Perceptions and Difficulties of Software Engineering Students in Code Smells Refactoring

Carla Bezerra
Federal University of Ceará
Quixadá, Brazil
carlailane@ufc.br

Humberto Damasceno
Federal University of Ceará
Quixadá, Brazil
hdamasceno1998@gmail.com

João Teixeira
Federal University of Ceará
Quixadá, Brazil
joao.nascimento@alu.ufc.br

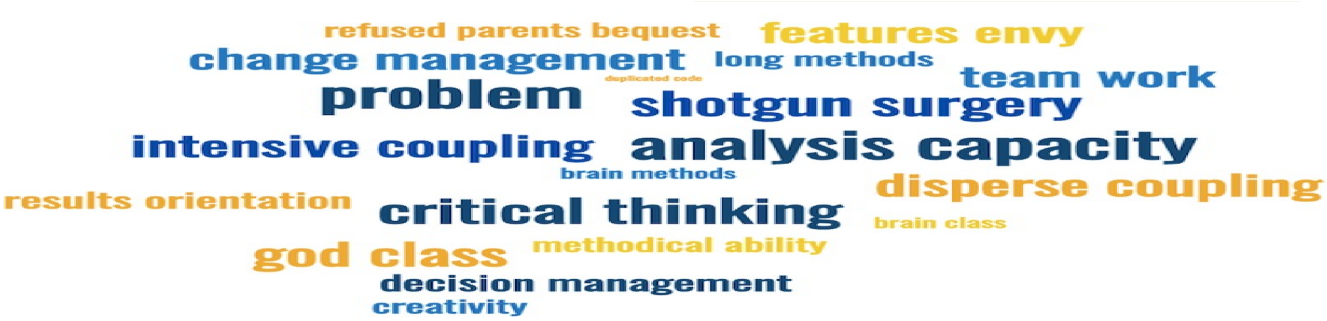

Figure 1: Word cloud of skills acquired by students in code smells refactoring.

ABSTRACT

Code smells are bad code structures that can harm the quality and maintainability of software. To remove these bad structures, developers use refactoring techniques. Refactoring helps code be easier to understand and modify by eliminating potential problems and improving the software quality. Most refactoring activities are usually performed manually and undisciplined, which can cause code degradation. Currently, concepts, practices and tools of refactoring and code smells are little discussed in undergraduate computing courses. This problem is reflected in the software industry, which generally does not use refactoring practices to improve code readability and maintainability. In this context, we present in this paper an experience report on teaching the practice of code smells refactoring, investigating the following topics: (i) skills acquired in the practice of code smells refactoring, (ii) difficulties encountered in code smells refactoring and (iii) students' perception of code smells refactoring practices. The study was carried out in two undergraduate classes with 20 students in the course of software quality. Our findings contribute to replicating the practice in other courses that want to delve into code smell refactoring and contribute to the use of refactoring practices and code quality evaluation in the industry.

KEYWORDS

code smells, refactoring, education

1 INTRODUCTION

Code smells can indicate problems related to aspects of code quality, such as readability and modifiability of software [7]. Refactoring can remove code smells and can have a direct impact on code quality [20]. Fowler [7] defines refactoring as a set of small changes that are made to the internal structure of the code without changing the external behavior. Refactorings can serve different purposes. For example, *Move Method* [7] is often applied by developers to eliminate the code smell *Feature Envy*. This refactoring involves moving a method from one class to another to remove excessive dependencies between the two classes. Refactoring can be done manually, or it can be supported by semi-automated tools [2]. However, refactoring is usually performed manually or undisciplined [5, 9, 13]. There is evidence in the literature that suggests unwanted effects of misguided refactorings, such as code degradation, especially for more complex refactorings [4, 10].

In the systematic review carried out by Lacerda et al. [14] on code smells and refactoring, the authors indicate that the teaching of code smells refactoring techniques and tools is still an open problem. Some studies even report the teaching of refactoring and gamification of these concepts [1, 3, 6, 11, 12]. However, it is known that these concepts are usually not addressed in the disciplines of computing courses. This is reflected in the software industry, as research by Golubev et al. [8] indicates that developers are not familiar with refactoring and IDEs.

