A Preliminary Interview Study on Developers' Perceptions of Code Smell Detection in Industry

Felipe Ribeiro[1], Eduardo Fernandes[2], and Eduardo Figueiredo[1]

[1] Federal University of Minas Gerais, Belo Horizonte, Brazil
felipelrib@ufmg.br, figueiredo@dcc.ufmg.br
[2] University of Southern Denmark, Odense, Denmark
edmf@mmmi.sdu.dk

Abstract. This paper presents a preliminary interview study aimed to understand i) how practitioners perceive code smells and ii) whether/why developers use code smell detection tools. We carefully designed an structured interview protocol composed of six major questions. We interviewed seven developers, recruited by convenience, who work for major companies worldwide on software maintenance and evolution. We followed strict guidelines for thematic synthesis to analyze the interview texts. The perception of interviewees on code smells is in line with the traditional definition, even when developers lack academic formation. All interviewees were concerned with adding code smells while they produce code, although a half of them feel that their pairs do not share these concerns. Most interviewees use detection tools, but costs with tool setup and company culture may prevent developers from using them.

Keywords: Code smell detection · Software tool · Interview study

1 Introduction

This paper introduces a preliminary interview study on code smell detection in industry. We rely on a six-question interview protocol to ask developers about i) how they perceive code smells, ii) how concerned developers are about adding code smells while they produce source code, and iii) whether/why they use (or do not use) code smell detection tools. We recruited seven developers who work for major companies. We interviewed each developer in isolation, for commodity and to avoid biases, via Telegram Messenger. We performed a qualitative analysis on the interview texts based on thematic synthesis guidelines [1].

Previous work [4, 5, 7] discusses that developers may be reluctant or too busy to use automated software development tools. This is particularly true in the case of the many tools available [2] that change the internal code structure. For instance, the use of refactoring tools has been neglected by developers [4, 7] because developers are afraid that the tools will affect the software functionalities and introduce bugs. In our study, we are specifically interested in understanding why our target developers do not use code smell detection tools.

The research questions are as follows: **RQ1:** *How do developers define code smells?* – Previous work [6, 8, 9] investigated whether developer's perception on code smells contrasts with the academic wisdom. Unfortunately, the most recent related research study [8] was conducted in 2016, so that the perceptions may not reflect the recent generation of developers. **RQ2:** *Are developers concerned about adding code smells to the source code they produce?* **RQ3:** *Do developers use tools to detect code smells on the source code they produce, consume, or maintain?* We assess the extent to which the recent generation of developers embraces code smell detection tool adoption. Our study artifacts are available online[3].

2 Study Design

The interview questions and follow-up questions are available in our website. We first approached interviewees with the main questions and, depending on their answers, we asked for clarifications or made follow-up questions as needed to deeply explore the subject and proper answer our research questions. For reference, the two first questions aim to collect background information about the interviewees. Likewise, the other four questions, identified as **C1** to **C4**, are related to the goal of our study and its research questions.

We designed core interview questions to address each of the research questions of our study. **C1** (*What do you understand as being a code smell?*) directly asks interviewees about their understanding of code smells, which means that **C1** and its follow-up questions are meant to address RQ1. Both **C2** (*Are you concerned about adding code smells to the source code you produce?*) and **C3** (*Do you believe that your teammates share the same concern?*) ask interviewees about their concerns (or concerns of their teammates) about adding code smells in the source code. Therefore, these interview questions, and their follow-up questions, aim at answering RQ2. Finally, **C4** (*Do you use tools to detect code smells on the code you produce, consume or maintain?*) is about code smell detection tools and, therefore, it is meant to provide answers to RQ3.

All interviewees are from Brazil; most of them work internationally. We selected the interviewees with four to eight years of experience in software development because our goal is to investigate the perception of a new generation of developers since the main related work is about ten years old [6, 8, 9].

