Journal of Software Engineering Research and Development, 2021, 9:9, doi: 10.5753/jserd.2021.1802

How are test smells treated in the wild? A tale of two empirical studies

Nildo Silva Junior [**Federal University of Bahia** | ***nildo.silva@ufba.br***]
Luana Martins [**Federal University of Bahia** | ***martins.luana@ufba.br***]
Larissa Rocha [**Federal University of Bahia / State Univ. of Feira de Santana**| ***lrsoares@uefs.br***]
Heitor Costa [**Federal University of Lavras** | ***heitor@ufla.br***]
Ivan Machado [**Federal University of Bahia** | ***ivan.machado@ufba.br***]

Abstract Developing test code may be a time-consuming process that requires much effort and cost, especially when done manually. In addition, during this process, developers and testers are likely to adopt bad design choices, which may lead to introducing the so-called test smells in the test code. As the test code with test smells size increases, these tests might become more complex, and as a consequence, much more challenging to understand and evolve them correctly. Therefore, test smells may harm the test code quality and maintenance and break the whole software testing activities. In this context, this study aims to understand whether software testing practitioners unintentionally insert test smells when they implement test code. We first carried out an expert survey to analyze the usage frequency of a set of test smells and then interviews to reach a deeper understanding of how practitioners deal with test smells. Sixty professionals participated in the survey, and fifty professionals participated in the interviews. The yielded results indicate that experienced professionals introduce test smells during their daily programming tasks, even when using their companies' standardized practices. Additionally, tools support test development and quality improvement, but most interviewees are not aware of test smells' concepts.

Keywords: *Test Smells, Survey Study, Interview Study, Mixed-Method Research.*

1 Introduction

Software projects, both commercial and open-source ones, commonly include a set of automated test suites as one crucial support to verify software quality (Garousi and Felderer, 2016). However, creating test code may require high effort and cost (Wiederseiner et al., 2010; Yusifoğlu et al., 2015; Garousi and Felderer, 2016). Automated test generation tools, such as Randoop[1], JWalk[2], and Evosuite[3], emerge as alternatives to facilitate and streamline this activity. If designed with high quality, automated testing offers benefits over manual testing, such as repeatability, predictability, and efficient test runs, requiring less effort and costs (Yusifoğlu et al., 2015; Garousi and Küçük, 2018). Therefore, tests should be concise, repeatable, robust, sufficient, necessary, clear, efficient, specific, independent, maintainable, and traceable (Meszaros et al., 2003).

However, the development of well-designed test code is neither straightforward nor a simple task. Developers are usually under time pressure and must deal with constrained budgets, which can stimulate anti-patterns in test code, leading to the occurrence of the so-called test smells. Test smells are indicators of poor implementation solutions and problems in test code design (Greiler et al., 2013). The presence of test smells in test code may lead to reduced quality and, consequently, may not reach its expected capabilities at finding bugs while remaining understandable, maintainable, and so on (Yusifoğlu et al., 2015; Garousi and Küçük, 2018). The literature reports 196 test smell types classified in the following groups (Garousi and Küçük, 2018): behavior, logic, design-related, issue in test steps, mock and stub-related, association in production code, code-related, and dependencies.

The literature presents studies aimed to identify and analyze the effect of test smells in software projects in several aspects (Greiler et al., 2013; Garousi and Felderer, 2016; Van Rompaey et al., 2006). The authors introduce test smells as non-functional quality attributes within the Software Test Code Engineering process in those studies. In addition, they discussed existing test smell types and their consequences in terms of test code maintenance (Garousi and Felderer, 2016). Some authors attempted to correlate metrics, and the presence of test smells (Greiler et al., 2013). However, few discussions about daily practices and programming styles that may contribute to insert test smells exist in the literature. Understanding the relationship between development practices and the introduction of test smell may support improving the activity of test creation.

This study extends our previous investigation (Silva Junior et al., 2020), which aimed to understand whether software testing practitioners [4] unintentionally insert test smells. We used an expert survey with sixty practitioners from Brazilian companies to analyze which and how often they adopt practices that might introduce test smells during test creation and execution. In this extension, we sought to understand (i) how much the practitioners know about test smells and (ii) how the practitioners deal with the test code quality regarding test smells. For identifying whether and to what extent the practitioners know about test smells and how they deal with them, we interviewed fifty practitioners. The results from both stud-

[1] `https://randoop.github.io/randoop/`
[2] `http://staffwww.dcs.shef.ac.uk/people/A.Simons/jwalk/`
[3] `http://www.evosuite.org/`
[4] For simplicity, we will use "practitioners" to inform "software testing practitioners"

ies are complementary. We found that most of the interviewees did not know anything about the concept of test smells. They commonly used practices that introduced test smells, but they hardly removed them from the test code.

We mapped which daily programming practices would be associated with each test smell for both test creation and execution. Then, we asked the practitioners if they used those practices without the need to name the test smells. We used the interviews to complement the survey and analyze the practitioners' unit test creation, maintenance, and quality verification activities. In addition, we investigated the practitioners' knowledge about test smells and how they treat those smells during unit test creation and maintenance.

Our study may provide insights to understand how and which practices may introduce test smells in test code. In addition, we presented the practitioners' point of view about activities related to unit test code and their beliefs about test smells' treatment. Thus, we investigated the following research questions:

- **RQ1: Do practitioners use test case design practices that might lead to the introduction of test smells?** We investigated whether bad design choices may be related to test smells.
- **RQ2: Which practices are present in practitioners' daily activities that lead to introducing test smells?** We investigated which test smells are associated with the most frequent practitioners' practices.
- **RQ3: Does the practitioners' experience interfere with the introduction of test smells?** We investigated whether, over time, practitioners improve the activity of test creation.
- **RQ4: How aware of test smells are the practitioners?** We investigated the practitioners' knowledge of test smells.
- **RQ5: What practices have practitioners employed to treat test smells?** We investigated how the practitioners deal with test smells in their daily activities.

The remainder of this article is structured as follows: Section 2 introduces the concept of test smells; Section 3 details the research method applied in this study; Section 4 presents the survey's design and results; Section 5 presents the interview's design and results; Section 6 discusses the main findings of this investigation; Section 7 presents the threats to validity; Section 8 discusses related work, and Section 9 draws concluding remarks.

2 Test Smells

Automated tests may generate more efficient results when compared to manually executed ones. Due to their repeatability and non-human interference, automated tests might lead to time and execution effort reductions (Yusifoğlu et al., 2015; Garousi and Küçük, 2018). However, developing test code is not a trivial task, and the automated tools may not ensure the system quality because they can generate one poor design (Palomba et al., 2016; Virgínio et al., 2019). In real-world practice, developers are likely to use anti-patterns during test creation and evolution, leading to errors in implementing test code (Van Deursen et al., 2001; Bavota et al., 2012). These anti-patterns may negatively impact test code maintenance (Van Rompaey et al., 2006).

Several studies investigated different types of test smells. Initially, Van Deursen et al. (2001) defined a catalog of 11 test smells and refactorings (to remove test smells from the test code). After that, other authors extended this catalog and analyzed the effects of the smells on the production and test code (Van Deursen et al., 2001; Meszaros et al., 2003; Van Rompaey et al., 2006; Bavota et al., 2012; Greiler et al., 2013; Bavota et al., 2015; Garousi and Felderer, 2016; Palomba et al., 2016; Peruma, 2018; Virgínio et al., 2019; Virgínio et al., 2020). For example, Garousi and Küçük (2018) identified more than 190 test smells in a literature review of 166 studies.