This paper presents an experience report of the teaching and practice of code smells refactoring, investigating students' perceptions, skills and difficulties in refactoring code smells. The practice took place in two groups of a software quality course, in which concepts of code smells and refactoring were presented. The practice consisted of manual code smell refactoring in Java code projects. The main results of teaching and practicing code smells refactoring are: (i) students acquire several personal and interpersonal skills after refactoring code smells, especially problem-solving, analytical skills and critical thinking skills; (ii) the refactoring of a code smell can lead to the emergence of another code smell; and (iii) according to the student's perception, the refactoring of code smells improves quality aspects with code readability, but can worsen other aspects such as code complexity and generate new code smells.

2 RELATED WORK

Abid et al. [1] report the teaching and practice of code refactoring from the discussion of two different approaches used in the execution of a course on refactoring and design patterns. The study was carried out with 23 undergraduate and graduate students with theoretical and practical workloads. As the main results, the authors found that making functional changes to the code before executing the code refactoring is more beneficial for teaching refactoring, as it facilitates understanding refactoring opportunities and contributes to obtaining higher quality refactorings. Keuning et al. [12] presented a tutoring system that enables students to exercise code improvement through small programs that are already working properly. The system is based on rules that were obtained through studies carried out with professors in the area and by professional tools. These rules are responsible for defining how the code should be rewritten without changing its functionality. As a result, the authors specified the system's design, presented sessions of practical examples, and discussed the impact of their contributions. Furthermore, motivating results were obtained by applying an exploratory study carried out with students without programming experience. As with Abid et al. [1] and Keuning et al. [12], our work also reports the teaching of code refactoring based on theoretical and practical activities. However, instead of focusing on the analysis of the teaching methodology, we focused our study on students' perception of refactoring activities, especially in what concerns refer to skills acquired, difficulties encountered and main practices used.

Agrahari and Chimalakonda [3], presented a game (Refactor4Green) developed to promote the teaching of code smells and refactoring. Such a game focuses on code smells related to energy efficiency and, in short, contains learning cards with the definition of some code smells and their refactoring strategies, followed by challenges in the form of objective quizzes. Similarly, dos Santos et al. [6] presented the CleanGame, a gamified platform for practicing code smell identification. This platform has two major modules, one for students to review the main concepts about various code smells and another that focuses on practical tasks of identifying code smells in the source code of java systems. In both works, the evaluation carried out with students shows that gamification can be helpful for teaching code smells and refactoring. As well as these works, our work also reports the practice of teaching code smells and refactoring and how it contributes to the student's experience.

3 METHODOLOGY

This study aims to investigate how the learning and practice of these concepts influence programming skills and what difficulties students encounter in refactoring these bad code structures.

3.1 Research Questions

We have elaborated the research questions (**RQs**) to guide our study:

***RQ*$_1$** – *What skills did students acquire by code smells refactoring?*

***RQ*$_2$** – *What are the main difficulties faced by students in the practice of code smells refactoring?*

***RQ*$_3$** – *What is students' perception about the practice of code smells refactoring?*

3.2 Students

We performed the study in two software quality classes with students in the semesters of 2020.2 (**C1**) and 2021.1 (**C2**). A total of 20 students effectively participated in the study. Table 1 shows the division of students by class.

Table 1: Students divided by class

Class	Students
Class 1 (C1)	P1, P2, P3, P4, P5, P6, P7, P8, P9
Class 2 (C2)	P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20

Table 2 presents the profile of students who participated in the practice of code smells refactoring collected via questionnaire. We do not consider all students in the class, as many students locked the course in the pandemic and some students did not complete the practice.

Table 2: Students profile

ID	Undergraduate semester	Experience in years	Refactoring	Code Smells
P1	4	1 year	None	None
P2	4	4 years	Minimum	None
P3	5	1 year	Basic	Intermediary
P4	6	1 year	Intermediary	Basic
P5	6	2 years	Intermediary	Intermediary
P6	6	2 years	Minimum	Minimum
P7	6	3 years	Basic	Intermediary
P8	8	3 years	Minimum	None
P9	4	3 years	Basic	None
P10	7	5 years	Intermediary	Intermediary
P11	7	4 years	Minimum	None
P12	5	3 years	Minimum	Basic
P13	7	Less than one year	Intermediary	Basic
P14	8	3 years	Intermediary	Basic
P15	7	2 years	Basic	None
P16	7	3,5 years	Minimum	Minimum
P17	10	0 years	None	None
P18	7	3 years	Basic	Basic
P19	8	1 year	Basic	None
P20	6	1 year and 8 months	None	None

3.3 Project Design

To perform code smells refactoring practice with students, we performed the following steps:

Step 1: Training of theoretical and practical content. We teach theoretical classes in the course of software quality to bring the necessary concepts to carry out the practice of code smells refactoring. We dedicated three weeks of theoretical classes (12 hours) to the concepts of refactoring and code smells, including training the JSpIRIT tool to detect code smells. This content corresponds to a part of the course content taught remotely by Google Meet.

