Code smells enabled by artificial intelligence: A Systematic Mapping

Moayid Ali Zaidi[1], Ricardo Colomo-Palacios[1]

[1] Faculty of Computer Sciences, Østfold University College
Postboks 700, 1757 Halden, Norway
[moayid.a.zaidi, ricardo.colomo-palacios]@hiof.no

Abstract. Code smells are an indicator of poor design in software systems. Artificial intelligence techniques have been applied in several ways to improve soft-ware quality in code smells detection i.e. (detection rules or standards using a combination of object-oriented metrics and Bayesian inference graphs). Literature in the field has identified artificial intelligence techniques and compare different artificial intelligence algorithms, which are used in the detection of code smells. However, to the best of our knowledge, there is not a systematic literature review devoted to study in deep the interaction of these fields. In this paper, authors conduct a systematic mapping to get to know how artificial intelligence inter-acts with code smells. Results show the deep connection of Artificial Intelligence with code smells in a solid way, as well as, providing potential challenges and opportunities for future research.

Keywords: Artificial intelligence, code smells, bad smells, Systematic Mapping

1 Introduction

The source code of the system is refined by iterations, reconstructed and changed due to many reasons to reflect the modification in requirements, enhance or to improve its features. It is reported that, these type of maintenance activities consume 90 percent of the total cost of the project [1]. Given the importance of software quality, several practices have been designed to improve its excellence and maintainability. Code smells are assumed to indicate bad design that leads to less maintainable code [2]. Code smells explained by Fowler [3] are symptoms of poor design and implementation choices. He also defines 22 sets of symptoms of code smells. They are also called design anomalies and they refer to design circumstances which affect the maintenance of the software. Code smell detection is seen as a technique to identify refactoring opportunities [4]. These refactoring opportunities include aspects like reducing program error proneness, increasing program comprehensibility, fault- and change-proneness and combating software design degradation, naming just some of the reported aspects[5]. Code smells present the advantage of focusing on aspects that requires deeper analysis [6].

However, the task is normally quite complex and time consuming [7], making software engineering highly dependent on human capital [8]. To avoid over costs and improve overall efficiency, there is a trend nowadays to automate software tasks in the field [9]. Therefore, several automatic detectors, able to identify instances of code smells in the source code have been produced by the literature [10]. Most of the works are based on the analysis of the structural properties of the source code. Heuristics-Based smell detectors have been proposed as well; it has two steps where metrics are computed first and then these metrics have to separate the smelly and non-smelly classes through thresholds [11]. There have been reports that, these detectors have been performing good in terms of accuracy [12]. Although, most of the detectors require the specific threshold to separate the non-smelly or smelly code and the threshold selection has an impact on the accuracy.

Machine Learning (ML) techniques and methods have been used in the detection of code smells [13] extensively. In this sense, supervised learning (when the model is trained by labeled data) is used and independent attributes are used to predict the dependent attributes using regression and other techniques. A first look at literature shows a set of initiatives in the intersection of these two fields. Where, Kreimer proposed an adaptive detection technique for smells based on metrics [14]. Khomh suggested a Bayesian approach to detect the existences of antipatterns in open source programs [15]. He also presented a proposal based on Bayesian belief networks [16] and Yang applied ML algorithms on code clones [17], naming just some of the most relevant works.

Given the increasing importance of the topic, in this paper, authors present a systematic mapping devoted to investigate the intersection of these two fields. To the best of our knowledge, there is no systematic mapping on artificial intelligence (AI) interacting with code smells. Although, there is a recent effort devoted to study ML techniques for code smell detection [11]. In spite there is a similar goal, our approach is wider in the use of AI as a whole and not only ML aspects, providing with it new insights on the topic.

The remainder of this paper is organized as follows: In Section 2, the method used in this study is described. In Section 3, the result of the systematic mapping is discussed. Section 4 presents the conclusion and provides recommendations for further research on this topic.