In this study, we selected 14 types of test smells frequently studied and implemented in cutting-edge test smell detection tools (Van Deursen et al., 2001; Meszaros et al., 2003; Peruma, 2018). These are described next:

- **Assertion Roulette (AR)**. A test method that contains assertions without explanation. If one of those assertions fails, it is not possible to identify which one caused the problem (Van Deursen et al., 2001);
- **Conditional Test Logic (CTL)**. A test method with conditional logic (if-else or repeat instructions). Tests with this structure do not guarantee that the same flow is verified, as they might not test a specific code piece (Meszaros et al., 2003);
- **Constructor Initialization (CI)**. A test class that presents a constructor method instead of a setUp method to initialize fields (Peruma, 2018);
- **Eager Test (ET)**. A test method checks many object methods at the same time. This test may be hard to understand and execute (Van Deursen et al., 2001);
- **Empty Test (EpT)**. A test method does not contain executable assertions (Peruma, 2018);
- **For Testers Only (FTO)**. A production class has methods only used by test methods (Van Deursen et al., 2001);
- **General Fixture (GF)**. The fields instantiated in the setUp method are not used by all test methods of a test class. It may be hard to read and understand and may slow down the test execution (Van Deursen et al., 2001);
- **Indirect Testing (IT)**. A test class has methods that perform tests in different objects because there are references to those objects at the test class (Van Deursen et al., 2001);
- **Magic Numbers (MN)**. A test method contains assertions with literal numbers as a test parameter (Meszaros et al., 2003);
- **Mystery Guest (MG)**. A test method uses an external resource, such as a file with test data. If the external file is removed, the tests may fail (Van Deursen et al., 2001);
- **Redundant Print (RP)**. A test method contains irrelevant print statements (Peruma, 2018);
- **Resource Optimism (RO)**. A test method contains optimist assumptions about the presence or absence of external resources. The test may return a positive result

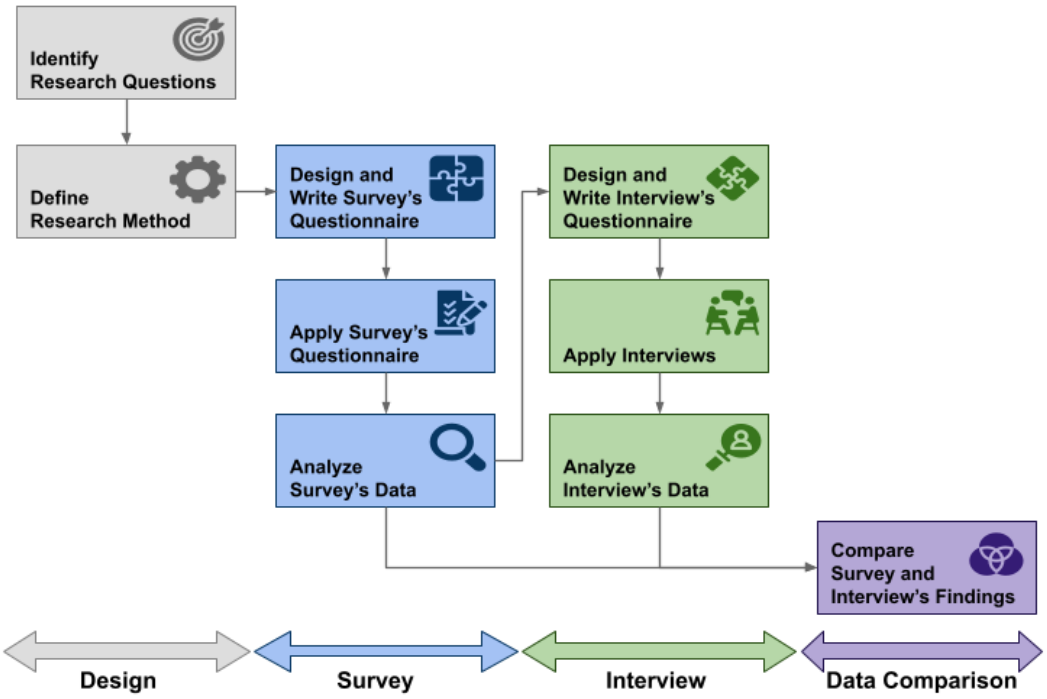

Figure 1. Research method overview.

once, but it may fail at other times (Van Deursen et al., 2001);
- **Test Code Duplication (TCD)**. A test method has undesired duplication (Van Deursen et al., 2001);
- **Test Run War (TRW)**. A test method fails when several tests run simultaneously and access the same fixtures (Van Deursen et al., 2001).

3 Research Method

We carried out two empirical studies in this investigation: a survey and an interview study (Miles et al., 2014). Figure 1 shows the methodological steps employed in this study.

Initially, we designed our study by defining the research questions and the suitable research methods to investigate them (Fig. 1 - Design). We used the *survey research method* to identify which programming practices respondents (practitioners who participate in the survey) adopt that might insert test smells in the test code (Fig. 1 - Survey). We next applied the *interview study method* to identify how the interviewees (practitioners who participate in the interview) deal with test smells during the test creation and execution (Fig. 1 - Interview). We compared results obtained from both surveys and interviews to understand the adoption of practices that might lead to introducing test smells with the practitioners' knowledge about test smells from different perspectives (Fig. 1 - Data Comparison).

For the survey, we adopted the design of observation by case-control. Case-control is a descriptive design used to investigate previous situations to support understanding a current phenomenon (Pfleeger and Kitchenham, 2001). It encompasses activities for the design, application, and analysis of a survey questionnaire. We designed the questionnaire not to require specific knowledge about test smells. We correlated each test smell to a set of programming practices, which the participants should read and analyze. Section 4 details the survey study.

To complement the findings of the survey questionnaire, we carried out a semi-structured interview (Singer et al., 2008; Gubrium et al., 2012). The interview's structure aims to capture the interviewees' perception of test smells. As we needed the interviewees to know the definition of test smells for elaborating on how they deal with them, we first introduced them to the concept of test smells. Section 5 details the interview study. The survey and interview instruments were written and applied in the Portuguese language with Brazilian practitioners. Finally, the data comparison summarizes the survey and interview results methods to answer the research questions (Creswell and Clark, 2018). Section 6 presents the results.

4 Survey Study

We applied the survey research method to investigate how the respondents commonly insert test smells in the test code when designing or implementing their software projects (Melegati and Wang, 2020). Throughout this section, we provide readers with detailed information about the research design and data analysis. All material used in the survey study, including the dataset, is publicly available at (Junior et al., 2021).

4.1 Design

We structured the questionnaire so that the respondents were not required to be aware of test smells beforehand. Thus, we

Table 1. Examples of practices related to test smells.

Test Smell	Test Creation Practices	Test Execution Practices
Mystery Guest	I often create test cases using some configuration file (or supplementary) as support.	A test case fails due to the unavailability of access to any configuration file.
Eager Test	I often create tests with a high number of parameters (number of files, database record, etc.).	I run some tests without understanding what their purpose is.
Assertion Roulette	I pack different test cases into one (i.e., put together tests that could be run separately).	Some tests fail, and it is not possible to identify the failure cause.
For Testers Only	I have already created a test to validate some feature that will not be used in the production environment.	I run some tests to validate features that will not be used in the production environment.
Conditional Test Logic	I have already created conditional or repeating tests.	I run tests with conditional or repeating structures.
Empty Test	I have already created an empty test with no executable statement.	I find empty tests, with no executable statement.

covered a larger number of potential practitioners. We correlated the concepts of test smells to commonly applied test creation and execution practices. Table 1 shows examples of those practices. For instance, the practices associated with *Conditional Test Logic (CTL)* use loops or conditions in the test code. In this case, the respondents should analyze the practices to determine whether and how often they adopt them. In CTL, the respondents should indicate how often they create tests with those structures or face them during test execution.

Questionnaire Instrument

The questionnaire comprises three blocks of questions. The first block characterizes the respondents (profile) and has thirteen questions to identify their age, gender, education degree, and software testing/programming skills.

The second block has fourteen statements and six complementary questions (four objective and two open-ended questions). The statements describe creation practices related to test smells. We structured those statements in a five-point Likert scale, where the respondents could choose one of the following answers: *always*, *frequently*, *rarely*, *never*, or *not applicable*. In this scale, *always* indicates the adoption of bad practices for test creation. For example, the *"I have already created a test to validate a feature that would not be used in the production environment"* statement corresponds to the *For Testers Only* test smell. Therefore, the answer *"Always"* means that the respondent usually uses that practice in her daily tasks. As a consequence, it is likely that she unintentionally inserts that test smells in the test code. We designed the six complementary questions to understand how the practitioners deal with the test creation activity.

