European Journal of Information Technologies and Computer Science
Vol 4 | Issue 3 | August 2024
ISSN 2736-5492

RESEARCH ARTICLE

Investigating the Role of Software Quality Managers and Software Developers in Creating Code Smells

Abdullah Abdualrahman H. Alzahrani*

ABSTRACT

Software code smells are regarded as signs of potential problems in code design or implementation, they might not affect functionality but impact future maintenance and readability. However, refactoring is a process of code transformation that improves code structure without changing code behavior. This process occurs in the stage of software maintenance where software developers and software quality managers/engineers must work closely in order to identify code smells and plan for refactoring. This research aims to investigate the role of quality managers and software developers in creating code smells.

Keywords: Code smells, Software development, Software maintainability, Software quality.

Submitted: July 09, 2024
Published: August 19, 2024

10.24018/ejcompute.2024.4.3.131

Departments of Computers, Engineering and Computers College, Alqunfuda Umm Al Qura University, Saudi Arabia.

****Corresponding Author:***
e-mail: aahzahrani@uqu.edu.sa

1. Introduction

Software code smells are the red flags in the code. They might not affect the current functionality of the software system [1], [2]. Throughout the software development processes, quality assurance is a crucial activity [3]–[5]. One of the activities in quality assurance is checking or detecting code smells during proactive maintenance [6].

This research focuses on the investigation of the impact made by both software quality managers and software developers towards the existence of code smells. To effectively conduct the data collection, two classes of participants are targeted for this research purpose. The participants will be provided with a set of questions in the form of a questionnaire.

This paper is structured as follows. First, general background and discussion of related work are provided and explained. Secondly, the research questions, along with the research methodology, are explained and elaborated. Thirdly, the main findings of this research are illustrated and discussed. Finally, the conclusions on the findings and the limitations of this research are drawn and explained.

2. Related Work and Background

Software development is not a straightforward task [7]. Different methodologies are introduced and used for software development. These methodologies can be categorized as traditional, dynamic, and adaptive [8], [9]. The most common methodology nowadays is Agile, which is an adaptive methodology [8], [10], [11].

Software quality can be defined as the extent to which the software meets the needs and expectations specified in the requirements of the software [12]–[15]. When considering software quality, many quality attributes are shaping the software quality. One of these quality attributes is Maintainability, which can be regarded as the extent to which the software can be altered for corrections, improvements, and/or changes of requirements [16], [17].

Maintenance is regarded as the process which takes place after the delivery of software in order to fix or change the software for improvement or requirement change [11], [18], [19]. In addition, the most famous maintenance activities are proactive and reactive maintenance. Proactive maintenance is the type of maintenance to perfect the code, whereas reactive maintenance is the type to fix the code [6].

Software code smells are regarded as signs of future issues in code design or implementation; however, software functionalities might not be affected at the time of detection, but code smells might affect the future maintainability and readability of the software code [20]–[24]. Therefore, the process of renovating these code smells might be crucial and is termed as software refactoring, which is one of many activities in software development in specific maintenance activities and it is a code transformation process that improves code structure without changing code behavior [1], [2], [24]–[26].

Many have reviewed the topic of code smells [27]–[32]. The reviews uncover the main topics related to code smells. These topics can be summarized in the following: 1) code smells understanding, 2) code smells impact, 3) code smells catalogs, 4) code smells detection, 5) code smells refactoring. In addition, he explained some of the challenges and related aspects of code smells such as fixing, prediction, and rating of code smells.

Many have introduced their lists or catalogs of code smells [22], [33]–[35] in different types of software projects. In addition, new code smells have been introduced and validated. This shows that code smells are not fixed; however, new types emerge as technology improves and software becomes more complex.

Many have introduced their approaches to detect or predict code smells in the source code [20], [36]–[42]. Some of these approaches are automated with tool support, and others are not. In addition, it is worth noting that machine learning algorithms are common to be used an automated code smell detection. However, a lack of maturity might exist in the approaches that are offered.

Pecorelli *et al.* [43] have offered a new approach for prioritizing code smells according to their criticality. The approach relies on machine learning and developers understanding of the code smells. The authors preferred to name this approach developer-driven. The approach has been compared to two other similar approaches [44], [45], and the authors have reported an outperformance of their approach.

Alfadel *et al*. [46] have studied the relationship between the existence of design patterns and the existence of code smells. The authors considered 20 software design patterns and 13 common code smells in their investigation. The investigation was carried out on 10 open-source codes of Java. The outcomes of this investigation showed a strong relation between some of the design patterns and certain code smells. In addition, the author reported that entities of code that form a design pattern probably do not have code smells.

Muse *et al*. [47] have investigated the Impact of SQL code smells in Data-Intensive Systems. The authors have conducted an empirical investigation on 150 software projects to study the frequency and impact of SQL code smells and compare them with the common code smells in code. The findings of their work were that they noticed a widespread of SQL code smells and noticed that they often occur earlier in the project development phases and remained not fixed.