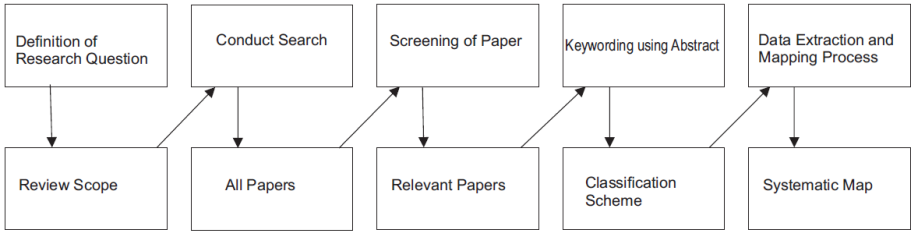
2 Research method

The purpose of this study is to structure and characterize the state of the practice on the use of AI techniques in code smells. This will be done by means of an analysis of previous works published in the literature to provide an overview of the topic and to help discover potential gaps for future research. Thus, the main research question driving this study is:

What is the state of the practice of the use of AI in code smells?

To do so, our study follows the guidelines by the Petersen [18] for Systematic Mapping literature studies. Figure 1 shows the systematic mapping process flow. Authors followed these steps to conduct the systematic mapping.

Process steps



Outcomes

Fig. 1. Systematic Mapping Process

2.1 Research questions

In the context of our goal underlined before, research questions are as follows:

RQ1: Which fields inside AI present more impact in Code Smells?

RQ2: Which aspects in Code Smells have been reported as influenced by AI?

RQ3: What are the reported challenges and opportunities in the use of AI in Code Smells?

In the first research question our focus is to detect which fields in artificial intelligence are more active in the code smells, while in the second, we discuss which features in code smells have been reported as affected by AI and finally, the third question investigates potential challenges and opportunities.

2.2 Study Protocol

Authors constructed a search string according to our basic goal and research questions. The string should be simple, in order to provide us several results to cover our exact topic. We use Boolean operator OR for the concatenation of alternative words and spellings, whereas, the AND operator used for the concatenation of major terms [19]. After identifying the keywords, we formulated the string and then we selected different digital libraries and databases on the basis of popularity in the computing field [18]. Although small modifications in the syntax were performed, the general string was:

("Artificial Intelligence") AND ("Code smell" OR "bad smell")

With regards to databases, the following databases were used to find relevant literature on the topic:

- ACM Digital Library (http://dl.acm.org)
- IEEE Explore (https://explore.ieee.com)
- SpringerLink (https://link.springer.com)
- ScienceDirect (https://www.sciencedirect.com/)

- Wiley Online Library (https://onlinelibrary.wiley.com)

These databases were chosen by authors, given that they are among the most relevant sources of articles within the broad field of computing and because they are accessible using institutional accounts.

Zotero was used as a reference management tool, to manage references and to avoid duplications.

The criteria of inclusion and exclusion are used to exclude articles and studies which are not relevant to answer our research questions. Table 1 depicts the inclusion and exclusion criteria we applied in this study:

Table 1. Inclusion and exclusion criteria used in the review

Inclusion Criteria	**Exclusion Criteria**
Papers written in English reporting AI techniques for code smells.	The paper contains no relevant data to answer the research questions.
Available papers that can provide answers to research questions	Paper's full text is not available,
Papers published 2014 onwards	Paper is not peer reviewed
	Papers Published before (2014)

For our study, authors followed the process which is shown in Figure 2. We followed a keywording strategy in this paper. We read the abstract and inspect the keywords and the concept of the paper and identify the context of the research. By doing this, the keywords from different papers are joined to develop a high level of understanding about the nature of the research, as well as, helping us to set up the different categories. When abstract is not able to explain the meaningful keywords then, we also study the introduction part and conclusion part of the paper. When the significant keywords have been chosen, then they are used to create the categories for the map.

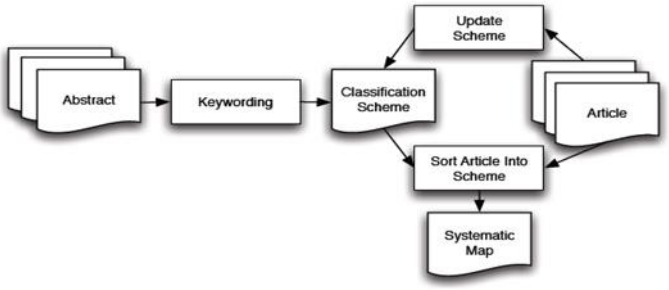


Fig. 2. Classification Scheme adopted

In our study, in order to analyze the kind of works present in the literature, we classify papers based on three different aspects: research scope, contribution type and research.