The third block has fourteen statements and one additional question. Those statements describe execution practices related to test smells. Like the former block, we structured those statements on a five-point Likert scale. The respondents could choose one of the following answers: *always*, *frequently*, *rarely*, *never*, or *not applicable*, where *always* indicates that the respondent comes across with test smells. We designed the complementary question to understand which problems the respondents deal with when executing the tests. The survey was available from April 3rd, 2019, to June 3rd, 2019. Appendix A includes all the questionnaire statements and questions used in this study.

Pilot Application

We ran a pilot survey with four practitioners to identify improvement opportunities. Based on the responses, we improved the questionnaire before running the survey. It is worth mentioning that we did not include data gathered in the pilot application in the research results.

Participants

We sent invitations and one questionnaire copy (C1 - C8) to practitioners from eight Brazilian companies on a convenience sampling basis. The questionnaire's different versions served to control the number of respondents from the companies. Those companies have 4 to 66 practitioners who perform manual and automated tests (Table 2). In addition, we also sent the questionnaire through direct message (D1) and posted it on a Facebook group dedicated to discussing software testing (G1). In total, we contacted 305 practitioners, and 60 practitioners participated in the survey (`#S1` - `#S60`).

Analysis Procedure

To answer RQ1, we analyzed the objective questions (statements) on test creation (second block) and execution (third block). To answer RQ2, we grouped the practices by frequency to identify the most commonly used ones. The practices may be associated with test smells according to their characteristics, such as external file usage, conditional structure, and programming style. To answer RQ3, we compared the professional experience with the frequency of use of test smells. We also used the same answer format of RQ1 but only considered test creation (second block). During the test execution, respondents identify test smells instead of creating them.

We analyzed the three open-ended questions through coding and continuous comparison (Kitchenham et al., 2015). The objective was to understand why the respondents use practices that may insert test smells. In addition, we also intended to understand which difficulties they encounter when creating and executing tests. Two researchers performed the coding task and validated it by consensus. We also associated some practices with the test code characteristics defined by Meszaros et al. (2003).

We employed open coding on the data collected to identify additional reasons why the respondents may use bad practices in their software testing activities. The obtained codes were peer-reviewed and changed upon agreement with the

Table 2. Respondents

Source	Professionals	Answers
C1	66	14
C2	30	1
C3	10	0
C4	6	0
C5	5	0
C6	4	4
C7	4	4
C8	4	0
D1	52	35
G1	124	2
Total	**305**	**60**

paper authors. We used coding to complement our results on open-ended questions because they were optional.

4.2 Results

We received 60 answers (out of 305 potential respondents) from three Brazilian states: 40 respondents from Bahia (66.7%), 19 respondents from São Paulo (31.7%), and one respondent from Paraná (1.6%). The respondents ranged from 22 to 41 years old, and their experience with quality assurance ranged from 0 to 13 years (5.16 on average). Experience as software developers also ranged from 0 to 13 years (average 1.67). Regarding gender, 35 respondents were male (65%), 19 respondents were female (32%), and two respondents were non-binary (3%).

Most of the respondents hold a degree in Computer Science-related courses (50 respondents - 83.3%), six respondents (10%) hold a degree in other STEM (Science, Technology, Engineering, and Mathematics) courses, and four respondents (6.7%) hold a degree in other areas. Most of the respondents (54 respondents - 90%) pursued higher education degrees, as follows: 40 respondents hold a bachelor's degree (66.7%), 13 respondents hold a graduate degree (21.7%), and one respondent holds a postdoc (1.6%).

Regarding the software testing tasks they commonly perform, (i) 26 respondents reported they create and run tests at the same rate (43.3%); (ii) 13 respondents execute tests with more frequency than create (21.7%); and (iii) 8 respondents create tests with more frequency than execute (13.3%). Moreover, 12 respondents only execute test cases (20%); one respondent only creates test cases (1.7%). They perform tests over many different platforms; 35 respondents (58%) work with two or more platforms (Web - 39 respondents (65%), Android - 35 respondents (58%), Desktop - 29 respondents (48%), and Apple - 17 respondents (28%)). They also cited other platforms, such as back-end, microservices, API, mainframe, and cable TV - one respondent each (1.67%).

In terms of domain, 39 respondents claimed they test mobile applications (65%), and 36 respondents test web applications (60%). We also identified the following domains: 14 respondents work with embedded systems (23.33%), 11 respondents work with cloud computing (18.33%), seven respondents test information security (11.67%), four respondents test Internet of Things systems (6.67%). They also mentioned other domains: big data, retail, artificial intelligence, cable TV, bioinformatics, commercial information, desktop system, and payment solutions - one respondent each (1,67% each).

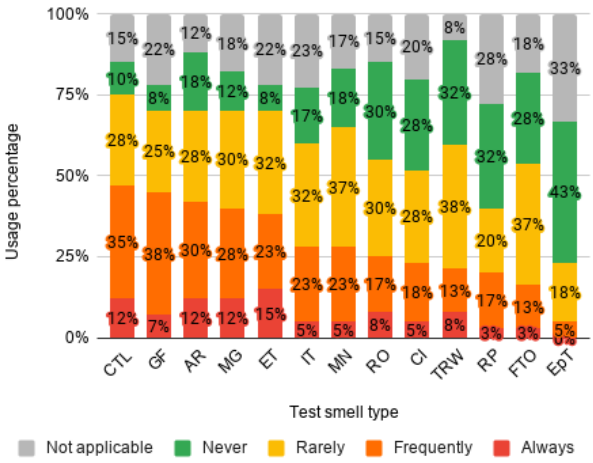

Figure 2. Test Smells frequency in test creation.

4.2.1 Test creation and execution practices

We asked whether the respondents search for test duplication and whether it was either personal or company practice. Twenty-nine respondents (48,3%) answered that it was only an individual activity. Eleven (18,3%) responded that it was only a company's practice, and three respondents (5%) claimed that it was a personal and company activity. However, seventeen respondents (28,3%) do not apply this activity. Checking tests with the same objective reduces the **Test Code Duplication (TCD)** test smell.

In addition, we established a relationship between the test creation and execution practices and the test smells occurrence using the data collected. Figures 2 and 3 show the usage frequency of test smells during the test creation and execution activities, respectively.

During test creation, the **Conditional Test Logic (CTL)** and **General Fixture (GF)** test smells were the most reported ones. The former obtained 28 (47%) of *Always* and *Frequently* responses, and the latter, 27 (45%) in both responses (Figure 2). The high rate of those responses may indicate a common everyday use of practices related to CTL and GF. We also analyzed why developers create tests with bad practices (one open-ended non-mandatory question answered by 27 respondents - 45%). The main reasons were related to the company or personally employed standards, limited time, and attempt to reach better coverage and efficiency.

We also asked whether they modified existing test sets when they came across tests containing any of the problematic test patterns illustrated in the survey. We found that seven respondents (11,7%) always perform any test code changes, twenty-three respondents (38,3%) frequently change, sixteen respondents (26,6%) rarely change, seven respondents (11,7%) never edit test code, and seven respondents (11,7%) answered as not applicable. Among the reasons to modify the test, eighteen respondents reported **ambiguities reduction** (30%), sixteen respondents claimed **execution speed improvement** (26,7%), fourteen respondents stated **adequacy**

to the company standards (23,3%), eight respondents **did not understand test objective** (13,3%) and four respondents stated **corresponding production class evolution** (6,7%).

In addition, the respondents pointed out that they used to face test structure problems. Thirty-one respondents indicated that some tests depended on third party resources (52%), 29 respondents reported that they were hard to understand (48%), 24 respondents claimed to contain unnecessary information (40%), 24 respondents said ambiguous information (40%), 20 respondents reported to depend on external files (33%), six respondents pointed to use an external configuration file (10%). One respondent presented resources limitation (2%).

