not clearly mentioned or is not accessible).

4. Are the predictive modelling techniques clearly defined?

Yes	paper either refers explicitly to a specific technique (by citing a paper that describes details) or describes in detail used modelling technique (for both prediction model and data model, if applicable)
Partly	paper either refers to a well-known technique (e.g. "genetic algorithm" or "decision tree") or describes general concepts of models (both prediction model and data model, if applicable)
No	paper either does not describe modelling techniques or uses only general terms such as "neural network" or "population-based algorithm"

5. 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

6. 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

7. 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

8. Is the research reproducible (is a package including data sets and code or a detailed description available)?

Yes	complete information required to reproduce the study is available
Partly	some of the information is missing (e.g. code version, data set version, script parameters)
No	paper lacks most of these elements

9. 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

10. Are the findings of 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)

11. 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

Results of assessing study quality are presented in Table 4 and Figure 2.

Table 4 Statistics for quality assessment scores

Statistic name	Statistic value
Max score	10.5
Average score	6.23
Min score	0.0
Total number of publications	44

Quality scores will help us during the interpretation of the findings of a review, but our observation is that the quality scores do not always correlate with the importance of the research ideas, to advance the domain, on which other researchers and tool vendors may build upon while developing code smells prediction tools. For example, [SLR36] scored only 4 points in quality assessment. However, this was a space-constrained conference paper, and it introduced concepts of smell detection via history mining and via text similarity analysis. To avoid rejection of such interesting papers, we have decided not to reject any paper solely on the basis of quality assessment score. All of the low-scoring papers are either short conference papers or workshop notes that usually lack details related to limited descriptions of threats to validity, reproducibility, and lack of comparison to the baseline, likely due to papers' length limitations.

2.6 Data extraction

Data extraction form was prepared in Google Sheets to streamline, as much as possible, data synthesis steps, as well as progress monitoring.

Selected fields of the form are presented below:

- DOI
- Authors
- Title
- Assessing study quality (each question has only three possible answers: "Yes", "Partly", or "No" and these three answers are scored in the following way: "Yes" = 1, "Partly" = 0.5, and "No" = 0):

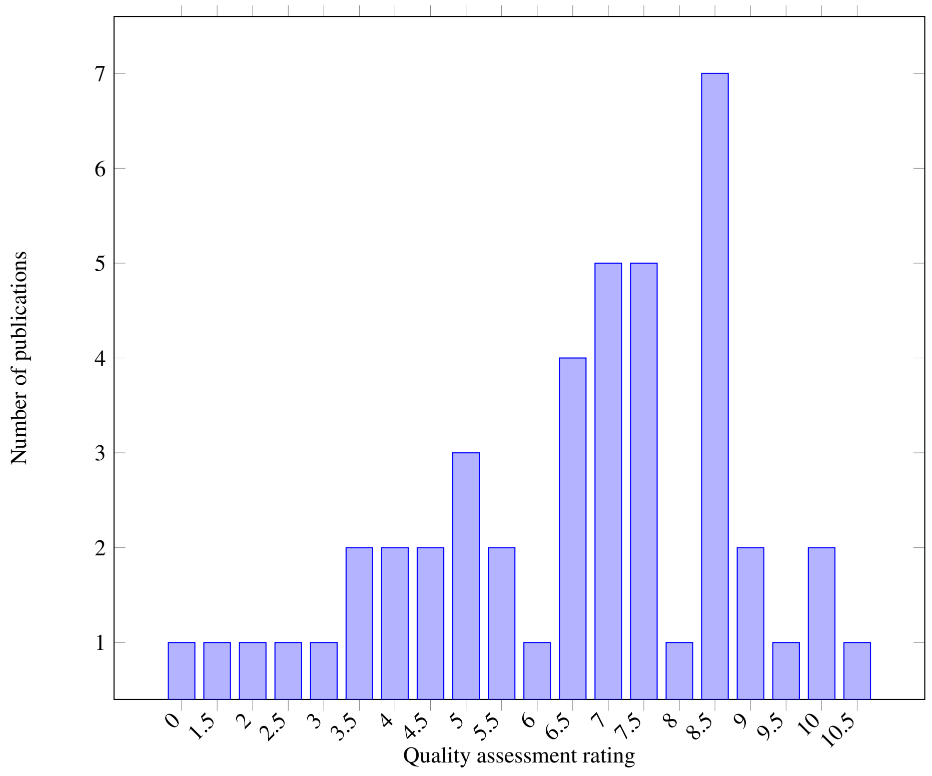

Fig. 2 Number of publications scored given amount of points in quality assessment rating

- Are the aims of the research clearly defined?
- Is there an adequate description of the context in which the research was carried out (analyzed projects, data sets, data collection procedure etc.)?
- Are the independent variables (predictors) and dependent variable(s) clearly defined?
- Are the predictive modelling techniques clearly defined?
- Are the performance measures used to assess the models clearly defined?
- Are the performance measures used to assess the models considered credible?
- Are the limitations or threats to validity of the study specified?
- Is the research reproducible (is a package including data sets and code or a detailed description available)?
- Is the proposed method or methods compared with other methods and/or baselines?
- Are the findings of study clearly stated and supported by reported results?
- Does the study provide convincing arguments about additional value given to academia or industry community?

- PQ1: Which code smells are analyzed in the paper?
- PQ2: Which predictors are used for each of those code smells?
- PQ3: Which ML/AI methods are used for detection of the smells?
- PQ4: Which data sets were used in the study?
- PQ5: What are the reported performance measures of prediction models?
- PQ6: What novel concept/technique is introduced by the study?

We decided to include in the workbook references to page numbers while performing data extraction of every important and hard to find again later chunk of information.

Data Extraction Process Task Allocation:

1. Lewowski undertook all the extractions, which was held in Google Sheets.
2. Madeyski independently checked the extraction for randomly selected papers.
3. Disagreements were resolved by discussion. There were two disagreements, that resulted in adjustments of collected data.