3 Results on Perceptions about Code Smells (RQ1)

Figure 1 depicts our results for the thematic synthesis procedures applied to the tabulated answers on the interviewees understanding of code smells (**C1**). The root node of the tree corresponds to the major theme. All leaves from the trees corresponds to codes found during the code extraction step, being represented as nodes with a solid border (*concrete codes*), with exception to the root node.

[3] `https://github.com/felipelrib/preliminary-study-code-smells-in-industry`

Some of the codes were grouped into themes present in the second and third levels, being represented as nodes with dashed borders (*abstract codes*). All leaf nodes have a number annotated to it. The number corresponds to the frequency of the content of the node in the answers, i.e., the amount of times the code was extracted from different answers for the same question.

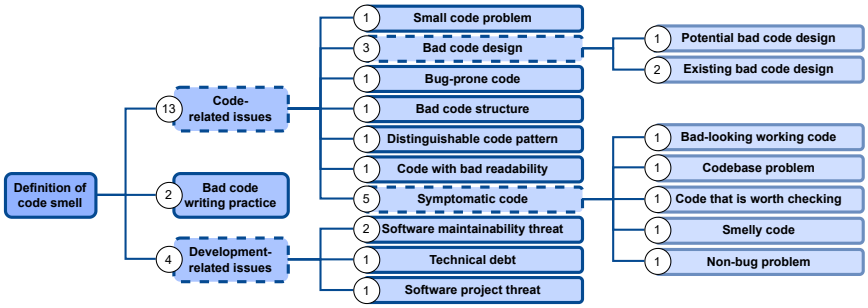

Fig. 1. Perceptions about Code Smells

The first intermediate level includes three elements: *Code-related issues*; *Bad code writing practice*; and the theme *Development-related issues*. The theme *Code-related issues* groups the nodes where the idea is related to the source code of the system. The code *Bad code writing practice* relates code smells to a bad practice from developers, associating the process to the results. The theme *Development-related issues* groups codes where the code smell is seen as an issue to the software development process.

The second intermediate level includes eight codes and two minor themes as follows: *Small code problem*; *Bad code design*; *Bug-prone code*; *Bad code structure*; *Distinguishable code pattern*; *Code with bad readability*; *Symptomatic code*; *Software maintainability threat*; *Technical debt*; and *Software project threat*. The theme *Symptomatic code* is worth noting since it is composed by some vague codes that, when grouped, mean the source code appears to have a problem in the way it was wrote – even if it works as intended. The following quote is an example of answer that specifically relates to this code: "Code smell is [...] not a bug or error – the code works, it is just possibly badly designed" (Interviewee I7). The third level contains the remaining seven codes.

Some of our results stand out as follows. Regarding the code smell definition, we found it curious that developers define code smells in terms of aspects that may not be obvious. For instance, there is this assumption in industry that smelly code can lead to bugs. If confirmed by empirical research, this can support the point of view of Interviewee I6 from which we extracted the code *Bug-prone code*: "Parts of some code [...] that potentially can bring some future bug".

Another code that is worth further discussion is the *Technical debt*, implying a need for refactoring at some point during the life cycle of a software system.

It was interesting to get this point of view from at least one of the interviewees because it somehow supports the idea that fixing code smells is relevant to some extent. In the end, we noticed that all answers are in line with the traditional definition of code smells [3], even when some interviewees lacked higher education. This could lead to the perception that the intuition behind code smells might be learned by practice, as developers with some experience can better understand the practical effects of dealing with code smells.

4 Results on Concerns about Adding Code Smells (RQ2)

The interview questions that addresses RQ2 were **C2** and **C3**. For **C2**, all seven interviewees claimed that they are concerned with adding code smells to their source code. On the other hand, **C3** answers were balanced between "Yes," "No," and a half-term "Not all", meaning that some interviewees understand that some of their colleagues share the same concern about code smells as them when coding, but others do not. The table with summarized answers is available in our website. All interviewees answered "Yes" for question **C2**, and there were three "Yes," two "No," and two "Not all" for question **C3**. The answer "Not all" was not literal in the answers from interviewees **I3** and **I6**, but was summarized in this way for a better overview in the table.