Regarding difficulties in creating test cases (one open-ended non-mandatory question answered by 23 respondents (38%)), requirement issues were the most frequent ones, reported by twelve respondents (52%). Other problems were related to the difficulties in the test code reuse, lack of knowledge, production code issues, code coverage, test environment problems, and time and resource limitation.

The test execution questions also presented a sequence of statements about ordinary situations the developers usually face, in which respondents should answer according to the frequency. The **CTL** (52%) and **GF** (47%) test smells were also the most cited during test execution (Figure 3). Those test smells obtained 31 and 28 answers of *Always* and *Frequently* frequencies, respectively.

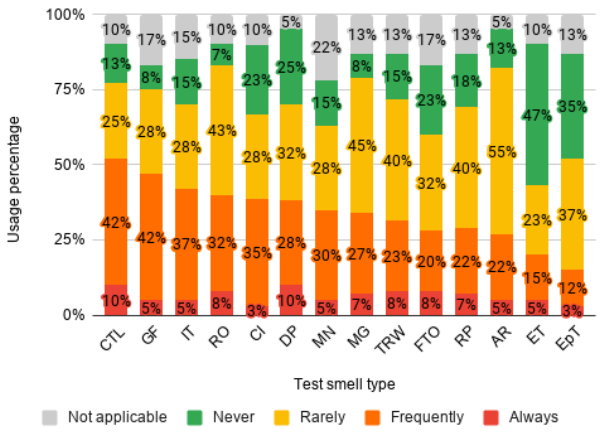

Figure 3. Test Smells frequency in test execution.

Regarding difficulties in running test cases (one open-ended non-mandatory question answered by 29 respondents - 48%), ten respondents reported test environment as a problem related to test execution (34%), such as test environment unavailability, demand for third-party features, and low-performance environments. The second most common problem is understanding the test purpose (28%), where eight respondents reported that tests were poorly written and without a standard, allowing multiple interpretations. The lack of test maintenance was the third problem (24%), which involves outdated and incomplete tests due to the system code evolution (7 respondents).

Table 3. Answers grouped by experience range

Experience (in years)	Number of respondents	Total
0 - 2	11	143
> 2 - 4	12	156
> 4 - 6	15	195
> 6 - 8	5	65
> 8 - 10	9	117
> 10 - 12	4	52
> 12 - 14	4	52

4.2.2 Professional Experience

Although most respondents from the survey reported they create and execute tests simultaneously, our investigation presented a different scenario as the tester gets more experienced. Figure 4 shows the daily activities according to the professional experience, with the following highlights: 10 respondents (16.7%) with experience ranging from 4 to 6, and 5 respondents (8.3%) with 8 to 10 years of experience create and execute tests at the same proportion. Eight respondents (13.4%) with less than two years of experience, six respondents (10%) ranging from 2 to 4 years of experience, and four respondents (6.7%) ranging from 6 to 8 years of experience only run tests or run tests with more frequency than create. Three respondents (5%) with more than 12 years of experience mostly create rather than run tests. Therefore, less experienced respondents run more than creating tests, and respondents with more experience create more than run tests.

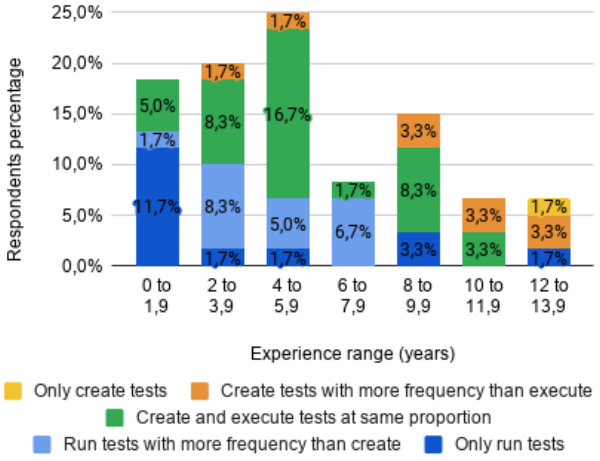

Figure 4. Testing tasks according to professional experience.

We also analyzed whether the use of good practices to create tests increases as respondents become more experienced. We provided the respondents with thirteen statements, with illustrative scenarios of problems with test cases. Each scenario relates to a given test smell. The respondents had to answer how often they experienced each scenario. Table 3 shows the number of respondents grouped by experience time (in years) and the number of valid responses.

Figure 5 presents the frequency of test smells grouped by professional experience. When we analyzed the first experience range (0-2), 71 answers (50%) from the respondents could not identify the adoption of practices related to test smells (*Not applicable*). 9 (6%) answers pointed that respon-

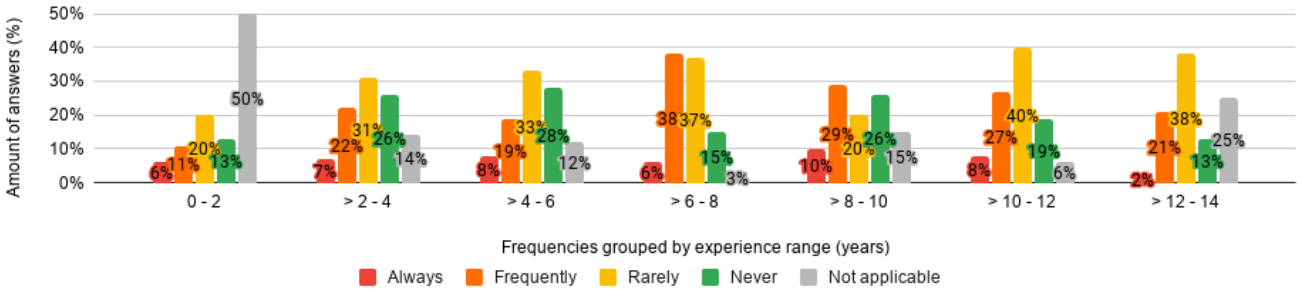

Figure 5. Test Smells frequency in test creation according to professional experience.

Table 4. Interview questions

#	Question
1	How did you start working with software test?
2	What were your learning sources about test code?
3	Which programming languages do you create tests for?
4	Which programming languages do you use in your current software project?
5	How is your test creation process?
6	Is there any flowchart or template document that standardizes this process?
7	Which support tools are used for test creation and execution?
8	How do you verify the quality of unit tests?
9	Moving to the test code maintenance process, tell me how is this process inside the company?
10	What do you know about the test smell?
11	How did you learn about that?
12	Do you have any doubt about test smell?
13	How are the test smells handled in the unit testing creation process?
14	How are the test smells handled in the unit testing maintenance process?
15	How would it be possible to avoid the introduction of test smells during test creation?
16	Do you have any question, additional information or suggestion to improve this interview?

dents *Always* adopted some practice related to test smells, 16 (11%) answers related to *Frequently*, 29 (20%) answers to *Rarely*, and 18 (13%) answers to *Never* adopted practices related to test smells. When we extended that analysis through the next experience ranges, we could not observe any increase in responses *Never* and *Rarely* with the professional experience, indicating that the experience might not influence the adoption of practices that lead to the introduction of test smells.

5 Interview Study

After carrying out the survey study, we interviewed Software Engineers to gather further evidence on how the practitioners deal with test smells, develop unit test code, and deal with test smells in test creation and maintenance. The interview dataset, including the interview transcriptions, interviewees profile, and coding summary, is publicly available at (Junior et al., 2021).

5.1 Design

We employed a semi-structured interview approach, guided by a set of sixteen questions, as Table 4 shows.

Interview Organization

We organized the interview into three blocks:

- *Warm-up block (#1-3).* Questions about the professional background, such as the learning resources on software test code the interviewees commonly use, as well as the programming language they often use to implement test code, if any;
- *Technical block (#4-9).* Questions about how they create, maintain, and assess the quality of developed unit tests;
- *Test Smell block (#10-15).* Questions about the interviewees' awareness of test smell and how they handle these in test case creation and maintenance.