After that the classification took place, we sorted the relevant articles to the scheme. While working on the data extraction, the classification scheme evolved dur-

ing the process, by merging and adding the categories. We used Excel for the data extraction process. The following categories are adopted.

Firstly, with regards to research scope, categories are as follows:

- Challenges: In this category, the challenges on the topic are discussed.
- Opportunities: This category contains articles with different opportunities for the topic.
- Fields: This category includes different fields used in the paper.
- Aspects: This category includes aspects which are reported.

With regards to contribution type, authors adopted the following categories:

- Process: A process describes the activities, action and their workflow.
- Tool: A software tool is developed to support code smells.
- Model: Representation of information to be used in the topic.
- Technique: It is used to achieve a specific task. It could come, accompanied by a support tool.

The final classification is research type. In this aspect, the following categories are adopted:

- Solution proposal: A solution to the problem is proposed.
- Opinion Paper: Someone expresses personal opinion.
- Experience Paper: What and how something is done in practice.
- Validation Research: The technique not yet been implemented in practice and novel.

3 Results

Table 2 presents results of the search performed in the different databases:

Table 2. Results obtained in the different databases and steps

Database	***Step 1***	***Step 2***
IEEE	7	3
ACM	9	1
Springer	75	3
ScienceDirect	20	2
Wiley Online Library	1	1
Total	**112**	**10**

In the column entitled Step 1, authors wrote initial results per database and in Step 2, results after inclusion and exclusion criteria are coded. One can find the significance of the various databases in the search and the key importance of Springer both for Step 1 and Step 2.

Apart from results themselves, it is also worth to present frequencies in each aspect. That is, the Analysis of the outcomes presenting the frequencies of publications for each category. It results in concluding which categories have been highlighted in past research and then, to find gaps and possibilities for future research work. This is

in further demonstrated by x-y coordinates scatterplots with bubbles in category intersections. The size of the bubble depends on the number of articles that are pair in each category, corresponding to the bubble coordinates. Figure 3 presents the information in a graphical way:

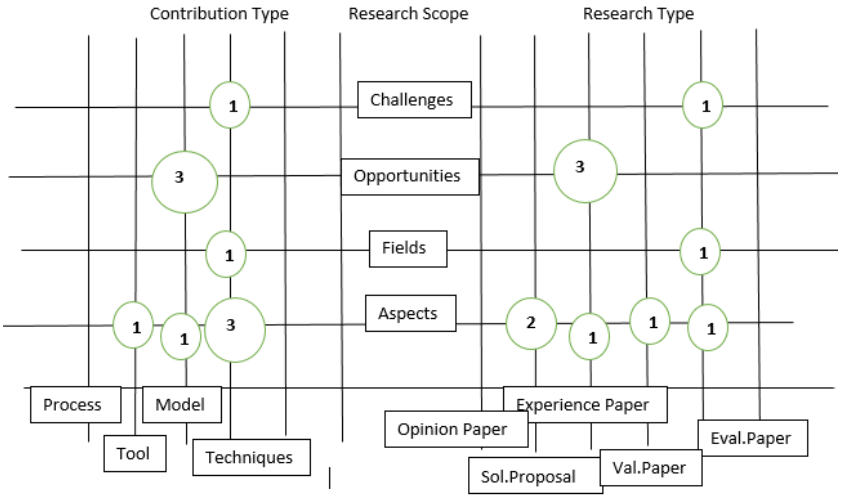


Fig. 3. Bubble plot of the distribution of contributions among the three criteria

The plot clearly shows that, the number of experience papers with different types of opportunities are more than any other category in research type. It also indicates, most of the papers are written on aspects of AI with different techniques and diverse opportunities, with different types of models in contribution type. Other areas have less number of papers so still, there is a need to do more research in those areas.

4 Analysis and Discussion

In this section, research questions are answered by authors by means of the literature review conducted.

RQ1: Which fields inside AI is more active in Code Smells?

Artificial intelligence is a vast term and it includes fields like ML, natural language processing (NLP), vision, expert systems, and robotics. As stated before, there is a very recent study on the use of ML in code smells, however, our idea is quite broader, including AI as a whole. Consistent with previous studies, in our work we found several ML methods which are used to detect code smells, compare different types of code smells tools and suggest different techniques for solving the code smell problem. Bad smells can be predicted using ML algorithms [20] and these techniques are also used to classify the severity of code smells.