Figure 2 is a tree of themes on the reasons the interviewees mentioned not to add a code smell in their source codes. The major theme is represented by the root node, *Reasons not to add code smells*, linked with some ideas extracted in form of codes and themes from the answers of interviewees to the question **C2**.

Fig. 2. Reasons Not to Add Code Smells

The first intermediate level corresponds to the code *Avoid future development problems*, illustrated by the quote "Adding [code smells] can potentially make my life harder in the future" (Interviewee I6) and the theme *Existing development issues*. The theme *Existing development issues* groups codes that refer to the issues that are raised during the development when code smells are present.

The second intermediate level has two codes, with emphasis on *Code problem symptom*. As the interviewee explained, the presence of code smells is a symptom of bigger problems in the source code, but leaving the "problems" as something vague. This is illustrated by the following quote: "Code smells exist and are in fact a strong sign of something funky (sic) going on" (Interviewee I7).

Figure 3 is a tree of themes on if the interviewees believe their teammates share the same concerns as them about code smells. The main theme is represented with the root node *Do you believe that your teammates share the same*

concern? Why?, being a "yes or no" question. Through this question, interviewees were expected to discuss their points of view on the topic.

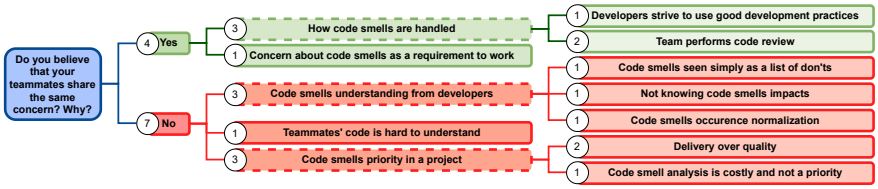

Fig. 3. Why Developers Believe Their Teammates (Do Not) Share Their Concerns

The first intermediate level is composed of the possible answers *Yes* and *No*. Their frequencies follow the previous definition, being the sum of their children frequencies, and does not represent the amount of positive or negative answers. Since these nodes were extracted as part of the answer itself, they are not counted as codes by themselves. Additionally, for both Figures 3 and 4, the children of the themes from this level is composed only by codes that ratify the theme and where the interviewees gave that initial answer, i.e., if a interviewee answered "No," but provided context that might be encoded into the "Yes" theme, this context was not considered for the taxonomy.

The second intermediate level, for the *Yes* answers, has the theme *How code smells are handled* and the code *Concern about code smells as a requirement to work*. These two elements present reasons why the interviewees believe their teammates are concerned about code smells. Regarding *No*, we have two themes and one code as follows. The theme *Code smells understanding from developers* groups some codes explaining that their teammates might not have enough knowledge about code smells and their impacts, or they simply consider their presence as something normal. The code *Teammates' code is hard to understand* means that the interviewee considers that code with low understandability is a result of a team that does not care about code smells. Finally, the theme *Code smells priority in a project* groups codes reporting that the teammates and the project management are more focused in delivering the functional requirements than in the overall project quality.

The third intermediate level consists of two codes for the *Yes* answers and five codes for the *No* answers. It is worth noting the code *Team performs code review* as a reason why some interviewees believe their teammates are concerned about code smells as they use this process, among other reasons, to analyze the presence of code smells and refactor the code as needed. This is illustrated by the following quote: "We also review each other's code, so 'code smell' normally doesn't make it to the final product" (Interviewee I4). Our perception is that a good quality code review is capable of detecting and resolving code smells.

The interviewees seemed concerned about avoiding inserting code smells in source code, even reassuring some reasons why this addition would be an issue.