2.7 Data Synthesis and Aggregation Process

The basic objective while synthesizing data is to accumulate and combine data and figures from the selected primary studies in order to formulate a response to the posed research questions. In order to answer the research questions we used visualization techniques such as bar charts and box plots. We also used tables for summarizing and presenting the results combined with narrative synthesis.

During data preprocessing, smells with similar definitions will be merged into one, details of execution of machine learning algorithms (such as whether boosting was used or which parameters were configured) will be erased and project names and versions will be adjusted to common scheme. If there are multiple similar machine learning techniques (for example C4.5, J48 and generic Decision Tree), they will also be merged into single category. Preprocessed data will be stored in a separate file in provided data set, so that access to raw data will not be lost. Preprocessing is performed manually.

Data used in the study as well as reproduction scripts are published on Zenodo: `https://doi.org/10.5281/zenodo.4783264`.

3 Results

Figure 3 presents number of publications relevant to this study published in a given year. A rise in interest is visible since 2009, probably due to increased interest in machine learning techniques.

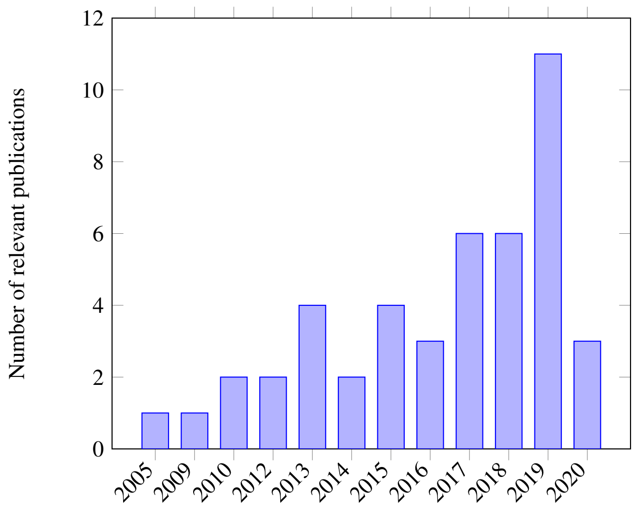

Fig. 3 Number of relevant publications per year

3.1 RQ1: Which predictors are used in prediction models to detect code smells?

While the concept of using product metrics is shared between most publications, predictor sets are hardly shared between any of them - most often used predictor set is used by three publications (not counting source code as input), therefore it is not possible to establish whether performance differences are related to usage of different predictors or to other differences between studies (e.g., different data sets).

Even if publications claim to use same predictor set, it is hard to guarantee that they actually do that—since software metrics are calculated automatically by some tool, even if the description of a metric is given, it is not sufficient for full reproducibility. This is because various tools, or even different versions of same tool, may have operational-level differences or defects in calculation routines, which will result in no possibility to reproduce the study. While some studies utilize high amount of metrics [SLR8], the same studies claim that best prediction model used only few of them.

Table 5 presents aggregated view on types of predictors used in the studies. Some publications ([SLR27, SLR36]) use more than one type of predictor, so the numbers in Table 5 do not add up to the number of studies.

Table 5 Number of publications containing reference to given predictors

Type of predictor	Number of publications
Code metrics	30
Vectorized source	3
Textual similarity	2
Code history	2
Other	6
Unknown	5

3.2 RQ2: Which ML/AI methods are used in prediction models to detect code smells?

We started with dividing used methods into the following categories:

Trees	C4.5, C5.0, J48, unnamed decision trees
Rules	JRip, Association rules
Genetic Programming	Genetic Programming, Multi-Objective Genetic Programming
Neural Networks	MLP, perceptrones, Voted perceptrones
Deep Neural Networks	Autoencoders, Convolutional Networks
Statistical	Bayesian Belief Networks, Naive Bayes
Population	Genetic Algorithms, PSO, BFO, Evolutionary Algorithms, SPOA
Regression	Logistic regression
Random Forests	Random forests
SVM	Support Vector Machine (with any kernel), SMO
Other	FP-growth, Decision tables, KNN, Clustering, Similarity measures, Tuning machine

Trees and SVMs (used in 12 and 11 studies respectively) are the most commonly used machine learning techniques, with statistical methods and Random Forests (both in 9 studies) closely following. A recent rise in deep learning techniques, used in 7 studies, is also visible. Numbers of publications are presented in Table 6.

3.3 RQ3: Which code smells are analyzed in scientific literature?

By far the most researched smell is Blob, a class-level smell, present in 32 publications. Blob category contains several smells (like Large Class, Big Class and God Class), for which boundaries are not defined well enough to justify separation. It is relatively well-defined, as a class which is bigger and/or more complex than it should be.

Next come Feature Envy with 26 analyzing publications and Long Method with 20 publications. Of those Long Method is well defined and fairly well understood,

Table 6 Number of publications containing reference to given ML methods

ML/AI method type	Number of publications
Trees	12
Support Vector Machine	11
Random Forests	9
Statistical	9
Population	8
Deep Neural Networks	7
Rules	6
Neural Network	5
Genetic Programming	3
Regression	3
Other	9

while the understanding of Feature Envy seems to be more vague—sometimes it is attributed to class, sometimes to method. Approaches to detect Feature Envy are also more diverse than those used to detect Long Method.

Further researched smells are Data Class (14 publications), Spaghetti Code (8 publications), Functional Decomposition (8 publications), Shotgun Surgery (5 publications), Lazy Class (5 publications) and Long Parameter List (5 publications).

In total there were references to 59 different smells. All smells that were researched in more than one publication are listed in Table 7. A substantial number of smells is only referred to by a single publication, probably because they were not formalized earlier. This applies particularly to domain-specific smells, such as Android smells (like UI Overdraw) or Web Service smells (like Chatty Web Service).