Han *et al.* [48] have studied the relationship between the identification of code smells and the actions taken by software developers. The authors have conducted an experiment on 2 OpenStack projects and collected over 1000 reviews of code smells out of over 19000 reviews, which have been done by, perhaps, software quality engineers or software professionals who are interested in software quality. The authors concluded that code smells are not often identified when code is checked or reviewed.

Yamashita *et al.* [49] have investigated the factors which might affect the maintenance of software code. The authors consider the work in [50], [51] which offer a set of factors that might affect maintainability of software code. In addition, the authors offer new factors after conducting an empirical study on 6 developers with 4 software projects and discovered a total of 13 factors.

Furthermore, Yamashita *et al*. [52] conducted a survey of 85 software professionals to investigate the understanding of code smells and concluded that a considerable number of professionals showed ignorance of code smells. In addition, the authors reported that professionals showed a lack of interest in building the code with consideration of avoiding the code smells.

3. Research Questions

This research focuses on investigating the role of software quality managers in software developers' creation of code smells. Therefore, two main research questions have been formulated:

1) Are the code smells checks a regular action in the software development processes?
2) What are the reasons behind creating code smells in source code by software developers?

Answering these questions requires conducting surveys to collect data and analyze it to draw conclusions. Therefore, two types of participants are needed to participate in this research, specifically software quality managers and software developers.

4. Methodology

In this research a questionnaire was designed in order to collect the required data to answer the research questions. The questionnaire consists of two parts. The first part collects the consent of participation and some personal information such as gender, experience, and job titles. Based on the participants' responses to the job title question in the first part, the second part displays the appropriate questions in order to collect either the software

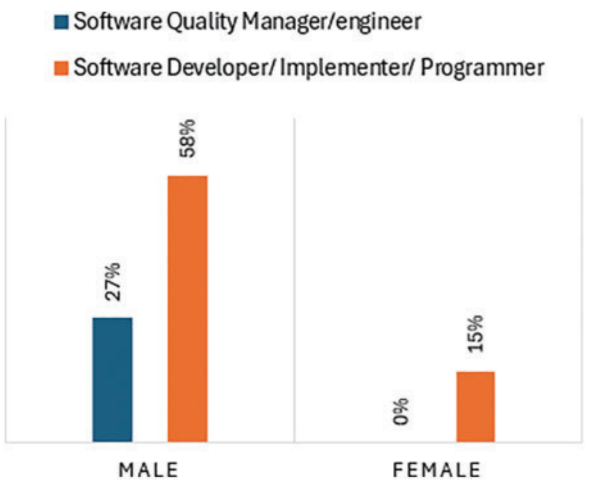

Fig. 1. Division of participants based on gender and job title.

developers' opinions or the software quality managers' opinions. Google Forms was used to create the questionnaire, which was later sent to the intended participants via emails and direct text messaging.

The questionnaire was sent to 150 participants, and the collected responses were 52. Fig. 1 illustrates the distribution of the responses based on the job titles and the gender of participants. It can be seen from Fig. 1 that around 85% of respondents are males. On the other hand, 73% of the respondents are developers.

The questionnaire is built with a Likert scale question, which allows the participants to specify their answers on a scale from 0 to 5, where 0 is never and 4 is always. The respondents are allowed to opt for the not applicable as 5. This scale is for the questions that are related to software quality managers and software developers regarding the Code Smells Checks frequency. However, other questions were asked using a scale from 0 to 4. Finally, the question for the software developers to rate their knowledge on the code smells is in addition using the scale from 0 to 4 where 0 refers to none and 4 refers to excellent.

5. Results and Discussion

In this section, the main findings of our questionnaire will be presented and discussed. The first subsection will show the results of the questions which were answered by the Software quality managers and software developers regarding the frequency of checks for code smells in different programming languages. The second subsection will illustrate the results of the questions that have been answered by Software quality managers regarding their perspectives. The third subsection will show and discuss the results of questions which have been answered by the software developers.

5.1. *Frequency of Checks for Code Smells in Different Programming Languages*

Participants have been asked to specify the scale of frequency of checks for code smells in several programming languages, which are Java, JavaScript, C, C++, C#, Visual Basic, and Python. The following are the presentations and discussions of these findings.

It can be noticed in Fig. 2 that around 37% of software developers participants specified that it is always or often that they conduct a code smell checks in their Java source code. However, the same portion of the software developers are stating that they have not conducted any code smells checks. This might lead to a reduction in the number of Java programmers. On the other hand, 50% of software quality managers specified rare or never as the frequency of code smells checks in source code made using Java programming language.

With regards to checks for code smells in source code made in JavaScript, it is obvious from Fig. 3 that 56% of software developers expressed that they often or always conducted checks. However, software quality managers stated the opposite opinion, as 57% of them stated that they rarely or never carried out code smells checks in source code made using Java programming language.