The interviewees could also ask for more information or give additional information and suggestions to increase the interview quality (question #16). Unlike the survey, in the interview, we employed the actual *test smell* term in the questions related to such the concept instead of considering a transitive approach through statements containing practices embedded with test smells. When the participants were not aware of the term or asked for more information on test smells, we presented the concept and two test smells samples, e.g., CTL and EpT (Virgínio et al., 2020). Those test smells were related to the most and the least frequently programming practice used on survey results, respectively. There were no questions about challenges or problems involved in creating and maintaining test code. The interviewees answered the questions in Table 4 according to their experiences, concepts, and shared information during the meeting.

The interviewer and interviewees did not access any test code from interviewees to analyze the presence of test smells.

At the beginning of the interview, the practitioners answered a professional profile' form with academic background and professional experience. They also provided an email to solve eventual doubts or collect more data during data analysis. We interviewed on June 3rd and June 30th. Due to the pandemic period, online meeting tools, such as Skype and Google Meet, were used upon the participants' request. We recorded the interviews with either the Skype conversation recording tool or the Google Meet screen capture feature. Additionally, we used an external voice recorder for every interview.

Participants

Initially, we contacted practitioners from the survey who agreed to keep contributing to research. Unlike the survey, we opted only for test code developers whose focus was creating and maintaining unit testing, including the treatment of test smells. Some interviewees participated in the survey study because we applied the snowballing technique (Kitchenham et al., 2015). Next, we used LinkedIn to invite other potential participants, using the "unit testing" expression in the profile ability search Linkedin provides users. A total of 50 practitioners accepted the invitation (`#I1` - `#I50`).

Pilot Study

We performed two pilot interviews with practitioners to measure the interview length and analyze whether it would be necessary to modify any part of the predefined instrument. As a result, there was no need to perform any changes in the instrument. The average interview length was around 30 minutes.

Analysis Procedure

The first author was the one responsible for transcribing the interviews. From them, we performed open coding (Corbin and Strauss, 2014) to answer the research questions. The remaining co-authors analyzed the transcriptions to understand how the practitioners develop tests and deal with test smells. First, we analyzed and validated the coding until we reach a consensus. In the following, two authors individually reviewed the proposed coding. In the end, one expert researcher reviewed the final coding.

5.2 Results

The interviewees could answer open-ended questions in different ways, according to their reality. Therefore, when presenting the results, some responses got more than 100% during the quantitative analysis.

The respondents' age ranged from 20 to 48 years old, most of them ranging from 25 to 34 years old (60%). Regarding their education, six respondents have completed high school (12%), 31 respondents completed an undergraduate school (62%), and 13 respondents hold a graduate degree (26%). Additionally, 48 respondents either have a degree or were studying any Computer Science-related course (96%), one respondent holds a degree in Applied Business (2%), and one respondent holds a degree in Psychology (2%).

The respondents worked in companies of different sizes, as follows: (i) 10 respondents worked in small companies (less than 50 employees - 20%); (ii) 5 respondents worked in medium-sized companies (number of employees in the range from 50 to 99 employees - 10%); and (iii) 35 respondents worked in large-sized companies (more than 99 employees - 70%). Additionally, the interviewees were responsible for different tasks within companies related to their current roles (Table 5). They created unit tests for mobile, desktop, and web platforms using different programming languages (Table 6). Their experience in software development tasks varied from 1 to 20 years, of which more than 50% were in the 1-6 years of experience range. Two out of them were not working with unit test creation when we interviewed them. In such cases, they should consider their previous experience.

Table 5. Respondents' roles

Role	Respondents	%
Developer	22	44%
Software Engineer	7	14%
Systems Analyst	7	14%
Software Architect	5	10%
Team Leader	3	6%
Automation Engineer	2	4%
Consultant	2	4%
Project Manager	2	4%
Quality Specialist	2	4%
Quality Engineer	1	2%
Test Developer	1	2%
Test Analyst	1	2%

Table 6. Programming languages

Language	Respondents	%
Java	25	30%
JavaScript	14	17%
C#	11	13%
TypeScript	8	10%
Python	7	9%
Kotlin	5	6%
PHP	4	5%
Swift	3	4%
Ruby	2	2%
C	1	1%
C++	1	1%
Elixir	1	1%
Go	1	1%

We compared and analyzed the information for the open coding analysis and grouped them into codes using sentences, paragraphs, or the entire document. For example, when we asked them about their unit test creation process, the interviewee #I47 answered: *"When I worked only with Java [...] if I know the context well if I have deep knowledge of the context that I will develop, I like to do a little TDD [...], but unfortunately this is not something that can be 100% reality in the business, because you have N situations, N circumstances. So I cannot do TDD; at least I develop the specific feature, [...] the features, methods, etc., and then I will test it, for example, for each method that I know has a logic within that method, I do the test cases for N possibilities"*. From this answer, we identified the following codes: `CodeA` - TDD; `CodeB` - TLD;

CodeC - depends on personal skill. We found 159 codes.

We did not consider the warm-up block answers (#1 - 3) as we used them to stimulate the interviewees to provide as much information as possible. We used the technical block answers (#4-9) to analyze how the interviewees created, maintained, and verified the test quality to complement and compare the survey's supplementary questions. We used the answers for question #10 to analyze which information the interviewees presented about test smells. Therefore, we could answer RQ4. Questions #11 and #12 complemented question #10. We used answers for questions #13 and #14 to analyze the strategies for dealing with test smells and answer the RQ5. Then, we analyzed the answers given to question #16 to understand how the interviewees believe it was possible to avoid introducing test smells. Those questions let us understand better how they create, maintain, and verify unit test codes and how they deal with and possibly avoid test smells.

5.2.1 Unit test code creation and maintenance

We found that the developers usually create unit test code using *Test-Driven Development (TDD)* (48%), *Test Last Development (TLD)* (42%), or *Behavior Driven Development (BDD)* (16%). Those strategies' usage was motivated according to the project task or developer's knowledge about the project's programming language or architecture. For example, the interviewee #I16 stated that he used TDD when he dominated the programming language; otherwise, the functional software code was created first and then tested (TLD). The interviewee #I25 claimed that she created unit tests according to the stories from the BDD scenario. When there was no scenario, she used TDD. The method adoption could also depend on if the software was new or legacy. The interviewee #I32 pointed that TDD was used on new projects when possible, and he used a BDD variation before the software code creation.

During the test code creation description, four interviewees (8%) mentioned using *Mocks* to simulate components, and two interviewees (4%) used to adopt *clean code practices*. For instance, the interviewee #I22 claimed he creates easy-to-read and understand, fast, and independent test codes. The interviewee #I36 uses code patterns and creates less verbose tests. Additionally, the focus of four interviewees (8%) is on *test coverage*. The interviewee #I12 claimed that he identifies "interesting features" to test. According to the interviewee #I43, the test code should cover 80% of the software code. Moreover, the interviewee #I10 mentioned the *SOLID principles*, and the interviewee #I15 adopts the *Model-View-ViewModel (MVVM)* project pattern (#I15) as practices during the test creation.

When we asked whether there was any document that standardized unit test creation, nine interviewees (18%) indicated the use of *templates* or some other *documentation*. The interviewees #I5 and #I9 mentioned a test template in their projects that the team members could adopt. The interviewee #I29 claimed his team followed the *Microsoft's official documentation*, but there was not any internal document. The interviewee #I39 mentioned using a *Domain Specific Language (DSL)* to share project information, as follows: "On project day 0, we create and standardize an official DSL for the code. You have prerogatives, you have the test, and you have the result". In addition, some interviewees answered that there was no documented standard, but they adopted the *Given-When-Then (GWT)* pattern and the *Arrange-Act-Assert (AAA)* programming practices.