Table 7 Number of publications containing reference to given code smells

Smell	Number of publications
Blob	32
Feature Envy	26
Long Method	20
Data Class	14
Spaghetti Code	8
Functional Decomposition	8
Shotgun Surgery	5
Lazy Class	5
Long Parameter List	5
Divergent Change	3
Swiss Army Knife	3
Misplaced Class	3
Duplicated Code	2
Leaking Inner Class	2
Member Ignoring Method	2
Parallel Inheritance	2
Promiscuous Package	2

3.4 RQ4: What datasets and projects, and of what sizes are used in research papers to predict code smells?

According to gathered data, there is no single data set that is shared and universally accepted. Most authors provide rough information regarding used projects—for example their names and versions ([SLR27, SLR41]) or their general characteristics (e.g. [SLR34, SLR39]).

The most common annotated data set is the one published by Fontana et al. [SLR8], and it is used in 8 studies. Another code smell data set, Landfill ([19]), is only used in a single paper, by Hadj-Kacem and Bouassida [SLR16]. Finally, data set from the work of Di Nucci et al [6] is used in a single study by Guggulothu and Moiz [SLR13]), but in a modified version. Two studies give no details on used data sets.

Out of projects, the one most often encountered was Xerces, used in 12 of the studies. Out of the rest only a few were used in more than two studies—these include Azureus, ArgoUML, Ant, Gantt Project, Log4j and JFreeChart. All these projects are outdated as of today—used versions were released over ten years ago (for example, Azureus 2.3.0.6 was released on 4th February 2006, Xerces:2.7.0 on 24 June 2005 and Nutch:1.1 no 7 June 2010. We were able to obtain release dates and basic size statistics for 30 open source projects and present those in Table 8. Those projects contain between 22 and 4844 Java source files with an average of 724 and between 3374 and 325301 LoC with an average of 81912 (as calculated by CLOC 1.72[7]). Release dates range between 17/05/2002 and 25/06/2014, with a median of 04/09/2009. Some studies used Qualitas Corpus [25] as the source of code smells. It contains sligthly older set of projects—starting from 10/07/2002, ending on 15/12/2011—but in similar size range (between single thousands and hundred of thousands of lines).

3.5 RQ5: Which performance metrics are most commonly used in the literature?

Two most commonly used performance metrics are precision (in 29 papers) and recall (26 papers). They are often accompanied by F-measure (in 17 papers). In earlier papers accuracy (11 papers) was fairly often used.

Seven publications use area under receiving operating characteristic (AUROC) as the performance metric, while three use Matthews Correlation Coefficient (MCC). Three papers do not use any performance metric. Only two papers report full confusion matrix.

Several papers use other performance metrics, usually strictly related to the way the research was performed—for example Kessentini and Ouni in [SLR22] use relevance, understood as count of algorithm recommendations that were accepted

[7] `https://github.com/AlDanial/cloc`

Table 8 Basic information about projects used as sources for data sets

Project name	URL	Release date	# of files	# of Java LoC
Apache Ant:1.5.2	https://github.com/apache/ant	13/01/2016	932	91132
Apache Ant:1.7.0	https://github.com/apache/ant	13/01/2016	1194	123697
ArgoUML:0.26	https://argouml-tigris-org.github.io/	27/09/2008	1752	186425
ArgoUML:0.30	https://argouml-tigris-org.github.io/	11/02/2010	2210	200662
ArgoUML:0.34	https://argouml-tigris-org.github.io/	15/12/2011	1922	195670
AspectJ:1.5.3	https://github.com/eclipse/org.aspectj	22/11/2006	4844	325301
Class Editor:2.23	http://classeditor.sourceforge.net	21/03/2004	66	10027
DavMail:4.5.1	https://github.com/mguessan/davmail	20/06/2014	181	29696
DirBuster:1.0	https://sourceforge.net/projects/dirbuster/	27/02/2009	75	12928
FormLayoutMaker:8.2.1rc	https://sourceforge.net/projects/formlayoutmaker/	26/03/2006	22	4239
HSQLDB:2.2.9	https://sourceforge.net/projects/hsqldb/	06/08/2012	529	164026
Java3D Modeler:1.3.5	https://sourceforge.net/projects/java3dmodeler/	24/07/2012	78	9105
jEdit:4.5pre1	https://sourceforge.net/projects/jedit/	19/11/2011	554	110869
JFreeChart:1.0.13	https://sourceforge.net/projects/jfreechart/	20/04/2009	989	143062
JFreeChart:1.0.14	https://sourceforge.net/projects/jfreechart/	20/11/2011	1005	146966
JFreeChart:1.0.9	https://sourceforge.net/projects/jfreechart/	04/01/2008	920	128209
JFtp:1.53	https://sourceforge.net/projects/j-ftp/	07/11/2010	133	23808
JHotDraw:6.1	https://sourceforge.net/projects/jhotdraw/	07/10/2004	484	28399
JPropsEdit:1.0.2	https://sourceforge.net/projects/jpropsedit/	22/07/2003	47	3374
Log4j:1.2.1	https://github.com/apache/log4j	17/05/2002	283	23363
Lucene:1.4.3	https://github.com/apache/lucene-solr	26/11/2004	244	25472
nTorrent:0.5.1	https://code.google.com/archive/p/ntorrent/	28/11/2009	377	36286
Nutch:1.1	https://github.com/apache/nutch	26/06/2010	447	45357
outliner:1.8.10.6	https://sourceforge.net/projects/outliner/	04/06/2004	418	35404
PDF Split and Merge: 2.2.4	https://sourceforge.net/projects/pdfsam/	25/06/2014	303	26717
pdfsam:2.2.1	https://sourceforge.net/projects/pdfsam/	24/11/2010	299	26058
Rhino:1.6	https://github.com/mozilla/rhino	23/07/2007	175	58303
Rhino:1.7R1	https://github.com/mozilla/rhino	25/04/2011	329	78197
Tyrant:0.334	https://sourceforge.net/projects/tyrant/	12/06/2005	179	41331
Xerces:2.7.0	https://github.com/apache/xerces2-j	24/06/2005	740	123275

by developers while Kaur et al in [SLR21] use the ratio of defects detected (in this research not every smell represents a defect) to defects in the source code.