Some interviewees reported that some of their colleagues appeared to have the same concerns. This claim is sustained by the perception that colleagues would apply good development practices and review code to achieve a better code quality. Others did not seem to share the same concerns, being justified by: i) their apparent lack of understanding of code smells and their impacts; ii) the source code produced by their colleagues being hard to understand; or iii) the lack of prioritization that code smells had in the scope of the project and its tasks.

The code *Delivery over quality*, for instance, refers to the interviewees describing that developers seem to be more focused on the requirements they are asked to implement rather than on the code quality. This may not mean necessarily that the teammates do not have the knowledge on code smells, but they are pressured to put this concern aside to deliver the clients' requirements faster. Meanwhile, the code *Not knowing code smells impacts* presents an idea that the lack of knowledge about code smells from teammates is latent due to this exact behavior, since the delivery is prioritized over this concern, possibly ignoring the future problems that could happen. This is illustrated by the following quote: "Maybe they don't see the direct impact that could cause, and they care more about delivering" (Interviewee I6).

5 Results on the Use of Detection Tools (RQ3)

For question **C4**, two out of the seven interviewees do not use detection tools in their projects, while five of them use. Out of those five, **I7** emphasized that they use tools in their company project, but they do not use any tools in personal projects, giving reasons why not, so we have attributed their answer as "Yes/No". The table with summarized answers is available in our website.

The tools the interviewees said they use in their projects and at which development stages are also available in the website. Interviewee **I7** informed that the tool their team use to detect code smells is developed by their company and only available there, not being publicly available. They also informed that their team uses another public library to detect code smells and detect vulnerabilities while building their projects, but they did not remember the name at the time, thus we attributed their answer as "Unknown". We see that, from all interviewees that said they use detection tools in their projects, all of them use the tools in their produced code, during the development stage.

Figure 4 is a tree of themes on if the interviewees use code smell detection tools and at which development stages, or else why they do not use tools. The main theme is represented with the root node *Do you use tools to detect code smells on the code you produce, consume or maintain? If yes: when? If not: why?*, being a "yes or no" question. Through this question, the interviewees were expected to discuss their points of view on the topic.

The first intermediate level has the possible answers *Yes* and *No*. The second intermediate level has the theme *Detection while producing code*; the code *Automated detection on maintained code*; and the theme *Detection while reviewing*

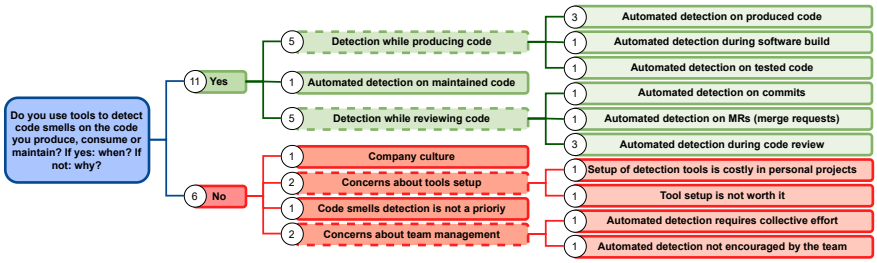

Fig. 4. Use of tools to detect code smells on produced, consumed or maintained code

code. These three, children of *Yes*, relate the interviewees answers to the stage of the code subject to the tools detection. From the *No* answer, we have the code *Company culture*; the theme *Concerns about tools setup*; the code *Code smells detection is not a priority*; and the theme *Concerns about team management*. These four nodes were extracted from the interviewees' answers to why they do not use code smell detection tools in their projects. The third intermediate level consists of six codes for the *Yes* answers and four codes for the *No* answers.

We observed that the usage of detection tools on produced code can be done in different stages of development while producing the code, as the interviewees reported using tools during the software build or while adding automated tests to the new code. It is very common for the interviewees to use tools for newly produced code that is being reviewed by their peers, be it automatically after pushing code commits or during code reviews on open merge requests (MRs) – also called pull requests (PRs) depending on the Git tool used. This is illustrated by the following quote: "I do run it again while reviewing code done by teammates to make sure this practice is being followed" (Interviewee I2).