Furthermore, the interviewees mentioned 90 different *tools* to create and run tests. Those tools are related to (i) code development (*JUnit* - 42%, *Jest* - 14%, and *Visual Studio* - 20%); (ii) metrics analysis (*Sonar tools* - 18%), and (iii) Continuous integration (*Jenkins* - 10%, *Azure* - 2%, and *Circle CI* - 2%).

After creating unit test code, the *test quality assessment* was performed through *code review* (78%) by one or more developers inside the project team. This activity usually was supported by tools, such as Pull Panda. For example, the interviewee #I2 claimed: *"Pull Panda*[5] *is a tool used to randomly assign one or more developers to perform the code review. [...]"*. Furthermore, two other interviewees (interviewee #I4) and (interviewee #I16) reported that they performed *peer review* (4%), and four interviewees claimed they commonly verify test code quality through *pair programming* (8%). Other practices identified were: *test coverage* (30%), *metric analysis tool* (24%) (e.g., SonarQube tool), reviewing by *continuous integration tool* (16%), *test execution* (10%), application of programming practices (10%) (*reuse, clean code,* and *libraries*), running *mutant test tool* (6%), test validation by *external Quality Assurance team* (2%), and *static validation* (2%). Three interviewees reported that there were *"no test quality assurance"* activities because there were not enough tests to perform this activity or because the company does not support it.

The interviewees adopted various *test maintenance types* distributed by *corrective* (62%), *adaptive* (36%), *preventive* (4%), and *perfective* (4%) maintenance. Four interviewees claimed there was *no test code maintenance* due to: (i) there was no defined maintenance process (interviewee #I22); (ii) the participation in one new project and no maintenance task was required (interviewee #I24); absence of maintenance activity because of shortage of time (interviewees #I24 and #I36); and (iv) project environment (interviewee #I45).

5.2.2 Test smells treatment

We asked the interviewees about their knowledge of test smells to understand whether they comprehended the study subject. Figure 6 summarizes the results. Seven interviewees (14%) demonstrated *some knowledge of test smells*. For example, the interviewee #I2 answered: *"I know a few things. I consider these as bad practices, bad choices that you make in your test code that difficult its maintenance and evolution."*. Twenty-three interviewees (46%) *related test smells with code smells* but claimed they have never heard of the test smells. The interviewee #I16 mentioned: *"Test smell, I do not know the concept. The code smell is a problem that the static test analysis tool found in the program. Would test smell be that same analysis on top of the test code?"*. Finally, twenty interviewees (40%) *did not know test smells* and did not relate to any smells type.

[5] https://pullpanda.com/

We presented the definition and examples of two test smells (CTL and EpT) for the interviewees who did not know about test smells or asked for more information. Table 7 shows how they prevent test smells during test code creation and how they treat test smells during the test code creation and maintenance. For example, during the test code creation, the *Code review* practice was the most recommended (38%), followed by *Tool usage* (26%) and *Programming practices* (24%). When developing the test code, the developer should follow the programming practices to prevent test smells. Tools and code reviews help to check the test smells insertion in an early stage of development. Two interviewees believed there were not test smells in their repository. For example, the interviewee #I39 said: *"I think we do not have this problem (test smells) in the recent project because of its difficulty level, we follow a coding standard. We educate people on how we code it [...]"*. The interviewee #I11 also said: *"As I am the only one working on the project, I coded, understood, and never had this vision of test smells. I do not think I have any problem with that."*.

Regarding maintenance, we asked how the interviewees treated test smells during the test code maintenance. The answers were similar to the previous question (Table 7). For the test code maintenance, the *Code review* was also the most recommended practice (28%), followed by *Refactoring* (20%) and *Tool usage* (18%). As the test code was already developed and might have test smells, they suggested using tools to help detect test smells and refactoring techniques to remove them from the test code. The code review practice can double-check the test code to treat the test smells during the maintenance.

We also asked the interviewees how to prevent test smells during test code creation (Table 7). For the test smells prevention, the *Tool usage* was the most recommended practice (44%), followed by *Developers' skills* (28%) and *Code review* (20%). The developers' skills are related to developing tests' know-how by following good practices, guidelines, and coding patterns. It should help the developers identify and prevent flaws in designing and implementing a test code. The tool usage can support the developers when developing a test code by identifying possible test smells. The code review is a manual analysis of the test code to double-check the test code for test smells prevention.

At the end of the interviews, they could either provide or ask for further information about test smells and test code quality assurance. Therefore, the interviewee #I29 claimed: *"For me, it is a quality guarantee in terms of dependence exemption, in terms of development, cohesion, coupling, and fundamental architecture. From the moment you have unit testing or even TDD, it helps you improve the code and architecture."*. The interviewee #I35 demonstrated interest in our study: *"I would like to know more about the study, we can talk about it later if you want, [...] I thought the term 'test smell' is complicated, at least it does not seem to be a common industry expression."*.

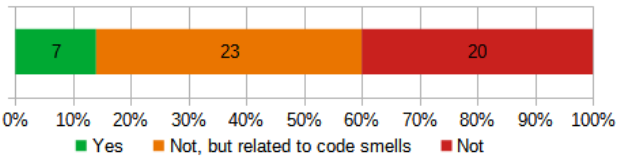

Figure 6. Prior knowledge about test smell.

Table 7. Practices to prevent test smells or to treat them during the test code creation and maintenance

#	Practice	Creation	Maintenance	Prevention
1	Code analysis	–	–	2 (4%)
2	Code removal	–	1 (2%)	–
3	Code reuse	–	–	1 (2%)
4	Code review	19 (38%)	14 (28%)	10 (20%)
5	Coding patterns	4 (8%)	5 (10%)	8 (16%)
6	Company support	–	–	1 (2%)
7	Culture's development	–	–	3 (6%)
8	Developer skills	2 (4%)	2 (4%)	14 (28%)
9	Documentation	–	–	1 (2%)
10	Guidelines	–	–	3 (6%)
11	Individual analysis	2 (4%)	6 (12%)	–
12	Mutant testing	1 (2%)	1 (2%)	–
13	No treatment	13 (26%)	13 (26%)	–
14	Pair programming	2 (4%)	1 (2%)	4 (8%)
15	Peer review	1 (2%)	1 (2%)	1 (2%)
16	Professional experience	–	–	6 (12%)
17	Programming practices	12 (24%)	8 (16%)	11 (22%)
18	Refactoring	5 (10%)	10 (20%)	–
19	TDD	–	–	3 (6%)
20	Technical debt	1 (2%)	5 (10%)	–
21	Technical meeting	1 (2%)	–	–
22	Tool usage	13 (26%)	9 (18%)	21 (44%)
23	Traceability	1 (2%)	1 (2%)	1 (2%)
24	Training	–	–	8 (16%)
25	Take breaks	–	–	1 (2%)
26	Software code improvement	–	1 (2%)	
27	Test Smell Catalog	–	–	1 (2%)

6 Discussion

This section discusses the results obtained after conducting the survey and interview to answer the research questions. RQ1, RQ2, and RQ3 are related to the survey, and RQ4 and RQ5 are related to the interview.

6.1 RQ1: Do practitioners use test case design practices that might lead to the introduction of test smells?

We observed that at least one respondent pointed to 1 out of 14 practices related to test smells from the results. We analyzed those practices when creating and maintaining tests to identify which types of test smells the participants frequently insert in the test code.

Regarding test creation, we observed that every test smell presented at least three out of four possible answers (*Always, Frequently, Rarely, and Never*). We classified the data into two groups: the Commonly-used practices group (CPG) and

the Unused practices group (UPG). CPG contains test smells that mostly present *Always* and *Frequently* as answers, and UPG that mostly present *Rarely* and *Never* as answers. We considered a test smell belonging to one group when the difference between the *Always* and *Frequently* rates and the *Rarely* and *Never* rates is greater than 10%. For example, the `Empty Test`, `For testers only`, `Test Run War`, `Constructor Initialization`, `Resource Optimism`, `Redundant Print`, `Magic Number`, `Indirect Test` test smells belong to UPG, which means practitioners rarely insert those smells on the testing activities.