Table 9 Number of publications using given performance metric

Performance metric	Number of publications
Precision	29
Recall	26
F-measure	17
Accuracy	11
AuROC	7
MCC	3

3.6 RQ6: What are the ideas, in the existing research, upon which code smell prediction using machine learning may be built?

The following directions of future research and development seem to be promising in the light of the performed review:

1. Text analysis, process metrics may add new useful predictors not correlated with classic ones (e.g., product based metrics) to the tool. Unlikely that this information is used by any of the tools on the market. Fusion of the classic metrics and new ones may lead to interesting results.
2. Search-based software engineering methods (e.g., multi-objective optimization algorithms using genetic programming [SLR23, SLR31, SLR22, SLR4]) may be combined with classic ML methods as well to improve the results even further.
3. Some studies, e.g. the one by Hozano et al [SLR18], analyze level of agreement between developers on the same set of code smells. The study yields 0.222 (Feature Envy) - 0.421 (Data Class) inter-rater agreement measured by κ, which is a measure to evaluate the concordance or agreement among multiple raters described by Fleiss [9], as to whether a given structure is or is not a smell. It is important to take this into account when setting goals for and evaluating code smell prediction tool(s), despite the fact that some scientific publications reported over 95% accuracy or F-measure in detecting code smells when prediction models were trained on data sets produced by a small group of people with similar background and experience (e.g., a small group of MSc students attending same preparation lectures). Tool vendors aiming to serve a wide range of developers with different background and skill sets may expect low inter-rater agreement. An interesting path of further R&D activities seems to be customization of code smell prediction models to specific projects.
4. Quite a lot of research (and thus one may expect code smells detection tools development) was performed using really old versions of software projects (e.g., webmail-0.7.10 released in 2002, being a part of QualitasCorpus), often using very old versions of Java (e.g., Java 5 released in 2004), see Table 8. A promising path of future research would be to take into account how long way made programming languages like Java, which now include, e.g., closures, streams, varargs, type inference for local variables, generics, enumerations, annotations, foreach loop, static imports and vast changes in standard libraries (introduction of immutable data types, improved concurrency, database access, IO and lot of others). Furthermore, very few projects were used in more than three studies (these include: Xerces, Gantt Project, Apache Ant, JFreeChart and Azureus). It would be important to create modern version of reference data/projects sets that would reflect modern constructs of programming languages (one such attempt was done by Grodzicka et al [SLR12]) instead of applying contemporary ML/AI techniques to old projects, which may not reflect fully how software is developed nowadays, and thus how code smells may look like when new language constructs are employed. This need for a benchmark data set is also supported by the fact that results vary greatly between publications, which is likely to be caused by

different training data—most of the publications do not publish data, thus easy replication is not possible.

5. Number of predictors (metrics) used in some papers is large (e.g.in the paper by Fontana et al [SLR8]), but even in such papers it was possible to extract the most important predictors (e.g., used to extract rules). In further research, it would make sense to focus on rather small number of important predictors to avoid overfitting of the models, which otherwise would overemphasize patterns that are not reproducible.
6. Another direction of further research could be focused on applying ML/AI methods to detect code smells which were not covered by any of the reviewed papers, see the list in the beginning of Section 4.
7. Publications generally use precision and recall as performance metrics. High precision is critical from the business point of view, while high recall is only nice-to-have (cost of smell detected later on is generally lower than cost of analyzing false positives). That said, valuable performance measures according to which prediction models should be evaluated are measures which take into account all of the four quadrants of the confusion matrix (e.g., MCC). Otherwise performance measure could be misleading. Hence, an important path of further research and development would be to evaluate models using better performance measures.
8. Data acquisition is generally a resource-consuming task. While it likely cannot be automated (since this would equal automated code smell detection and no machine learning would be necessary), it may be reasonable to use some kind of advisors. Advisors proposed in literature vary from regular code smell detection tools (as in the work by Fontana et al [SLR8]) up verification of effects of a potential refactoring (as in the work by Liu et al [SLR29])

Combining directions 3, 1 and 2 with observation that the work by Hozano et al [SLR18] was inspired by Fontana et al [SLR8] and thus used product metrics as predictors, we pose a hypothesis that using only product metrics may yield good results for homogeneous groups of developers producing training data (for example, accuracy was over 95% in the results obtained by Fontana et al [SLR8]), but much worse for groups of developers with more divergent backgrounds, as described by Hozano et al [SLR18] (mean accuracy 43.7-63%). In subject literature there are the following promising inspirations to embrace that may not be yet well-explored by code smells detection tool vendors:

- process metrics, such as code change history,
- lexical analysis, such as similarities between fragments of code,
- deep learning, which includes analysis of source as token stream,
- search-based methods.

4 Discussion

In this section we address our research questions, discuss our results and their implications.

Table 7 shows that most research that applies ML/AI techniques to code smell detection focuses on the original smells by Fowler et al. [12] (with an exception for Blob, which is present in the original list as “Large Class”). The only smell from top 5 most often researched that did not appear in the original list is “Spaghetti Code”. While other researchers attempt to extend the smell list with new ones, for example Kessentini and Ouni [SLR22] introduced smells dedicated to mobile development, apparently these attempts did not yet cause a major change in perception of code smells and did not make it yet to the mainstream of the discussed domain.

What we did not find, but expected to find in our review were publications addressing some of the code smells originally defined by Fowler et al. [12]: Data Clumps, Switch Statements, Middle Man, Alternative Classes with Different Interfaces, Incomplete Library Class. Hence, almost one fourth of code smells defined by Fowler was not considered in any of the analyzed papers.

4.1 Threats to Validity

It is important to assess the threats to validity (e.g., construct, internal, external), particularly constraints on the search process and deviations from the standard practice.