The algorithms from ML as well as artificial intelligence helps to detect the smells and improve the performance of detection techniques. These algorithms are used to

detect smells in both object-oriented and software-oriented paradigms[21]. In natural language processing applications, the support vector machine (SVM) is used to detect the code smell or anti-pattern i.e. SC (spaghetti code), Blob, FD (functional decomposition) and SAK (Swiss Army Knife) [22].

Analytical learning techniques are also used to identify code smells. With regards to the distribution of efforts among fields in the period analyzed (2014-2019), this is as follows:

- ML: 8 studies
- NLP: 1 study
- Expert Systems: 1 study

It is important to mention the importance of ML techniques for the code smell detection. With that, we agree with [11] on the key role played by ML, but it is also worth to note the scarce repercussion of NLP and expert systems in this field. However, it is also worth to note that most of the ML based solutions also use, at least partially, NLP techniques.

RQ2: Which Aspects in Code Smells research have been reported as influenced by AI?

With the development of ML, the number of proposed approaches are increasing over the years. The propose approaches are decision-based approach, Bayesian belief networks, Support Vector Machine(SVM), and a metrics-based approach to identify feature envy and code smells [23]. Usually, as literature reports, a supervised method is used to detect code smells [24]. ML based and rule-based approaches gained importance and most of the researchers move to work with these approaches because, metrics, rules and classifier algorithm can define to detect smells in a more precise way[25]. NLP has also used for the detection and correction of smells or REST (Representational State Transfer) linguistics antipatterns [21]. Ban also identify the bugs and antipatterns through AI algorithms[26]. Feature envy detection has also been reported by the AI, one of the most common code smell which is exploited by the deep learning. The proposed approach improves the refactoring solutions and features envy detection[23]. Table 3 shows the different machine learning methods on the basis of accuracy on identifying different code smells [27]. The reason behind selecting these code smells is that it has a severe impact on the software quality i.e. God Class, Data Class, Feature Envy and Long Method [22].

Table 3. Comparison of ML Methods

ML Method	***Data Class***	***God Class***	***Feature Envy***	***Long Method***
JRip	97.14	**97.86**	96.19	99.52
Random Forest	**98.57**	97.38	96.90	**99.76**
Multilayer Perception	97.86	96.43	**99.67**	93.43
Naïve Bayes	83.81	95.95	89.76	96.43

In Table 3., the results which are bolded, are the highest accuracy in the detection of code smells. For the Data Class, Random Forest achieved 98.57%, for God class JRip achieved 97.86%, for Feature Envy Multilayer Perceptron achieved 99.76% and Random Forest achieved 99.76 for Long Method.

The three main aspects for code smell detection (i) performance of a set of classifiers over the total number of sample in the dataset, (ii) accurately detect the code smell on the minimum training dataset size, (iii) different classifiers detected the number of code smell over the dataset) are analyzed by ML algorithms[23].

RQ3: What are the reported challenges and opportunities in the use of AI in Code Smells?

One of the important challenges which are faced by the ML based detectors is the need to gather a large number of labeled samples to train the classifiers. In fact, more study is needed toward structuring the datasets correctly within the predictors to be used [24]. The disadvantage of using the relational model is that, it only specifies the defective Class, but cannot find the exact problem; so a separate analysis of metrics and metrics error is required to determine the defect [28]. The main focus of the studies are on detection of smells as compared to maintenance. There is also literature reporting the use of version control in code smells [21]. According to Nunez, there are less papers on code smell detection in object-oriented metrics and on object-oriented programming compared to other paradigms [29]. Apart from literature itself, recent studies find that there is a lack of human expertise and validation benchmarking processes [25]. Bafandeh Mayvan underlined that 17% of the publications on the code smell and anti-patterns detection present a lack of maturity [30]. So, authors perform an indirect call for mature works in the topic. Maturity in tools is a common issue in all available solutions in a variety of fields of application and distribution models [31], and maybe both the relative novelty of the topic is not helping to get a mature and standardized versions code smells tools and processes.