In Fig. 4, some agreement appears between software developers and software quality managers, as they tend to rarely or never conduct code smell checks in source code made in C programming language, with 44% and 72%, respectively. Moreover, It is important to note that 32% of software developers stated "not applicable" for this matter.

Considering source codes made in C#, as seen in Fig. 5 66% of software developers and 64% of software quality managers have answered with "rarely" or "not applicable." However, 19% of software developers stated that they often or always conducted code smells checks on source code made in C#. On the other hand, 21% of software quality managers stated that they often conduct code smells checks on source code made in C#.

Fig. 6 shows that 71% of software quality managers specified the answer of the frequency they conduct code smells checks on source code made in C++ programming language as "rarely," "never," or "not applicable." On the other hand, 52% of software developers tend to have the same answer. It is worth noting that 29% of software developers stated that they always or often conduct code smells checks for C++ projects.

With regards to checks for code smells in source code made in Visual Basic programming language, Fig. 7 illustrates that software quality managers and software developers have specified the answers for frequency as "never" or "not applicable," with percentages of 64% and 55%, respectively. However, software quality managers and software developers have specified the answers for frequency as "often," with percentages of 21% and 11%, respectively.

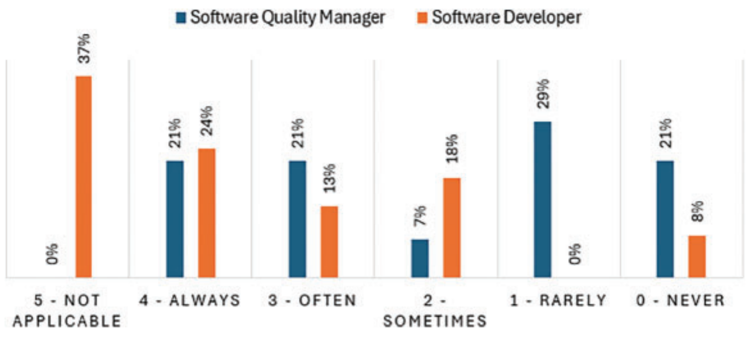

Fig. 2. Frequency of checks for code smells in Java.

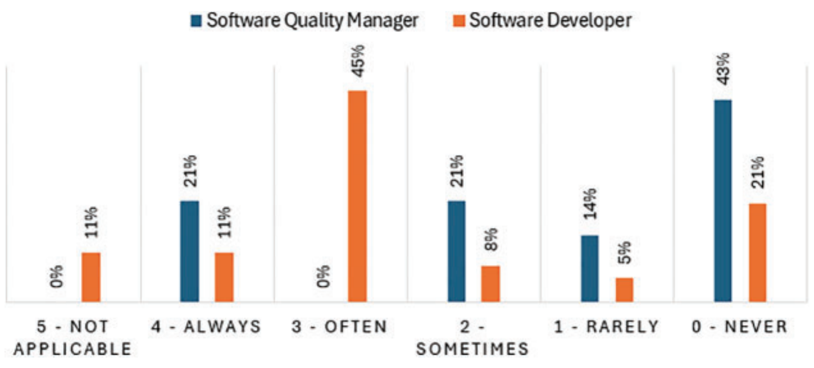

Fig. 3. Frequency of checks for code smells in JavaScript.

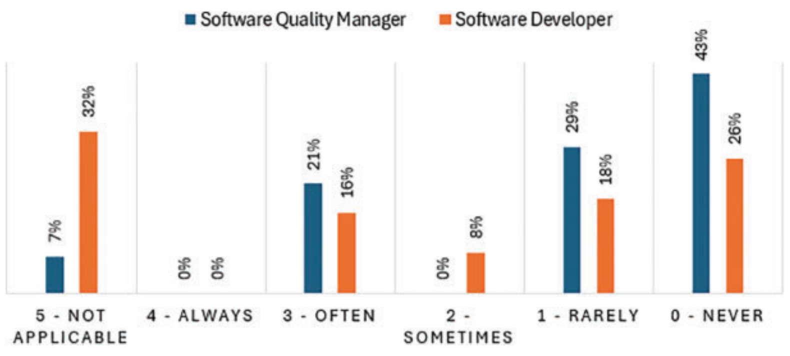

Fig. 4. Frequency of checks for code smells in C.

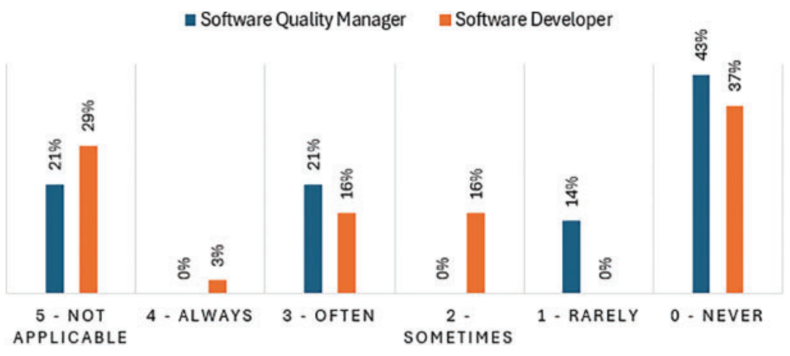

Fig. 5. Frequency of checks for code smells in C#.