4.1.1 Internal validity

Internal validity concerns the process of performing the study. We exclude threats related to study reproducibility, as these are explained in detail in Section 4.1.4. An important threat arises from data preprocessing layer—since we merged some of the categories (e.g., stripped parametrization from all methods, merged similar smells), it is possible that we accidentally analyze multiple concepts under same common name (this would be particularly visible for Blob smell and SVM machine learning method).

Additionally we assume that metrics are calculated in similar manner in multiple publications. While it would seem reasonable for a software metric (e.g., WMC or LoC) to always represent the same value, it is not guaranteed, especially if researchers use different frameworks for calculating metric values (or even different versions of the same framework), that they are actually implemented in exactly the same manner—even if used specification is same, there may be defects in implementation, variations between tools or versions of the same tool.

Next threat to internal validity arises from various projects and techniques used as teaching/training data for ML/AI algorithms. The range of used projects is remarkably wide, but for the sake of quantitative analysis we analyzed values of performance measures wrt. code smell, machine learning technique and used set of predictors, without regarding characteristics of specific data sets on which the values were calculated. While we were not able to find research proving that smells are context-sensitive, this does not seem unlikely, which constitutes a threat.

4.1.2 Construct validity

Construct validity concerns design of the study and its possibility to reflect the actual goal of the research. To avoid threats in study design we have applied a procedure of systematic literature review. To assure that researched area is relevant for study goal, we have cross-checked research questions with developers from *code quest* and adjusted them several times to address the business needs.

As always in literature review, it is possible that some relevant studies were not included in the search. To address this issue, we conducted a verification using a quasi-gold standard procedure using publications from 10 top venues from year 2017. However, both initial study selection and quasi-gold standard search were performed using Scopus database, therefore only publications present in Scopus are analyzed. The search term used in this systematic literature review is limited. For example, only papers referring to code smells via `"code smell" OR "bad smell" OR antipattern OR anti-pattern OR "anti pattern"` will form the initial data set. While it is possible that some will refer to the same concepts with a different naming, we believe that the terms are established well enough to ensure that a significant majority of relevant papers will use them.

We decided to focus on precision and treat recall as a slightly less important performance measure—this decision was made, because goal of the whole NCBiR project is to reduce ratio of false positive errors. False positive errors are related to code snippet classified as smells, but which are not perceived as smells. While we believe that focus on precision is a reasonable choice, it may be considered a risk for construct validity.

It is important to also note that, while the term "code smells" was coined to name the original 22 coding structures described by Fowler et al. [12], it is not exactly restricted to them—on the contrary, this metaphor was widely adopted to name not only structures in the source code, but also in the process, architecture and many other areas. This study is not restricted to the original set of smells, if others are well-represented, they will be analyzed as well.

4.1.3 External validity

External validity concerns possibility of generalizing the study to broader range of applications. Most of the papers studied smells in Java programming language, often in old versions (e.g., projects from 2005 in the work of Fontana and Zanoni [11]). While Java as a language is very common, we admit that it is not used in every branch of industry (e.g., iOS applications are generally written in Swift, web interfaces use mostly JavaScript, Android has recently adopted Kotlin as a standard language competing Java). Code smells described by Fowler et al. [12] were generally mean for object-oriented languages like Java, Smalltalk or C#—in other languages they may be even recommended solutions. This may be especially true when analyzing languages with different paradigm—which is relevant, considering recent increase of interest in functional programming (Scala, Elixir, Clojure) and adoption of its features even in the mainstream languages (e.g., Java). For example, Data Class is considered a code smell in object-oriented paradigm, but is an absolutely fine pattern in the realm of functional programming.

Projects used in most of the studies are relatively old. As a result, they are generally written in old versions of Java—even Java 5 or 6. As of today, the most recent version is Java 15 and the oldest supported LTS version is Java 11. Between these versions, significant changes were made to the core of the language, including shifting the paradigm from strictly object-oriented to object-oriented with minor functional features (like streams or immutable objects). This yields a threat for generalizing the results to newer versions of Java.

An big threat to external validity is the technique used by researchers to assess existence of code smells—in most cases these were assessed by a briefly trained students. This is a problem, because existence of smell may be linked to some more complex program structure, which cannot be spotted by novices.

4.1.4 Reliability

Reliability is concerned with possibility to reproduce the research and achieve same results. To guarantee maximum possibility for reproduction, we describe research procedure in detail in the and attach links to gathered data and processing scripts. However, some steps were performed manually. To further improve auditability, we provided a checklist for the first step of publication filtration. While most publication selection and data extraction was done by one person, we performed three levels of cross-checks (after initial screening, after final selection and after data gathering) with high level of agreement. Another threat is that in our original search, we considered all studies present in the database, i.e., we did not constrain upper bound for publication date. While this was done on purpose—to include as many recent studies as possible—the effect is that using the same search string will not yield same results, which may impact study reproducibility.

5 Conclusions

Interest in academia for using machine learning techniques for code smell detection has definitely increased as of lately, which is indicated by the growing number of papers published on the topic and conducted literature reviews.

It is clear that currently the most common predictors for whether a code sample constitutes a code smell or not are source code metrics. Typical machine learning algorithms are still dominant, with trees, SVMs, Random Forests and statistical methods being the most commonly used techniques. However, with the advent of deep learning, a new trend is visible—on one hand, feature reduction, and on the other—automated feature extraction from code using tools like word2vec.

Blob, Feature Envy, Data Class and Long Method are four most commonly researched smells in the literature. It is likely that existence of an independent data set provided by Fontana et al [SLR8] has boosted research in these particular areas. On the other hand, there are many smells that are only referred by a single paper, which may mean that either they are not noticed by the research community, or the research on them is carried under different labels (for example „anomalies" or „inconsistencies").

Precision, recall and F-measure are three most commonly reported model performance metrics. Full confusion matrix is reported only in a few cases.