5 Conclusions and future work

A code smell is an understanding that something has gone wrong in your code. Code smells describe a particular sort of outline failures in the code. The main motivation for this work was to examine how Artificial intelligence interacts with the code smells through systematically mapping literature on the topic. Results show the importance of the intersection of these research fields paving the way to new research efforts. Conclusions on our work also identifies challenges and opportunities in the topic. In addition, the review conducted highlights that ML is the most active field inside AI in code smells while vision, NLP, robotics and expert system attracts less attention. However, and in spite of the low attention, authors want to extend their works towards the use of NLP in Code Smells apart from the use of ML techniques in the detection of code and architectural smells.

References

1. Kessentini, W., Kessentini, M., Sahraoui, H., Bechikh, S., Ouni, A.: A Cooperative Parallel Search-Based Software Engineering Approach for Code-Smells Detection. IEEE Trans. Softw. Eng. 40, 841–861 (2014). https://doi.org/10.1109/TSE.2014.2331057.
2. Sjøberg, D.I.K., Yamashita, A., Anda, B.C.D., Mockus, A., Dybå, T.: Quantifying the Effect of Code Smells on Maintenance Effort. IEEE Trans. Softw. Eng. 39, 1144–1156 (2013). https://doi.org/10.1109/TSE.2012.89.
3. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional (2018).
4. Hozano, M., Garcia, A., Fonseca, B., Costa, E.: Are you smelling it? Investigating how similar developers detect code smells. Inf. Softw. Technol. 93, 130–146 (2018). https://doi.org/10.1016/j.infsof.2017.09.002.
5. Palomba, F., Bavota, G., Penta, M.D., Fasano, F., Oliveto, R., Lucia, A.D.: On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. Empir. Softw. Eng. 23, 1188–1221 (2018). https://doi.org/10.1007/s10664-017-9535-z.
6. Walter, B., Fontana, F.A., Ferme, V.: Code smells and their collocations: A large-scale experiment on open-source systems. J. Syst. Softw. 144, 1–21 (2018). https://doi.org/10.1016/j.jss.2018.05.057.
7. Liu, H., Guo, X., Shao, W.: Monitor-Based Instant Software Refactoring. IEEE Trans. Softw. Eng. 39, 1112–1126 (2013). https://doi.org/10.1109/TSE.2013.4.
8. Garcia-Crespo, A., Colomo-Palacios, R., Gomez-Berbis, J.M., Mencke, M.: BMR: benchmarking metrics recommender for personnel issues in software development projects. Int. J. Comput. Intell. Syst. 2, 257–267 (2009).
9. Colomo-Palacios, R., Fernandes, E., Soto-Acosta, P., Larrucea, X.: A case analysis of enabling continuous software deployment through knowledge management. Int. J. Inf. Manag. 40, 186–189 (2018). https://doi.org/10.1016/j.ijinfomgt.2017.11.005.
10. Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R., De Lucia, A.: A large-scale empirical study on the lifecycle of code smell co-occurrences. Inf. Softw. Technol. 99, 1–10 (2018). https://doi.org/10.1016/j.infsof.2018.02.004.
11. Azeem, M.I., Palomba, F., Shi, L., Wang, Q.: Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. Inf. Softw. Technol. 108, 115–138 (2019). https://doi.org/10.1016/j.infsof.2018.12.009.
12. Fernandes, E., Oliveira, J., Vale, G., Paiva, T., Figueiredo, E.: A review-based comparative study of bad smell detection tools. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering - EASE '16. pp. 1–12. ACM Press, Limerick, Ireland (2016). https://doi.org/10.1145/2915970.2915984.
13. Arcelli Fontana, F., Mäntylä, M.V., Zanoni, M., Marino, A.: Comparing and experimenting machine learning techniques for code smell detection. Empir. Softw. Eng. 21, 1143–1191 (2016). https://doi.org/10.1007/s10664-015-9378-4.