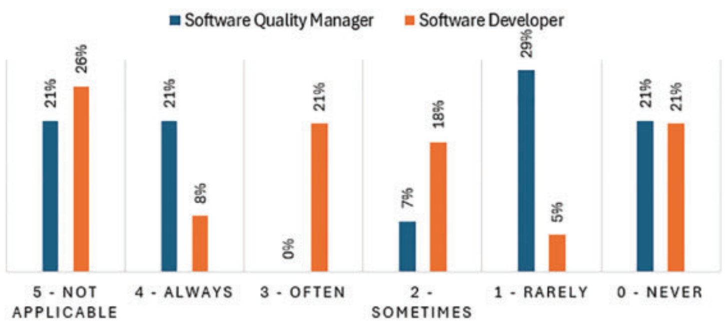

Fig. 6. Frequency of checks for code smells in C++.

As can be seen in Fig. 8, software quality managers and software developers have specified the answers for frequency for checking code smells in projects in Python source codes as "often" or "always," with percentages of 43% and 37%, respectively. Interestingly, the other portion of the software quality managers participants chose "rarely" or "never." However, 42% of software developers specified "never" or "not applicable" as their answers.

5.2. *Software Quality Managers' Perspectives*

In this subsection, the results of the questions which have been provided to the software quality managers are illustrated and discussed. These questions are to measure the impact of software developers in imposing code smells and to measure the impact of software code smells awareness encouraged by software quality managers.

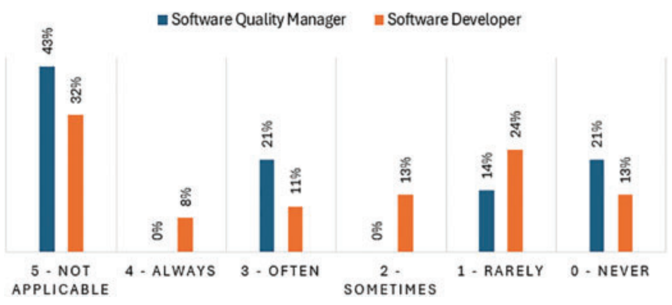

Fig. 7. Frequency of checks for code smells in Visual Basic.

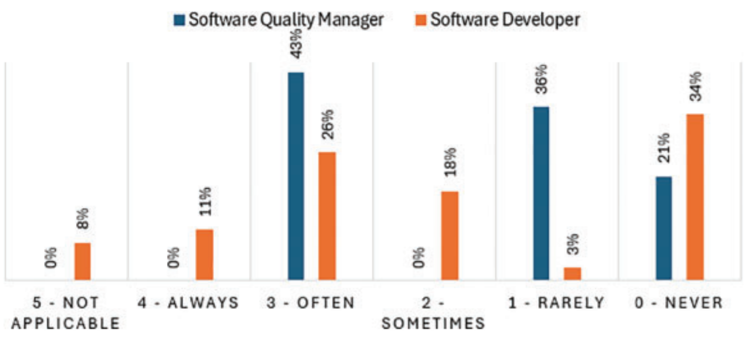

Fig. 8. Frequency of checks for code smells in Python.

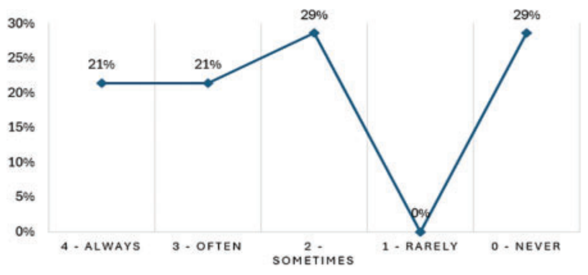

Fig. 9. Software quality managers' opinions on developers creating code smells.

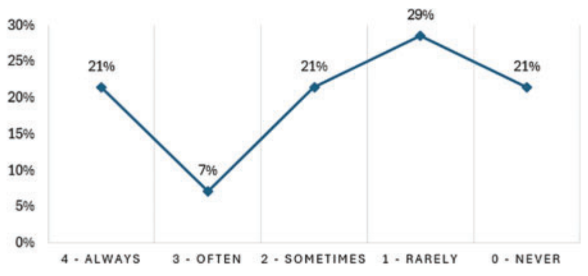

Fig. 10. Frequency of need for Code Smell explanations to developers.