Step 2: Presentation of the code smells refactoring practice. After passing the concepts and tools to the students, we presented the code smells refactoring practical activity, which consisted of the final work of the course. In teams of 2 people or individually, students would have to select a Java code project to identify and refactor the code smells of the project. Each team had to refactor at least 5 different types of code smells, wherein at least 40 occurrences of code smells of these 5 types would be refactored. The delivery of the practice was divided: (i) removing all occurrences of at least

two types of code smells that added together would have at least 20 occurrences, and (ii) removing the rest of the missing code smells.

For project selection, students used the following defined criteria: (i) JAVA language, (ii) having at least 1,200 code lines, and (iii) presenting at least five types of code smells with a total of 40 occurrences or more. Students used their projects or open-source projects selected on GitHub. Table 3 presents the characterization of the projects selected by the students. The projects are available in our dataset[1].

Table 3: Project characterization

System	Teams	Students	# of classes	# LOC
S1	T1	P1, P2	239	2145
S2	T2	P3, P4	112	6599
S3	T3	P5, P6	183	8384
S4	T4	P7, P8	163	8411
S5	T5	P9	82	4946
S6	T6	P10	93	5676
S7	T7	P11, P12	19	2076
S8	T8	P13	152	4923
S9	T9	P14, P15	182	13172
S10	T10	P16, P17	56	4525
S11	T11	P18	320	17081
S12	T12	P19	221	7702
S13	T13	P20	52	1369

Step 3: Code smell detection and refactoring. This step corresponds to the first delivery of the work. The students used the JSpIRIT tool to detect at least 5 types of code smells with at least 40 occurrences in the selected projects. This tool identifies a total of 10 types of code smells. We use the JSpIRIT tool to detect code smells due to its high level of accuracy [19] and because it has already been used in similar research [16, 17]. The second work delivery consisted of partial refactoring of the selected code smells. Students should refactor at least 20 occurrences of code smells. For the refactoring of each code smell, the students described the refactoring techniques used in the report. The third and final deliverable consisted of refactoring all remaining occurrences of code smells. The refactoring techniques used in the rest of the code smells are also complemented. Each team should present the design and rationale of the refactorings. Table 4 presents the code smells considered in this study, because the code smells were detected and refactored in the projects.

Table 4: Code smells detected in this study

Code Smells	Description
Feature Envy	Method "envying" other classes' features [7]
God Class	Too many software features into a class. It tend to be very large and hard to read and understand [7]
Dispersed Coupling	Method that calls too many methods [7]
Intensive Coupling	Method that depends too much from a few others [7]
Shotgun Surgery	Method whose changes affect many methods [7]
Long Method	Too long and complex method [7]
Brain Method	Long and complex method that centralizes the intelligence of a class [15]
Refused Parent Bequest	Subclass that doesn't use its superclass's protected methods [7]

Step 4: Application of the perception questionnaire of code smells refactorings. Finally, we applied a questionnaire to classes C1 and C2 to understand what were the difficulties encountered in refactoring code smells and the importance of refactoring code smells in the perception of the students themselves. All data collected from questionnaire data are available in our dataset[2].

4 RESULTS

4.1 RQ_1: Skills acquired in code smells refactoring

We solved $\mathbf{RQ_1}$ by evaluating student responses generated from an online questionnaire. The skills were extracted from the systematic review on skills of software engineers [18]. From this, we performed a qualitative analysis through the collected responses, and we obtained 9 main categories of personal and interpersonal skills acquired by students after code smells refactoring, which are displayed in Table 5. The first column displays all categories, the second column presents the skills description and the third contains the number of students who have acquired these skills.