Problem of data sets used for machine learning is visible and is actively addressed by researchers. In this review, we only found one data set used by several researchers ([SLR8]), but new ones have already been published ([15, 17]). We hope that this will lead the community into a shared understanding of the concept of each code smell, and to a solution that is relevant to the industry.

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

References

[1] Al-Shaaby, A., Aljamaan, H., Alshayeb, M.: Bad Smell Detection Using Machine Learning Techniques: A Systematic Literature Review. Arabian Journal for Science and Engineering **45**, 2341–2369 (2020). doi:10.1007/s13369-019-04311-w

[2] Azeem, M.I., Palomba, F., Shi, L., Wang, Q.: Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. Information and Software Technology **108**, 115 – 138 (2019). doi:https://doi.org/10.1016/j.infsof.2018.12.009

[3] Buenen, M., Muthukrishnan, G.: World quality report 2016-17. Tech. rep., Capgemini, Sogeti and Hewlett Packard Enterprise (2016)

[4] Caram, F., de Oliveira Rodrigues, B.R., Campanelli, A., Silva Parreiras, F.: Machine learning techniques for code smells detection: A systematic mapping study.

International Journal of Software Engineering and Knowledge Engineering **29**, 285–316 (2019). doi:10.1142/S021819401950013X

[5] Chen, B., Jiang, Z.M.: Characterizing and detecting anti-patterns in the logging code. In: Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering, ICSE 2017, pp. 71–81 (2017). doi:10.1109/ICSE.2017.15

[6] Di Nucci, D., Palomba, F., Tamburri, D.A., Serebrenik, A., De Lucia, A.: Detecting code smells using machine learning techniques: Are we there yet? In: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 612–621 (2018). doi:10.1109/SANER.2018.8330266

[7] Dieste, O., Grimán, A., Juristo, N.: Developing search strategies for detecting relevant experiments. Empirical Software Engineering **14**(5), 513–539 (2009). doi:10.1109/ESEM.2007.19

[8] Dybå, T., Dingsøyr, T.: Empirical Studies of Agile Software Development: A Systematic Review. Information and Software Technology **50**(9-10), 833–859 (2008). doi:10.1016/j.infsof.2008.01.006

[9] Fleiss, J.L.: Measuring nominal scale agreement among many raters. Psychological Bulletin **76**(5), 378–382 (1971). doi:10.1037/h0031619

[10] Fontana, F.A., Pigazzini, I., Roveda, R., Zanoni, M.: Automatic detection of instability architectural smells. In: Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, pp. 433–437 (2017). doi:10.1109/ICSME.2016.33

[11] Fontana, F.A., Zanoni, M.: Code smell severity classification using machine learning techniques. Knowledge-Based Systems **128**, 43–58 (2017). doi:10.1016/j.knosys.2017.04.014

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

[13] Gartner: Gartner says worldwide software market grew 4.8 percent in 2013 (2014)

[14] Kitchenham, B., Budgen, D., Brereton, P.: Evidence-Based Software Engineering and Systematic Reviews. CRC Press (2016). doi:10.1007/11767718_3

[15] Madeyski, L., Lewowski, T.: MLCQ: Industry-Relevant Code Smell Data Set. In: Proceedings of the Evaluation and Assessment in Software Engineering, EASE '20, p. 342–347. Association for Computing Machinery, New York, NY, USA (2020). doi:10.1145/3383219.3383264

[16] Malhotra, R.: A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing **27**, 504 – 518 (2015). doi:10.1016/j.asoc.2014.11.023

[17] Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R., Lucia, A.: On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. Empirical Software Engineering pp. 1–34 (2017). doi:10.1007/s10664-017-9535-z

[18] Palomba, F., Di Nucci, D., Panichella, A., Zaidman, A., De Lucia, A.: Lightweight detection of android-specific code smells: The adoctor project. In: SANER 2017 - 24th IEEE International Conference on

Software Analysis, Evolution, and Reengineering, pp. 487–491 (2017). doi:10.1109/SANER.2017.7884659

[19] Palomba, F., Di Nucci, D., Tufano, M., Bavota, G., Oliveto, R., Poshyvanyk, D., De Lucia, A.: Landfill: An Open Dataset of Code Smells with Public Evaluation. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pp. 482–485 (2015). doi:10.1109/MSR.2015.69

[20] Palomba, F., Panichella, A., Zaidman, A., Oliveto, R., De Lucia, A.: The scent of a smell: An extensive comparison between textual and structural smells. IEEE Transactions on Software Engineering (2017). doi:10.1109/TSE.2017.2752171

[21] Romano, S., Scanniello, G., Sartiani, C., Risi, M.: A graph-based approach to detect unreachable methods in java software. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, SAC '16, p. 1538–1541. Association for Computing Machinery, New York, NY, USA (2016). doi:10.1145/2851613.2851968

[22] Santos, J.A.M., Rocha-Junior, J.B., Prates, L.C.L., do Nascimento, R.S., Freitas, M.F., de Mendonça, M.G.: A systematic review on the code smell effect. Journal of Systems and Software **144**, 450 – 477 (2018). doi:https://doi.org/10.1016/j.jss.2018.07.035

[23] Sharma, T., Spinellis, D.: A survey on software smells. Journal of Systems and Software **138**, 158 – 173 (2018). doi:https://doi.org/10.1016/j.jss.2017.12.034

[24] Singh, S., Kaur, S.: A systematic literature review: Refactoring for disclosing code smells in object oriented software. Ain Shams Engineering Journal (2017). doi:https://doi.org/10.1016/j.asej.2017.03.002

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

[26] Wasylkowski, A., Zeller, A., Lindig, C.: Detecting object usage anomalies. In: 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2007, pp. 35–44 (2007). doi:10.1145/1287624.1287632

[27] Wen, J., Li, S., Lin, Z., Hu, Y., Huang, C.: Systematic Literature Review of Machine Learning Based Software Development Effort Estimation Models. Information and Software Technology **54**(1), 41–59 (2012). doi:10.1016/j.infsof.2011.09.002

[28] Wohlin, C.: 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). doi:10.1145/2601248.2601268