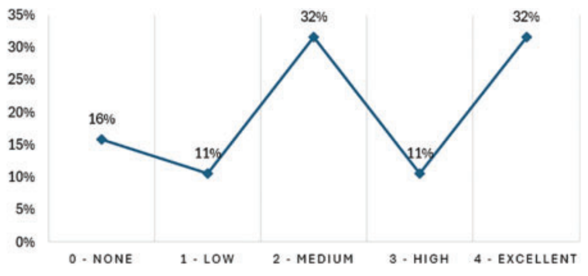

Fig. 11. Software developers rate of knowledge in code smells.

Fig. 9 demonstrates the opinion of the software quality managers on the frequency that software developers contribute to the production of code smells. It is obvious from Fig. 9 that 42% of software quality managers believe that code smells are always or often caused by software developers.

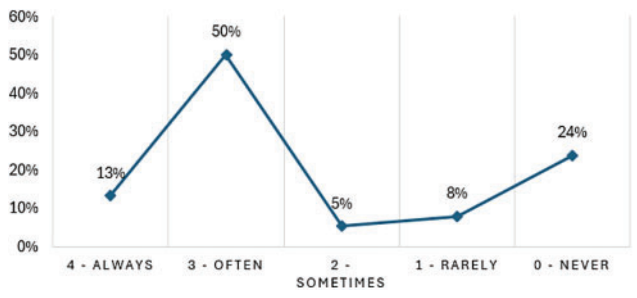

Fig. 12. Frequency of proactive maintenance.

On the other hand, 50% of software quality managers, as can be seen in Fig. 10, admitted that they were rarely or never required to explain or raise awareness of code smells for software developers. The results shown in Figs. 9 and 10 can be linked in order to find correlation between lack of awareness raise and developers making code smells.

5.3. *Software Developers' Perspectives*

In this subsection, the results of the questions which were given to software developers are demonstrated and discussed. The questions are to investigate the software developers' knowledge of code smells and the frequency of proactive and reactive code maintenance that the developers conduct.

Fig. 11 shows that 43% of software developers rated their knowledge of code smells as "high" or "excellent." However, 59% of developers articulated that their knowledge of code smells is "medium" or less. This shows that knowledge of code smells is not high around software developers.

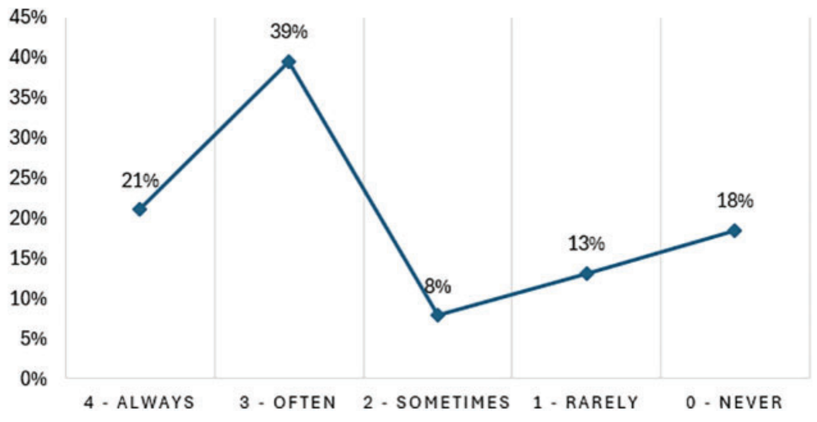

Fig. 13. Frequency of reactive maintenance.

With the difference discussed earlier in this paper between proactive and reactive maintenance, software developers were asked to specify the frequency of conducting each of them. Figs. 12 and 13 illustrate that more than 60% of software developers either always or often conduct software maintenance proactively or reactively. This might link to the results shown in Fig. 11, as 59% of developers articulated that their knowledge of code smells is "medium" or less. It seems that developers cannot distinguish proactive and reactive maintenance.

6. Conclusion

This study shows the results of the investigation of the impact made by both software quality managers and software developers towards the existence of code smells. The results of the investigation can be summarized as follows.

The participants' answers for the frequency of code smells checks for several well-known programming languages were illustrated and discussed. Interesting results have been discovered. In general, code smells checks are not regular action in software development processes. In addition, an interest in code smells check might be noticed in source code made in Java, JavaScript, C++, and Python.

In addition, it can be claimed that there is a relationship between software developers' awareness of code smells, and the judgment of the software quality managers of developers being the reason behind more code smells in source codes.

Furthermore, the software developers' knowledge on code smells and types of maintenance seems not satisfactory and might be a reason behind producing code smells in software source codes. From the discussed results in the previous sections, it can be seen clearly that there is a gap between software development and software quality assurance.

The limitations of this study can be seen in the number of participants. In order to generalize the results and conclusions, more participants are required. However, the current number allows us to hold the lead for the case, and the current conclusions are valid as initial conclusions. In addition, it might be better to have an equal number of software quality managers to the number of software developers.