Table 5: Skills acquired by students after code smells refactoring [18]

Skill	Description	Total
Analytical skills	The ability to understand and explain each part of a whole, to know better than nature, functions, causes, among others	18
Problem solving	The ability to understand, articulate and solve complex problems	17
Critical thinking	The ability to determine carefully and deliberately accepted, refutation or suspension of the trial about a particular piece of information	12
Change management	The ability to propose and/or take any action without need for others to come to ask or say	7
Team work	ability of an individual who is good at working closely with other people	7
Decision-making	The ability to make sensible decisions based on available information	4
Results orientation	The ability to achieve and/or exceed. Sales goals and/or objectives	4
Methodical	The ability to use a set of steps, neatly, arranged, set by methods (techniques) to solve a particular issue or problem	3
Creativity	Not defined	3

Analyzing the information contained in the Table 5, it can be seen that analytical skills (18), problem solving (17) and critical thinking (12) were the most cited by students in the questionnaire. Therefore, this infers that the code smells refactoring activity positively impacted students in acquiring new skills.

> **Finding 1**: The code smells refactoring activity provided new skills for students, especially to problem solving, analytical skills and critical thinking skills.

In the Table 5, it is possible to notice that the students also improved their interpersonal skills. Considering that skills such as teamwork (7), decision making (4), results orientation (4) and creativity (3). Therefore, it can be seen that the code smells refactoring activity improved the students' interpersonal skills.

> **Finding 2**: Code smells refactoring activity is beneficial for students' interpersonal skills.

Implications of $\mathbf{RQ_1}$. Our findings indicate that applying code smells refactoring activity to students was beneficial. Considering

[1]https://github.com/leanresearchlab/VEM-2022

[2]https://github.com/leanresearchlab/VEM-2022

that both personal and interpersonal skills improved after refactoring code smells. Such benefits underscore the importance of teaching about code smells in computer courses. It is a highly relevant topic for the academic community mainly because of improving students' skills in the industry.

4.2 RQ_2: Difficulties identified by students in the practice of refactoring

We analyzed the $\mathbf{RQ_2}$ performing a qualitative analysis of the students' answers about the difficulties identified from the practice of refactoring the code smells. From the answers collected, we identified 4 categories of difficulties that most occurred among the students: (i) difficulty in understanding the source code; (ii) emergence of new code smells after refactoring a code smell; (iii) difficulty choosing the refactoring technique; and, (iv) lack of reference material to aid refactoring. Table 6 presents the categories identified after analyzing the developers' responses. The first column lists the categories found, and the second contains the number of students who had the respective difficulties listed during the code smells refactoring.

Table 6: Difficulties identified by students in the practice of refactoring

Categories	Students
Difficulties understanding the source code	P4, P5, P8, P10, P11, P19
Emergence of new code smells after refactoring a code smell	P1, P2, P9, P12, P13, P14
Difficulty choosing the refactoring technique	P3, P5, P10, P11, P20
Lack of reference material to aid refactoring	P6, P7, P16

Analyzing the Table 6, we can highlight that 6 students reported that they had difficulties refactor a code smell without other code smells occurring after the refactoring. With this, it is possible to notice that the code smells refactoring process is susceptible to new code smells. Some student reports support this statement:

P1:"After solving one code smells, another one appeared."

P12:"I couldn't remove the code smell without others appearing."

P13:"In some cases, when refactoring a certain smell, others of another type emerged."

Finding 3: Refactoring a code smell can lead to the emergence of other code smells of different types.

We also identified through the information contained in Table 6 that some students are present in more than one category of difficulties to refactor code smells. As a result, we can see relationships between the categories of difficulties encountered by students. The relationship we highlight is that the more difficult the source code is to understand, the more complicated it will be to refactor the code smell. Some student reports corroborate this relationship:

P10:" I had difficulties understanding the source code, and consequently performing the refactoring."

D11:"For me, the fact of taking a project with an organization of the code that is not familiar, made the refactoring more complicated.."

Finding 4: The harder it is to interpret the source code, the more complicated it is to refactor code smells.

Implications of $\mathbf{RQ_2}$. Our findings indicate that the more difficult it is to understand the source code, the more complicated the application of the code smells refactoring technique will be. It is noticeable that students do not feel safe refactoring a certain code smell when they do not understand the source code. Thus, it is important to emphasize the importance of good programming practices to facilitate understanding the source code. Furthermore, removing a code smell can lead to the emergence of a new code smell. As a result, it is essential to verify and validate all refactorings to mitigate possible new code smells.