[29] Zhang, H., Babar, M.A., Tell, P.: Identifying Relevant Studies in Software Engineering. Information and Software Technology **53**(6), 625–637 (2011). doi:10.1016/j.infsof.2010.12.010

[30] Zhang, M., Hall, T., Baddoo, N.: Code Bad Smells: a review of current knowledge. Journal of Software Maintenance and Evolution: Research and Practice **23**(3), 179–202 (2011). doi:10.1002/smr.521

Systematic Literature Review References

[SLR1] Amorim, L., Costa, E., Antunes, N., Fonseca, B., Ribeiro, M.: 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 (2016). doi:10.1109/ISSRE.2015.7381819

[SLR2] Barbez, A., Khomh, F., Gueheneuc, Y.G.: Deep learning anti-patterns from code metrics history. In: Proceedings - 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019, pp. 114–124 (2019). doi:10.1109/ICSME.2019.00021

[SLR3] Barbez, A., Khomh, F., Guéhéneuc, Y.G.: A machine-learning based ensemble method for anti-patterns detection. Journal of Systems and Software **161** (2020). doi:10.1016/j.jss.2019.110486

[SLR4] Boussaa, M., Kessentini, W., Kessentini, M., Bechikh, S., Ben Chikha, S.: Competitive coevolutionary code-smells detection. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **8084 LNCS**, 50–65 (2013). doi:10.1007/978-3-642-39742-4_6

[SLR5] Bryton, S., Brito e Abreu, F., Monteiro, M.: Reducing subjectivity in code smells detection: Experimenting with the long method. In: Proceedings - 7th International Conference on the Quality of Information and Communications Technology, QUATIC 2010, pp. 337–342 (2010). doi:10.1109/QUATIC.2010.60

[SLR6] Chen, Z., Chen, L., Ma, W., Zhou, X., Zhou, Y., Xu, B.: Understanding metric-based detectable smells in python software: A comparative study. Information and Software Technology **94**, 14–29 (2018). doi:10.1016/j.infsof.2017.09.011

[SLR7] Fakhoury, S., Arnaoudova, V., Noiseux, C., Khomh, F., Antoniol, G.: Keep it simple: Is deep learning good for linguistic smell detection? In: 25th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2018 - Proceedings, vol. 2018-March, pp. 602–611 (2018). doi:10.1109/SANER.2018.8330265

[SLR8] Fontana, F.A., Mäntylä, M.V., Zanoni, M., Marino, A.: Comparing and experimenting machine learning techniques for code smell detection. Empirical Software Engineering **21**(3), 1143–1191 (2016). doi:10.1007/s10664-015-9378-4

[SLR9] Fontana, F.A., Zanoni, M., Marino, A., Mäntylä, M.V.: Code smell detection: Towards a machine learning-based approach. In: IEEE International Conference on Software Maintenance, ICSM, pp. 396–399 (2013). doi:10.1109/ICSM.2013.56

[SLR10] Fu, S., Shen, B.: Code bad smell detection through evolutionary data mining. In: International Symposium on Empirical Software Engineering and Measurement, vol. 2015-November, pp. 41–49 (2015). doi:10.1109/ESEM.2015.7321194

[SLR11] Gauthier, F., Merlo, E.: Semantic smells and errors in access control models: A case study in PHP. In: Proceedings - International Conference on Software Engineering, pp. 1169–1172 (2013). doi:10.1109/ICSE.2013.6606670

[SLR12] Grodzicka, H., Ziobrowski, A., Łakomiak, Z., Kawa, M., Madeyski, L.: Code Smell Prediction Employing Machine Learning Meets Emerging Java Language Constructs. In: A. Poniszewska-Marańda, N. Kryvinska, S. Jarząbek, L. Madeyski (eds.) Data-Centric Business and Applications: Towards Software Development (Volume 4), vol. 40 of book series Lecture Notes on Data Engineering and Communications Technologies, pp. 137–167. Springer International Publishing, Cham (2020). doi:10.1007/978-3-030-34706-2_8

[SLR13] Guggulothu, T., Moiz, S.A.: Code smell detection using multi-label classification approach. Software Quality Journal (2020). doi:10.1007/s11219-020-09498-y

[SLR14] Guo, X., Shi, C., Jiang, H.: Deep semantic-based feature envy identification. In: ACM International Conference Proceeding Series (2019). doi:10.1145/3361242.3361257

[SLR15] Hadj-Kacem, M., Bouassida, N.: A Hybrid Approach To Detect Code Smells using Deep Learning. In: ENASE 2018 - Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering, vol. 2018-March, pp. 137–146 (2018). doi:10.5220/0006709801370146

[SLR16] Hadj-Kacem, M., Bouassida, N.: Deep Representation Learning for Code Smells Detection using Variational Auto-Encoder. In: Proceedings of the International Joint Conference on Neural Networks, vol. 2019-July (2019). doi:10.1109/IJCNN.2019.8851854

[SLR17] Hassaine, S., Khomh, F., Guéhéneucy, Y.G., Hamel, S.: IDS: An immune-inspired approach for the detection of software design smells. In: Proceedings - 7th International Conference on the Quality of Information and Communications Technology, QUATIC 2010, pp. 343–348 (2010). doi:10.1109/QUATIC.2010.61

[SLR18] Hozano, M., Antunes, N., Fonseca, B., Costa, E.: 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,, pp. 474–482. INSTICC, SciTePress (2017). doi:10.5220/0006338804740482

[SLR19] Karaduzovic-Hadziabdic, K., Spahic, R.: Comparison of machine learning methods for code smell detection using reduced features. In: UBMK 2018 - 3rd International Conference on Computer Science and Engineering, pp. 670–672 (2018). doi:10.1109/UBMK.2018.8566561

[SLR20] Kaur, A., Jain, S., Goel, S.: A support vector machine based approach for code smell detection. In: Proceedings - 2017 International Conference on Machine Learning and Data Science, MLDS 2017, vol. 2018-January, pp. 9–14 (2018). doi:10.1109/MLDS.2017.8