Additionally, from the interviewees that do not use code smell detection tools, we could see scenarios where the usage of tools was not encouraged by the team management due to lack of prioritization in face of other functional and cross-functional requirements. Moreover, specifically for personal projects, some interviewees mentioned the fact that the setup of these tools would be too costly given the size of the projects. Another interviewee mentioned laziness as a factor to prevent them to setup tools in personal projects, justifying that the potential problems would not worth the cost of configuring a tool. This is shown in the following quote: "In personal projects it would be mostly a matter of laziness – there's no easier setup than no setup at all, and it is easy to justify to myself that the potential problems aren't worth the hassle" (Interviewee I7).

None of the interviewees answered that they use detection tools on consumed code, may it be a library or imported code from other projects. **I2** said that they never thought about using tools on this kind of code: "I never thought about doing it in code I consume". **I1** mentioned that they never maintain consumed code, so they would never use tools on it: "I do not run any linting or any testing

whatsoever for code I consume (third party, open source projects, etc.) as part of a personal policy of not maintaining code that does not belong to me".

6 Threats to Validity

Construct Validity: To have an interview protocol that is well-structured and sufficient to address our RQs, we decided to define the interview protocol in pairs. One of the authors proposed the first version and we had meetings to iteratively refine the questions and add new questions. We recruited the interviewees based on convenience (industry colleagues) and arbitrarily (as they are experienced, work for different development teams and are trustworthy). We had meetings to set up the way how we approach the developers during the interview conduction.

Internal Validity: We conducted the interviews via Telegram for convenience. This may prevent interviewees from engaging in the interviews, thereby resulting in information loss. In contrast to interviewing people by email, the tool allowed to closely interact with the interviewees and ask questions in a more efficient and reactive way. We tried to keep interviewees engaged with the interview by asking additional/follow-up questions, especially to make sure they understand the questions. The use of email could prevent us from clarifying things to interviewees so they answer questions properly.

Conclusion Validity: Inappropriate extraction and analysis of interview data may lead to information loss and harm the study conclusions. To guide us on performing the textual data analysis, we followed guidelines for thematic synthesis [1]. All procedures were performed in a pair. Three sessions to extract the codes (open coding). Two sessions to sort the codes (axial coding) and build the taxonomies, also based on literature guidelines [1].

Acknowledgements: CAPES, CNPq and FAPEMIG funded this work.

References

1. Cruzes, D., Dyba, T.: Recommended steps for thematic synthesis in software engineering. In: 5th ESEM. pp. 275–284 (2011)
2. Fernandes, E., Oliveira, J., Vale, G., Paiva, T., Figueiredo, E.: A review-based comparative study of bad smell detection tools. In: 20th EASE. pp. 18:1–18:12 (2016)
3. Fowler, M.: Refactoring. Addison-Wesley Professional, 2nd edn. (2018)
4. Kim, M., Zimmermann, T., Nagappan, N.: An empirical study of refactoring. IEEE Transactions on Software Engineering **40**(7), 633–649 (2014)
5. Lethbridge, T., Sim, S., Singer, J.: Studying software engineers. Empirical Software Engineering **10**, 311–341 (2005)
6. Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A.: Do they really smell bad? In: 30th ICSME. pp. 101–110 (2014)
7. Sharma, T., Suryanarayana, G., Samarthyam, G.: Challenges to and solutions for refactoring adoption. IEEE Software **32**(6), 44–51 (2015)
8. Taibi, D., Janes, A., Lenarduzzi, V.: How developers perceive smells in source code. Inf. Softw. Technol. **92**, 223–235 (2017)
9. Yamashita, A., Moonen, L.: Do developers care about code smells? In: 20th WCRE. pp. 242–251 (2013)