Acknowledgment

I would like to express my special thanks and gratitude to my university (Umm Al Qura University), which gave me the golden opportunity to conduct this research on the topic of software engineering in specific software quality assurance.

Appendix

TABLE I: Survey on Opinion of Software Quality Manager and Developers

#	Question	Type of response
1.	Select your job role?	Software quality manager Software developer
2.	Select your experience range?	1 year to 6 years 6 years to 10 years Over 10
3.	How often do you go through code smell check in source code made in Java programming language?	0–Never
4.	How often do you go through code smell check in source code made in JavaScript programming language?	1–Rarely
5.	How often do you go through code smell check in source code made in C programming language?	2–Sometimes
6.	How often do you go through code smell check in source code made in C# programming language?	3–Often
7.	How often do you go through code smell check in source code made in C++ programming language?	4–Always
8.	How often do you go through code smell check in source code made in Visual Basic programming language?	5–Not applicable
9.	How often do you go through code smell check in source code made in Python programming language?	

TABLE II: Survey on Opinion of Software Quality Manager

#	Question	Type of response
1.	How often developers create code smells?	0–Never
2.	How often do you need to explain software code smell to developers?	1–Rarely
		2–Sometimes
		3–Often
		4–Always

TABLE III: Survey on Opinion of Developers

#	Question	Type of response
1.	Rate your knowledge of software code smell in general?	0–None
		1–Low
		2–Medium
		3–High
		4–Excellent
2.	How often do you go through Proactive Maintenance? (Address code smells and potential problems before they cause issues)	0–Never 1–Rarely
3.	How often do you go through Reactive Fixes? (Address critical issues like bugs, performance bottlenecks, or major code changes urgently.)	2–Sometimes 3–Often 4–Always

Conflict of Interest

The authors declare that they do not have any conflict of interest.

References

[1] Fowler M. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 2018. Accessed: Jun. 05, 2024. [Online]. Available from: https://books.google.com/books?hl=en&lr=&id=2H1_DwAAQBAJ&oi=fnd&pg=PT14&dq=Improving+the+Design+of+Existing+Code&ots=NhxxvhkXOX&sig=oQBKIvLpmxsliM-AW4FjV0ODSLc.

[2] Brown WH, Malveau RC, McCormick HWS, Mowbray TJ. *AntiPatterns: refactoring Software, Architectures, and Projects in Crisis*. John Wiley & Sons, Inc, 1998. Accessed: Jun. 05, 2024. [Online]. Available from: https://dl.acm.org/doi/abs/10.5555/280487.

[3] Kassie NB, Singh J. A study on software quality factors and metrics to enhance software quality assurance. *Int J Product Qual Manag*. 2020;29(1):24. doi: 10.1504/IJPQM.2020.104547.

[4] Felderer M, Ramler R. Quality assurance for AI-based systems: overview and challenges (Introduction to interactive session). *Softw Qual Future Perspect Softw Eng Qual*. 2021;404:33–42. doi: 10.1007/978-3-030-65854-0_3.

[5] Pargaonkar S. The crucial role of inspection in software quality assurance. *J Sci Technol*. 2021;2(1):70–7.

[6] Canfora G, Cimitile A. Software maintenance. *Handbook Softw Eng Knowl Eng*. 2001;1:91–120. doi: 10.1142/9789812389718_0005.

[7] Basili VR. Software development: a paradigm for the future. *[1989] Proceedings of the Thirteenth Annual International Computer Software & Applications Conference*, pp. 471–85, IEEE, 1989. Accessed: Jun. 05, 2024. [Online]. Available from: https://ieeexplore.ieee.org/abstract/document/65127/.

[8] Abrahamsson P, Salo O, Ronkainen J, Warsta J. Agile software development methods: review and analysis. arXiv. 2017. Accessed: Jun. 05, 2024. [Online]. Available from: http://arxiv.org/abs/1709.08439.

[9] Sawyer S, Guinan PJ. Software development: processes and performance. *IBM Syst J*. 1998;37(4):552–69.

[10] Barroca L, Dingsøyr T, Mikalsen M. Agile transformation: a summary and research agenda from the first international workshop. In *Agile Processes in Software Engineering and Extreme Programming—Workshops*. Hoda R. Ed. Cham: Springer International Publishing, 2019. pp. 3–9. doi: 10.1007/978-3-030-30126-2_1.

[11] Sommerville I. *Software Engineering*. 10th ed. Boston: Pearson, 2015.

[12] Miguel JP, Mauricio D, Rodriguez G. A review of software quality models for the evaluation of software products. *Int J Softw Eng Appl*. 2014 Nov;5(6):31–53. doi: 10.5121/ijsea.2014.5603.

[13] IEEE Standard Boaed. *IEEE Standard Glossary of Software Engineering Terminology*. New York, N.Y: Institute of Electrical and Electronics Engineers, 1990.