4.3 RQ_3: Students' perception of refactoring practices

We approached $\mathbf{RQ_3}$ by analyzing the data collected through a questionnaire. We asked students about their positive or negative perceptions during and after refactoring the code smells. Table 7 presents the perceptions reported by students.

Table 7: Students' perceptions about the impacts of code smells refactoring

Perceptions	Students
Improved	P1, P6, P7, P8, P12, P14, P18, P19
Improved in parts	P2, P3, P9, P11, P13, P16, P17, P20
Indifferent	P10, P15
Harmed in parts	P2, P3, P9, P11, P13, P16, P17, P20
Harmed	P4, P5

Observing the Table 7, it is possible to notice that a considerable portion of the students (8) judged that the code smells refactoring had a positive impact on the quality of the system. On the other hand, only 2 students considered the code smells refactoring to be totally harmful to the system quality. This suggests that the code smells refactoring tends to have a positive impact on the system quality, according to the students' perceptions. Some student reports corroborate this statement:

P6:"I noticed that refactoring code smells had improved code readability by simplifying and rearranging classes, methods and attributes."

P18:"The code smells refactoring activity has improved my ability to visually detect code smells that need refactoring."

Finding 5: Part of the students had the perception of an improvement in the system's quality after the code smells refactorings, such as readability.

Continuing the analysis of the Table 7, part of the students (8) considered that the code smells refactoring had a positive impact in some cases. Still, in other cases, it harmed the internal quality attributes. This infers that the code smells refactoring for developers and has a negative impact in some aspects. However, it also affects quality negatively. Some reports strengthen this statement:

P3:"Code smells refactoring significantly improved cohesion, but the other attributes got worse overall."

P9:"Code smells refactoring improved code readability but not complexity."

P16:"In my project the practice had a balanced performance, for some scenarios it had an improvement, for others it got worse, with new code smells appearing."

P20:"Code smells refactoring made some attributes worse, and improved others."

Finding 6: Some of the students considered the refactoring of code smells improved the internal quality attributes of the system in some aspects but worsened other aspects, such as: new code smells appeared and worsened the complexity.

Implications of RQ$_3$. Our findings infer that most students reported that code smells refactoring improved the system's quality. On the other hand, we also noticed that the perceptions of other students indicated that the refactoring of code smells positively impacted the internal quality attributes in some aspects but negatively in other criteria, such as increased complexity. Due to these results, we emphasize to students that the occurrence of any code smell in a software project can negatively impact it. And that refactoring of such smells must be performed immediately after detection.

5 FINAL REMARKS

Our study investigated the skills, difficulties and perceptions of Software Engineering students in the practice of code smells refactoring. We considered 8 types of code smell and 20 students carried out the refactoring of code smells for 2 months in 2 classes of the software quality course. Our main results were: (i) students acquired several personal and interpersonal skills after refactoring code smells, especially problem-solving, analytical skills and critical thinking skills; (ii) the refactoring of a code smell can lead to the emergence of another code smell; and, (iii) according to the student's perception, the refactoring of code smells improves quality aspects with code readability, but can worsen other aspects such as code complexity and generate new code smells.

As limitations of the study, we have: (i) some teams selected projects from GitHub that they had not had contact with before, making the activity and refactoring difficult; (ii) lack of student experience; (iii) we only use one tool to detect code smells; and, (iv) the students only refactored some code smells from the projects and we did not collect more detailed information from these refactorings.

In future work, we intend to: (i) use other tools to detect other code smells that were not included in the study; (ii) use automatic refactoring tools to remove code smells; and, (iii) perform the practice also for refactoring co-occurrences of code smells.

REFERENCES

[1] Shamsa Abid, Hamid Abdul Basit, and Naveed Arshad. 2015. Reflections on Teaching Refactoring: A Tale of Two Projects. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (Vilnius, Lithuania) *(ITiCSE '15)*. Association for Computing Machinery, New York, NY, USA, 225–230. https://doi.org/10.1145/2729094.2742617

[2] Mansi Agnihotri and Anuradha Chug. 2020. A Systematic Literature Survey of Software Metrics, Code Smells and Refactoring Techniques. *Journal of Information Processing Systems* 16, 4 (2020).

[3] Vartika Agrahari and Sridhar Chimalakonda. 2020. Refactor4Green: a game for novice programmers to learn code smells. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 324–325.