[SLR21] Kaur, A., Jain, S., Goel, S.: SP-J48: a novel optimization and machine-learning-based approach for solving complex problems: special application in software engineering for detecting code smells. Neural Computing and Applications (2019). doi:10.1007/s00521-019-04175-z

[SLR22] Kessentini, M., Ouni, A.: Detecting Android Smells Using Multi-Objective Genetic Programming. In: 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft), pp. 122–132 (2017). doi:10.1109/MOBILESoft.2017.29

[SLR23] Kessentini, W., Kessentini, M., Sahraoui, H., Bechikh, S., Ouni, A.: A cooperative parallel search-based software engineering approach for code-smells detection. IEEE Transactions on Software Engineering **40**(9), 841–861 (2014). doi:10.1109/TSE.2014.2331057

[SLR24] Khomh, F., Vaucher, S., Guéehéeneuc, Y.G., Sahraoui, H.: A Bayesian Approach for the Detection of Code and Design Smells. In: Proceedings - International Conference on Quality Software, pp. 305–314 (2009). doi:10.1109/QSIC.2009.47

[SLR25] Kiyak, E.O., Birant, D., Birant, K.U.: Comparison of multi-label classification algorithms for code smell detection. In: 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies, ISMSIT 2019 - Proceedings (2019). doi:10.1109/ISMSIT.2019.8932855

[SLR26] Kreimer, J.: Adaptive detection of design flaws. Electronic Notes in Theoretical Computer Science **141**(4 SPEC. ISS.), 117–136 (2005). doi:10.1016/j.entcs.2005.02.059

[SLR27] Liu, H., Jin, J., Xu, Z., Bu, Y., Zou, Y., Zhang, L.: Deep learning based code smell detection. IEEE Transactions on Software Engineering (2019). doi:10.1109/TSE.2019.2936376

[SLR28] Liu, H., Liu, Q., Niu, Z., Liu, Y.: Dynamic and automatic feedback-based threshold adaptation for code smell detection. IEEE Transactions on Software Engineering **42**(6), 544–558 (2016). doi:10.1109/TSE.2015.2503740

[SLR29] Liu, H., Xu, Z., Zou, Y.: Deep learning based feature envy detection. In: ASE 2018 - Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp. 385–396 (2018). doi:10.1145/3238147.3238166

[SLR30] Maiga, A., Ali, N., Bhattacharya, N., Sabané, A., Guéhéneuc, Y.G., Aimeur, E.: SMURF: A SVM-based incremental anti-pattern detection approach. In: Proceedings - Working Conference on Reverse Engineering, WCRE, pp. 466–475 (2012). doi:10.1109/WCRE.2012.56

[SLR31] Mansoor, U., Kessentini, M., Maxim, B.R., Deb, K.: Multi-objective code-smells detection using good and bad design examples. Software Quality Journal **25**(2), 529–552 (2017). doi:10.1007/s11219-016-9309-7

[SLR32] Merzah, B.M.: Software quality prediction using data mining techniques. In: 2019 International Conference on Information and Communications Technology, ICOIACT 2019, pp. 394–397 (2019). doi:10.1109/ICOIACT46704.2019.8938487

[SLR33] Mkaouer, M.W.: 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 LNCS**, 281–287 (2016). doi:10.1007/978-3-319-47106-8_24

[SLR34] Ocariza, F.S., Pattabiraman, K., Mesbah, A.: Detecting unknown inconsistencies in web applications. In: ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, pp. 566–577 (2017). doi:10.1109/ASE.2017.8115667

[SLR35] Palomba, F.: Textual Analysis for Code Smell Detection. In: Proceedings - International Conference on Software Engineering, vol. 2, pp. 769–771 (2015). doi:10.1109/ICSE.2015.244

[SLR36] Palomba, F.: Alternative sources of information for code smell detection: Postcards from far away. In: Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, pp. 636–640 (2017). doi:10.1109/ICSME.2016.26

[SLR37] Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A., Poshyvanyk, D.: Detecting bad smells in source code using change history information. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013 - Proceedings, pp. 268–278 (2013). doi:10.1109/ASE.2013.6693086

[SLR38] Pradel, M., Heiniger, S., Gross, T.R.: Static detection of brittle parameter typing. In: Proceedings of the 2012 International Symposium on Software Testing and Analysis, ISSTA 2012, p. 265–275. Association for Computing Machinery, New York, NY, USA (2012). doi:10.1145/2338965.2336785

[SLR39] Rubin, J., Henniche, A.N., Moha, N., Bouguessa, M., Bousbia, N.: Sniffing android code smells: An association rules mining-based approach. In: Proceedings - 2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems, MOBILESoft 2019, pp. 123–127 (2019). doi:10.1109/MOBILESoft.2019.00025

[SLR40] S, J.B.F., Vinod, V.: Design and analysis of improvised genetic algorithm with particle swarm optimization for code smell detection. International Journal of Innovative Technology and Exploring Engineering **9**(1), 5327–5330 (2019). doi:10.35940/ijitee.A5328.119119

[SLR41] Sahin, D., Kessentini, M., Bechikh, S., Deb, K.: Code-smell detection as a bilevel problem. ACM Transactions on Software Engineering and Methodology **24**(1) (2014). doi:10.1145/2675067

[SLR42] Sharma, P., Kaur, E.A.: Design of testing framework for code smell detection (OOPS) using BFO algorithm. International Journal of Engineering and Technology(UAE) **7**(2.27 Special Issue 27), 161–166 (2018). doi:10.14419/ijet.v7i2.27.14635

[SLR43] Tummalapalli, S., Kumar, L., Neti, L.B.M.: An empirical framework for web service anti-pattern prediction using machine learning techniques. In: IEMECON 2019 - 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference, pp. 137–143 (2019). doi:10.1109/IEMECONX.2019.8877008

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