[14] Mamone S. The IEEE standard for software maintenance. *ACM SIGSOFT Softw Eng Notes*. 1994 Jan;19(1):75–6. doi: 10.1145/181610.181623.

[15] Samadhiya D, Wang S-H, Chen D. Quality models: role and value in software engineering. *2010 2nd International Conference on Software Technology and Engineering*, pp. V1–320, IEEE, 2010. Accessed: Jun. 05, 2024. [Online]. Available from: https://ieeexplore.ieee.org/abstract/document/5608852/.

[16] ISO 9241-11. ISO 9241-11:2018(en), Ergonomics of human-system interaction—Part 11: Usability: Definitions and concepts. 2018. Accessed: Jan. 25, 2020. [Online]. Available from: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en.

[17] International Organization for Standardization. *I. ISO, I. FDIS. 25020. Software Engineering-Software Product Quality Requirements and Evaluation (SQuaRE)-Measurement Reference Model and Guide*. Geneva, Switzerland: ISO; 2007.

[18] Despa ML. Comparative study on software development methodologies. *Database Syst J*. 2014;5(3):37–56. Accessed: Jun. 05, 2024. [Online]. Available from: http://dbjournal.ro/archive/17/17.pdf#page=38.

[19] Chapin N, Hale JE, Khan KMd, Ramil JF, Tan W. Types of software evolution and software maintenance. *J Softw Maint Evol Res Pract*. Jan. 2001;13(1):3–30. doi: 10.1002/smr.220.

[20] Di Nucci D, Palomba F, Tamburri DA, Serebrenik A, De Lucia A. Detecting code smells using machine learning techniques: Are we there yet?. *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 612–21. IEEE, 2018. Accessed: Jun. 05, 2024. [Online]. Available from: https://ieeexplore.ieee.org/abstract/document/8330266/.

[21] Fontana FA, Lenarduzzi V, Roveda R, Taibi D. Are architectural smells independent from code smells? An empirical study. *J Syst Softw*. 2019;154:139–56.

[22] Gesi J, Liu S, Li J, Ahmed I, Nagappan N, Lo D, *et al.* Code smells in machine learning systems. *arXiv*. Mar. 01, 2022; Accessed: Jun. 05, 2024. [Online]. Available from: http://arxiv.org/abs/2203.00803.

[23] Hall T, Zhang M, Bowes D, Sun Y. Some code smells have a significant but small effect on faults. *ACM Trans Softw Eng Methodol*. Sep. 2014;23(4):1–39. doi: 10.1145/2629648.

[24] Khomh F, Di Penta M, Gueheneuc Y-G. An exploratory study of the impact of code smells on software change-proneness. *2009 16th Working Conference on Reverse Engineering*, pp. 75–84. IEEE, 2009. Accessed: Jun. 05, 2024. [Online]. Available from: https://ieeexplore.ieee.org/abstract/document/5328703/.

[25] Baqais AAB, Alshayeb M. Automatic software refactoring: a systematic literature review. *Softw Qual J*. Jun. 2020;28(2):459–502. doi: 10.1007/s11219-019-09477-y.

[26] Mens T, Tourwé T. A survey of software refactoring. *IEEE Trans Softw Eng*. 2004;30(2):126–39.

[27] Lacerda G, Petrillo F, Pimenta M, Guéhéneuc YG. Code smells and refactoring: a tertiary systematic review of challenges and observations. *J Syst Softw*. 2020;167:110610.

[28] Pereira Dos Reis J, Brito E Abreu F, De Figueiredo Carneiro G, Anslow C. Code smells detection and visualization: a systematic literature review. *Arch Comput Methods Eng*. Jan. 2022;29(1):47–94. doi: 10.1007/s11831-021-09566-x.

[29] Lewowski T, Madeyski L. Code smells detection using artificial intelligence techniques: a business-driven systematic review. *Dev Inf Knowl Manag Bus Appl*. 2022;377:285–319. doi: 10.1007/978-3-030-77916-0_12.

[30] Agnihotri M, Chug A. A systematic literature survey of software metrics, code smells and refactoring techniques. *J Inf Process Syst*. 2020;16(4):915–34.

[31] Piotrowski P, Madeyski L. Software defect prediction using bad code smells: a systematic literature review. in *Data-Centric Business and Applications*, vol. 40. In *Lecture Notes on Data Engineering and Communications Technologies*. vol. 40, Poniszewska-Marańda A, Kryvinska N, Jarząbek S, Madeyski L, Eds. Cham: Springer International Publishing, 2020, pp. 77–99. doi: 10.1007/978-3-030-34706-2_5.

[32] Lewowski T, Madeyski L. How far are we from reproducible research on code smell detection? A systematic literature review. *Inf Softw Technol*. 2022;144:106783.

[33] Zhang H, Cruz L, Van Deursen A. Code smells for machine learning applications. *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*, pp. 217–28, Pittsburgh Pennsylvania: ACM, May 2022. doi: 10.1145/3522664.3528620.