On the other hand, the respondents frequently adopt practices related to the `General Fixture` test smell, the only member of CPG, indicating that they usually create tests with that smell. Still, four test smells presented a similar pertinence frequency to both groups (less than 10% of difference). For them, there was not a pattern among respondents. For instance, the `Eager Test` test smell obtained 38% to CPG and 40% to UPG.

In the test execution, UPG contains the `Empty Test`, `Eager Test`, `Assertion Roulette`, `Redundant Print`, `Duplicated Test`, `Test Run War`, `For testers Only`, `Mystery Guest`, `Constructor Initialization`, and `Resource Optimism` test smells, which means that the respondents rarely face those smells during the test execution. Otherwise, the respondents frequently find practices related to two test smells, `General Fixture` and `Conditional Test Logic`, which compose the CPG group. In addition, we did not perceive a significant difference among respondents for two other test smells, `Indirect Test` and `Magic Number`, which presented similar pertinence frequency to both groups.

We also investigated the reasons that lead the respondents to adopt the practices presented in the survey. Thus, we analyzed the open-ended questions and identified 16 different tags. The most common ones were *company standard*, *personal standard*, *project politics*, *professional experience*, *saving time*, and *improving coverage*. For example, the respondent `#S26` of the survey reported applying company standards when creating tests that may insert smells and commonly use bad practices *"to match company development standards."* In another situation, respondent `#S54` reported using personal standards when said: *"I group tests by modules to execute them sequentially without compromising effectiveness."* This behavior suggests that participants may have misunderstood the test smells definition. When grouping tests, it is possible to insert the `Assertion Roulette` test smell and compromise test independence. A similar situation occurred with the respondents `#S14`, `#S16`, `#S27`, `#S50`, and `#S59`.

In general, our study identified that all test smells appeared in testing activities. They all were cited by respondents, even if rarely.

> Practitioners adopt practices for test case design, which introduce test smells. Usually, those practices come from improper personal and company standards.

6.2 RQ2: Which practices are present in practitioners' daily activities that lead to introducing test smells?

Although there are specific tools to support test automation (Fraser and Arcuri, 2011; Smeets and Simons, 2011), 62% of respondents perform more manual than automated tests. Besides, 55% have no experience with software development (less than two years of experience), the lack of knowledge does not influence the adoption of bad practices in the test code.

According to the practices explored in the survey, we identified that the respondents usually come across: *(i) the use of generic configuration data*, which produces the `General Fixture` test smell (most frequent on the activities of test creation and execution - CPG); and *(ii) the use of conditional or repetition structure*, directly associated with the `Conditional Test logic` test smell (second most detected on the activity of test execution - CPG).

The respondents indicated they usually face several problems with tests, such as poorly written tests and outdated and incomplete test procedures. According to them, when the tests are associated with generic configuration data, test cases are hard to understand and may cause incorrect results. Moreover, the test coverage on the production code is unclear due to the conditional logic presence on the tests. Understanding which practices are most prevalent in the professionals' activities supports improving test quality. Other identified problems are related to incompleteness, outdatedness, or lack of documentation. These may hinder traceability, evolution, and maintenance of the testing tasks.

> The practices most present in the practitioner's daily life that lead to test smells insertion were conditional structure or repetition and generic configuration data.

6.3 RQ3: Does the practitioners' experience interfere with the introduction of test smells?

In the survey study, we analyzed the respondents' experience and its influence in adopting practices that might lead to insert test smells in their projects. As a result, we did not identify any clear cause-effect correlation. For example, the *Always* option indicates they always use harmful practices. When we analyzed the answers' frequencies for this option, the usage rate did not reduce over time. Instead of that, we may observe from Figure 5 that respondents with 8 to 10 years of experience achieved a higher usage rate of this frequency. We also identified that behavior when we analyzed the other usage frequencies. However, we could not infer that inexperienced practitioners introduce more test smells than experienced ones regarding the activity of test creation.

On the one hand, when testers are inexperienced programmers, they may write lower-quality tests. On the other hand, they can carry programming biases that may contain bad practices when they are more experienced. Thus, the absence of a tendency indicates a non-behavioral change between less and more experienced practitioners.

Experienced practitioners may not produce fewer test smells than inexperienced ones.

6.4 RQ4: How aware of test smells are the practitioners?

The survey results indicate that the lack of information on test smells is one reason that leads practitioners to adopt programming practices that may introduce test smells. Although the test smell concept had appeared in 2001 (Van Deursen et al., 2001), when we asked in the interview what they know about them, 14% of the interviewees demonstrated having some knowledge. For example, two interviewees mentioned: *"I know a little bit about test smells. If I am not mistaken, there are smells like Test Assertion and Duplicated [...]"* (`#I5`) and *"Test smell? From smells? I know the basics"* (`#I19`). We believe that the industry should explore this topic more through the initiatives proposed in academia (Santana et al., 2020).

Some interviewees (46%) associated the test smells' term with the code smells and related test smells detection with tool's usage or personal practices. For example, interviewee `#I04` mentioned: *"Although I had never heard the term, it makes sense, because I saw everything as a code smell, but there are some strategies, some guidelines that I follow for unit tests."*. This behavior may generate disagreement on the tool functioning, such as the interviewee `#I10` said: *"One of the outputs of those software that I mentioned, SonarQube and Code Climate, are these test smells. They can find some of them, [...] because we can not publish a project with these types of test structures, tests with commented content, such as empty test, the test with a complexity greater than 1"*. Conversely, In the SonarQube documentation, there is no information about test smells analysis. Thus, we considered that those analyses are related to code smells in test code, which is different from test smells detection.

Test practitioners do not know what test smell is. They can associate the test smell concept with code smells, but they have no information about test smell types and refactoring.

6.5 RQ5: What practices have practitioners employed to treat test smells?

Commonly, the interviewees did not know what test smells are. After explaining the concepts to them in the interview, they could understand and explain how they deal with test smells in their daily activities. They reported adopting a set of project's activities (e.g., code review, pair programming, and technical debt) and programming practices during the test creation and maintenance processes (e.g., the clean code approach and Given, When, Then (GWT), and Arrange, Act, Assert (AAA) patterns) to either prevent or treat test smells.

The interviewees tended to develop unit tests according to their skills. The professional abilities also determine the result of code review. The interviewees who did not learn about test smells or programming practices can approve a submitted package with these issues. The code review was the most reported activity to treat test smell in test creation (38%) and the most common activity performed by the interviewees (78%) during test quality verification. In this activity, one or more practitioners analyze the submitted code. In this context, the reviewer's knowledge determines whether the code is good enough to merge it on the repository.

Each team adopts different strategies to perform code reviews based on the number of reviewers, number of approvals, and professional experience. Although some interviewees reported that only the experienced members review software and test code, the review may not avoid test smells in the project repository, mainly because both experienced and inexperienced practitioners adopt practices that introduce test smells.

When we asked about the test smell treatment during test maintenance, some interviewees reported creating a technical debt to refactor test smell in another moment (interviewees `#I08`, `#I09`, `#I22`, `#I25`, and `#I50`). This behavior may indicate that the test smell correction is not a priority. The technical debt creation may also be the reason why test smells remain in the repository. For example, interviewee `#I09` said: *"There is nearly no treatment for test smells. [...] when removing a feature from the software or its business rule is changed, the test code is commented and left there. [...] Hardly the developers handle with commented test codes. [...]"*.

The interviewees hardly addressed the technical debt and failing tests because they needed to prioritize other tasks as software code development. With less time for testing, test smells would be introduced in the test code during test creation and maintenance and keep in the repository through postponing maintenance activities.

We did not know whether practitioners have learned about test smells. Thus, we adopted the concepts of test smells in literature. The validation of those concepts was out of scope. Although we did not ask specifically whether the interviewees considered test smell as a problem or agreed with the given test smells examples as a smell, during their answers about test smell treatment, part of them told about how they treat at least one of the given examples. For example, the interviewee #I07 said: *"Despite not having worked exactly with this type of concept, Sonar itself warned us about these two problems, both when the logic was very complex, with a lot of "if," it warned us to break it in different methods, things like type. Moreover, I remember that it identified comments, commented code, and sends a warning"*.