14. Kreimer, J.: Adaptive Detection of Design Flaws. Electron. Notes Theor. Comput. Sci. 141, 117–136 (2005). https://doi.org/10.1016/j.entcs.2005.02.059.
15. Khomh, F., Vaucher, S., Guéhéneuc, Y., Sahraoui, H.: A Bayesian Approach for the Detection of Code and Design Smells. In: 2009 Ninth International Conference on Quality Software. pp. 305–314 (2009). https://doi.org/10.1109/QSIC.2009.47.
16. Khomh, F., Vaucher, S., Guéhéneuc, Y.-G., Sahraoui, H.: BDTEX: A GQM-based Bayesian approach for the detection of antipatterns. J. Syst. Softw. 84, 559–572 (2011). https://doi.org/10.1016/j.jss.2010.11.921.
17. Yang, J., Hotta, K., Higo, Y., Igaki, H., Kusumoto, S.: Filtering clones for individual user based on machine learning analysis. In: 2012 6th International Workshop on Software Clones (IWSC). pp. 76–77. IEEE, Zurich, Switzerland (2012). https://doi.org/10.1109/IWSC.2012.6227872.
18. Petersen, K., Vakkalanka, S., Kuzniarz, L.: Guidelines for conducting systematic mapping studies in software engineering: An update. Inf. Softw. Technol. 64, 1–18 (2015). https://doi.org/10.1016/j.infsof.2015.03.007.
19. da Mota Silveira Neto, P.A., Carmo Machado, I. do, McGregor, J.D., de Almeida, E.S., de Lemos Meira, S.R.: A systematic mapping study of software product lines testing. Inf. Softw. Technol. 53, 407–423 (2011). https://doi.org/10.1016/j.infsof.2010.12.003.
20. Czibula, G., Marian, Z., Czibula, I.G.: Detecting software design defects using relational association rule mining. Knowl. Inf. Syst. 42, 545–577 (2015). https://doi.org/10.1007/s10115-013-0721-z.
21. Sabir, F., Palma, F., Rasool, G., Guéhéneuc, Y.-G., Moha, N.: A systematic literature review on the detection of smells and their evolution in object-oriented and service-oriented systems. Softw. Pract. Exp. 49, 3–39 (2019). https://doi.org/10.1002/spe.2639.
22. Kaur, A., Jain, S., Goel, S.: A Support Vector Machine Based Approach for Code Smell Detection. In: 2017 International Conference on Machine Learning and Data Science (MLDS). pp. 9–14 (2017). https://doi.org/10.1109/MLDS.2017.8.
23. Liu, H., Xu, Z., Zou, Y.: Deep learning based feature envy detection. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering - ASE 2018. pp. 385–396. ACM Press, Montpellier, France (2018). https://doi.org/10.1145/3238147.3238166.
24. Nucci, D.D., Palomba, F., Tamburri, D.A., Serebrenik, A., Lucia, A.D.: 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). https://doi.org/10.1109/SANER.2018.8330266.
25. Alkharabsheh, K., Crespo, Y., Manso, E., Taboada, J.A.: Software Design Smell Detection: a systematic mapping study. Softw. Qual. J. (2018). https://doi.org/10.1007/s11219-018-9424-8.
26. Bán, D., Ferenc, R.: Recognizing Antipatterns and Analyzing Their Effects on Software Maintainability. In: Murgante, B., Misra, S., Rocha, A.M.A.C., Torre,

C., Rocha, J.G., Falcão, M.I., Taniar, D., Apduhan, B.O., and Gervasi, O. (eds.) Computational Science and Its Applications – ICCSA 2014. pp. 337–352. Springer International Publishing (2014).
27. Karađuzović-Hadžiabdić, K., Spahić, R.: Comparison of Machine Learning Methods for Code Smell Detection Using Reduced Features. In: 2018 3rd International Conference on Computer Science and Engineering (UBMK). pp. 670–672 (2018). https://doi.org/10.1109/UBMK.2018.8566561.
28. Czibula, G., Marian, Z., Czibula, I.G.: Detecting software design defects using relational association rule mining. Knowl. Inf. Syst. 42, 545–577 (2015). https://doi.org/10.1007/s10115-013-0721-z.
29. Nuñez-Varela, A.S., Pérez-Gonzalez, H.G., Martínez-Perez, F.E., Soubervielle-Montalvo, C.: Source code metrics: A systematic mapping study. J. Syst. Softw. 128, 164–197 (2017). https://doi.org/10.1016/j.jss.2017.03.044.
30. Bafandeh Mayvan, B., Rasoolzadegan, A., Ghavidel Yazdi, Z.: The state of the art on design patterns: A systematic mapping of the literature. J. Syst. Softw. 125, 93–118 (2017). https://doi.org/10.1016/j.jss.2016.11.030.
31. Colomo-Palacios, R., Fernandes, E., Sabbagh, M., de Amescua Seco, A.: Human and Intellectual Capital Management in the Cloud: Software Vendor Perspective. J. Univers. Comput. Sci. 18, 1544–1557 (2012).