[34] Van Oort B, Cruz L, Aniche M, Van Deursen A. The prevalence of code smells in machine learning projects. In *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE, 2021. pp. 1–8. Accessed: Jun. 05, 2024. [Online]. Available from: https://ieeexplore.ieee.org/abstract/document/9474395/.

[35] Siddiq ML, Majumder SH, Mim MR, Jajodia S, Santos JC. An empirical study of code smells in transformer-based code generation techniques. *2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pp. 71–82. IEEE, 2022. Accessed: Jun. 05, 2024. [Online]. Available from: https://ieeexplore.ieee.org/abstract/document/10006873/.

[36] Emden E Van, Moonen L. Java quality assurance by detecting code smells. *Ninth Working Conference on Reverse Engineering, 2002. Proceedings*, pp. 97–106. IEEE, 2002. Accessed: Jun. 05, 2024. [Online]. Available from: https://ieeexplore.ieee.org/abstract/document/1173068/.

[37] Kovačević A, Slivka J, Vidaković D, Grujić K, Luburić N, Prokić S, *et al.* Automatic detection of long method and god class code smells through neural source code embeddings. *Expert Syst Appl*. Oct. 2022;204:117607. doi: 10.1016/j.eswa.2022.117607.

[38] Mhawish MY, Gupta M. Predicting code smells and analysis of predictions: using machine learning techniques and software metrics. *J Comput Sci Technol*. Nov. 2020;35(6):1428–45. doi: 10.1007/s11390-020-0323-7.

[39] Lin T, Fu X, Chen F, Li L. A novel approach for code smells detection based on deep learning. *Appl Cryptogr Comput Commun*. 2021;386:171–4. doi: 10.1007/978-3-030-80851-8_12.

[40] Dewangan S, Rao RS, Mishra A, Gupta M. A novel approach for code smell detection: an empirical study. *IEEE Access*. 2021;9:162869–83.

[41] Sharma T, Efstathiou V, Louridas P, Spinellis D. Code smell detection by deep direct-learning and transfer-learning. *J Syst Softw*. 2021;176:110936.

[42] Walker A, Das D, Cerny T. Automated code-smell detection in microservices through static analysis: aa case study. *Appl Sci*. 2020 Jan;10(21):21. doi: 10.3390/app10217800.

[43] Pecorelli F, Palomba F, Khomh F, Lucia ADe. Developer-driven code smell prioritization. *Proceedings of the 17th International Conference on Mining Software Repositories*, pp. 220–31, Seoul Republic of Korea: ACM. Jun. 2020. doi: 10.1145/3379597.3387457

[44] Vidal SA, Marcos C, Díaz-Pace JA. An approach to prioritize code smells for refactoring. *Autom Softw Eng*. Sep. 2016;23(3):501–32. doi: 10.1007/s10515-014-0175-x.

[45] Fontana FA, Zanoni M. Code smell severity classification using machine learning techniques. *Knowl-Based Syst*. 2017;128:43–58.

[46] Alfadel M, Aljasser K, Alshayeb M. Empirical study of the relationship between design patterns and code smells. *Plos One*. 2020 Apr;15(4):e0231731. doi: 10.1371/journal.pone.0231731.

[47] Muse BA, Rahman MM, Nagy C, Cleve A, Khomh F, Antoniol G. On the prevalence, impact, and evolution of SQL code smells in data-intensive systems. *Proceedings of the 17th International Conference on Mining Software Repositories*, pp. 327–38, Seoul Republic of Korea: ACM. Jun. 2020. doi: 10.1145/3379597.3387467.

[48] Han X, Tahir A, Liang P, Counsell S, Luo Y. Understanding code smell detection via code review: a study of the openstack community. *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, pp. 323–34. IEEE, 2021. Accessed: Jun. 05, 2024. [Online]. Available from: https://ieeexplore.ieee.org/abstract/document/9463021/.

[49] Yamashita A, Moonen L. Do code smells reflect important maintainability aspects?. *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pp. 306–15. IEEE, 2012. Accessed: Jun. 05, 2024. [Online]. Available from: https://ieeexplore.ieee.org/abstract/document/6405287/.

[50] Markus P, Florian D. How to effectively define and measure maintainability. *Softw Eng J*. 2008;6:303–16.

[51] Anda B. Assessing software system maintainability using structural measures and expert assessments. *2007 IEEE International Conference on Software Maintenance*, pp. 204–13. IEEE, 2007. Accessed: Jun. 06, 2024. [Online]. Available from: https://ieeexplore.ieee.org/abstract/document/4362633/.

[52] Yamashita A, Moonen L. Do developers care about code smells? An exploratory survey. *2013 20th Working Conference on Reverse Engineering (WCRE)*, pp. 242–51. IEEE, 2013. Accessed: May 21, 2024. [Online]. Available from: https://ieeexplore.ieee.org/abstract/document/6671299/.