[4] Jehad Al Dallal and Anas Abdin. 2017. Empirical evaluation of the impact of object-oriented code refactoring on quality attributes: A systematic literature review. *IEEE Transactions on Software Engineering* 44, 1 (2017), 44–69.

[5] Vahid Alizadeh, Mohamed Amine Ouali, Marouane Kessentini, and Meriem Chater. 2019. RefBot: Intelligent Software Refactoring Bot. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 823–834. https://doi.org/10.1109/ASE.2019.00081

[6] Hoyama Maria dos Santos, Vinicius H. S. Durelli, Maurício Souza, Eduardo Figueiredo, Lucas Timoteo da Silva, and Rafael S. Durelli. 2019. CleanGame: Gamifying the Identification of Code Smells. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering* (Salvador, Brazil) *(SBES 2019)*. Association for Computing Machinery, New York, NY, USA, 437–446. https://doi.org/10.1145/3350768.3352490

[7] Martin Fowler. 2018. *Refactoring: improving the design of existing code.* Addison-Wesley Professional.

[8] Yaroslav Golubev, Zarina Kurbatova, Eman Abdullah AlOmar, Timofey Bryksin, and Mohamed Wiem Mkaouer. 2021. One Thousand and One Stories: A Large-Scale Survey of Software Refactoring. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 1303–1313. https://doi.org/10.1145/3468264.3473924

[9] Péter Hegedűs, István Kádár, Rudolf Ferenc, and Tibor Gyimóthy. 2018. Empirical evaluation of software maintainability based on a manually validated refactoring dataset. *Information and Software Technology* 95 (2018), 313–327.

[10] Satnam Kaur and Paramvir Singh. 2019. How does object-oriented code refactoring influence software quality? Research landscape and challenges. *Journal of Systems and Software* 157 (2019), 110394.

[11] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. 2019. How Teachers Would Help Students to Improve Their Code. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE '19)*. Association for Computing Machinery, New York, NY, USA, 119–125. https://doi.org/10.1145/3304221.3319780

[12] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. 2021. A Tutoring System to Learn Code Refactoring. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 562–568. https://doi.org/10.1145/3408877.3432526

[13] Miryung Kim, Thomas Zimmermann, and Nachiappan Nagappan. 2014. An Empirical Study of RefactoringChallenges and Benefits at Microsoft. *IEEE Transactions on Software Engineering* 40, 7 (2014), 633–649. https://doi.org/10.1109/TSE.2014.2318734

[14] Guilherme Lacerda, Fabio Petrillo, Marcelo Pimenta, and Yann Gaël Guéhéneuc. 2020. Code smells and refactoring: A tertiary systematic review of challenges and observations. *Journal of Systems and Software* 167 (2020), 110610. https://doi.org/10.1016/j.jss.2020.110610

[15] Michele Lanza and Radu Marinescu. 2007. *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems.* Springer Science & Business Media.

[16] Júlio Martins, Carla Bezerra, Anderson Uchôa, and Alessandro Garcia. 2020. Are Code Smell Co-Occurrences Harmful to Internal Quality Attributes? A Mixed-Method Study. In *Proceedings of the 34th Brazilian Symposium on Software Engineering* (Natal, Brazil) *(SBES '20)*. Association for Computing Machinery, New York, NY, USA, 52–61. https://doi.org/10.1145/3422392.3422419

[17] Júlio Martins, Carla Bezerra, Anderson Uchôa, and Alessandro Garcia. 2021. *How Do Code Smell Co-Occurrences Removal Impact Internal Quality Attributes? A Developers' Perspective.* Association for Computing Machinery, New York, NY, USA, 54–63. https://doi.org/10.1145/3474624.3474642

[18] Gerardo Matturro, Florencia Raschetti, and Carina Fontán. 2019. A Systematic Mapping Study on Soft Skills in Software Engineering. *J. Univers. Comput. Sci.* 25, 1 (2019), 16–41.

[19] Santiago A Vidal, Claudia Marcos, and J Andrés Díaz-Pace. 2016. An approach to prioritize code smells for refactoring. *Automated Software Engineering* 23, 3 (2016), 501–532.

[20] Aiko Yamashita and Leon Moonen. 2013. Do developers care about code smells? An exploratory survey. In *2013 20th Working Conference on Reverse Engineering (WCRE)*. 242–251. https://doi.org/10.1109/WCRE.2013.6671299