Regarding the Conditional Test Logic smell example, #I37 said: *"This specific code enters into a specific clean code case. This test may be doing more than it should"*. According to these comments, the interviewees consider test smells, including the given examples, as structures to fix.

Practitioners adopt a set of project activities and programming practices to treat test smells. As they do not know well the test smells concepts, it is impossible to guarantee that those strategies treat test smells appropriately.

7 Threats to validity

Internal validity. Although there are more than 100 test smells, this study only considered 14 test smells. However, we selected the most frequent test smells discussed in the literature. In addition, the test smells were presented in the survey as practices. To mitigate ambiguities and text comprehension, we applied a pilot with four testers from different companies. We used professional social networking to reach as many respondents as possible from Brazilian companies demographically distributed for the survey and interview execution.

External validity. Our survey and interview respondents may not adequately represent the practices adopted by the practitioners in the wider software engineering industry. Although our results may not generalize, they provide a practice adopted initial view by the testers. There is an agreement among the practitioners' responses, indicating that additional data might not reveal new insights.

Construct validity. The survey did not inform that the questions referred to test smells to investigate whether the practitioners non-intentionally insert test smells. We prevented the respondents' partiality when identifying the practices adopted. Complementary, to investigate how the practitioners deal with test smells. We presented the concept to the interviewees who did not know this subject. After learning test smells, the respondents were interested in finding solutions for this "problem" (test smells). We collected open-ended questions answers and performed one peer-reviewed coding process to avoid biases. The survey and interview instruments were written in Portuguese and translated to English by one author but reviewed by others.

Conclusion validity. The data analysis was an exhaustive process, which depends on the researchers' interpretation of the open-ended questions answers. To prevent biases, we performed the data analysis in three steps: i) two researchers analyzed the data on pair to discuss the identification of the code, ii) two researchers analyzed the data individually, checking if new codes could emerge, and iii) all researchers discussed and compiled the results from steps i and ii. Additionally, to increase transparency, the crude survey and interview data are available online for other researchers to validate and replicate our study.

8 Related work

Bavota et al. (2015) presented a case study to investigate the test smells impact on maintenance activities. In that study, developers and students analyzed testing code to compare whether their experience would make a difference in test smell identification. As a result, they found that the intensity of the test smells' impact is different for different levels of experience; the number of impacting test smells is higher for students than industry professionals. Additionally, they found that test smells have a significantly negative impact on maintenance activities. Conversely, our survey found that the practitioners' experience does not interfere in the test smell introduction during test creation and execution activities. Moreover, the interview revealed that the practitioners are not aware of test smells, reinforcing that the experience is not influencing the test smells insertion in the test code.

Tufano et al. (2016) proposed an interview study with 19 participants to investigate developers' perception of test smells. They performed an empirical investigation to analyze where test smells occur at the source code. The results showed that developers generally do not recognize test smells, and there are test smells since the first code commit in the repository. Similarly, our interview indicated a lack of awareness of the developers about the underlying concept of test smells. Additionally, we did not find any study investigating how professional practices affect the test smells introduction, and therefore we investigated it through a survey.

Spadini et al. (2020) surveyed developers to evaluate the severity thresholds for detecting test smells and investigate test smells' perceived impact on test suite maintainability. The developers had to classify whether a test smell instance is valid and rate the test smell instance regarding its importance to its maintainability. The evaluation of test smells instances requires knowledge about the topic. Therefore, our survey presented practices that might lead to test smells insertion, and our interview provided information about test smells to level the respondents about the topic.

In our previous work (Silva Junior et al., 2020), we conducted an expert survey to understand whether practitioners unintentionally insert test smells. We surveyed sixty Brazilian practitioners regarding fourteen bad practices that might lead to the test smell insertion during the test code creation and execution. The results indicated that the practitioners' experience might not influence the test smells insertion. Usually, practices that lead to test smells insertion came from improper personal and company standards. This current study complements the previous one by investigating the practitioners' knowledge about test smells and how they deal with the test code quality regarding the presence of test smells. We conducted interviews with fifty Brazilian practitioners to ask them about the test code creation and maintenance processes. As a result, the interviewees indicated a set of practices that might be useful to treat test smells. However, as they do not know about test smells concepts, those practices need further investigation for test smell treatment.

9 Conclusion

Test smells may decrease the test code quality and maintenance. Our study aimed to identify whether practitioners unintentionally insert test smells in the test code and how they treat them. Therefore, we applied two complementary research methods: a survey and an interview study.

We surveyed sixty respondents to investigate the unintentional test smells insertion in the test code. They evaluated a set of practices related to the test smells insertion in the test code. The results indicated that the respondents adopt bad practices that might lead to insert the test smells. The bad practices adoption is more related to the improper company standards than the respondent's experience with test code development.

To investigate how the practitioners treat test smells, we interviewed fifty respondents. They answered questions on

how they prevent and treat test smells during the test code development. The results indicated an overall knowledge lack on the test smells. For most of the interviewees, it was their first contact with this subject. However, after explaining one test smell to the respondents, they recognized it in their test code and identified practices that they adopted to deal with it. Among the recommended practices, we highlight the adoption of tools, coding patterns, programming practices, code review, and training to improve the developers' skills and expertise.

After analyzing the answers to the survey and the interview, we could identify that practitioners did not know test smells. Thus, they insert different test smells types, even the experienced ones. They have tried to treat test smells through some strategies, but as they have not learned about this subject, they have inserted test smells in their test code, and the strategies may not be enough to avoid that. Those studies are starting points to researches that consider practitioners as agents in the test smell treatment.

As future work, we aim to follow the Grounded Theory methodology (Corbin and Strauss, 1990) to leverage a common understanding of how the software industry is receptive to improving the test code quality by taking test smells into consideration. We would validate the respondents' practices to prevent and treat test smells and elaborate a checklist for test code quality development and assurance with an in-depth study.

Acknowledgements

We would like to thank the participants in our survey and pilot study. This research was partially funded by INES 2.0; CNPq grants 465614/2014-0 and 408356/2018-9 and FAPESB grants JCB0060/2016 and BOL0188/2020.

References

Bavota, G., Qusef, A., Oliveto, R., Lucia, A., and Binkley, D. (2012). An empirical analysis of the distribution of unit test smells and their impact on software maintenance. In *28th IEEE International Conference on Software Maintenance (ICSM)*.

Bavota, G., Qusef, A., Oliveto, R., Lucia, A., and Binkley, D. (2015). Are test smells really harmful? An empirical study. *Empirical Software Engineering*, 20(4).

Corbin, J. and Strauss, A. (2014). *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage publications.

Corbin, J. M. and Strauss, A. (1990). Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology*, 13(1):3–21.

Creswell, J. W. and Clark, V. L. P. (2018). *Designing and Conducting Mixed Methods Research*. SAGE Publications, third edition.

Fraser, G. and Arcuri, A. (2011). Evosuite: Automatic test suite generation for object-oriented software. In *13th European Conference on Foundations of Software Engineering*, ESEC/FSE, New York, NY, USA. ACM.

Garousi, V. and Felderer, M. (2016). Developing, verifying, and maintaining high-quality automated test scripts. *IEEE Software*, 33(3).

Garousi, V. and Küçük, B. (2018). Smells in software test code: A survey of knowledge in industry and academia. *Journal of systems and software*, 138.

Greiler, M., van Deursen, A., and Storey, M. (2013). Automated detection of test fixture strategies and smells. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*.

Gubrium, J. F., Holstein, J. A., Marvasti, A. B., and McKinney, K. D. (2012). *The SAGE Handbook of Interview Research: The Complexity of the Craft*. SAGE Publications, 2nd edition.

Junior, N. S., Martins, L., Rocha, L., Costa, H., and Machado, I. (2021). How are test smells treated in the wild? A tale of two empirical studies [Dataset]. Available at: `https://doi.org/10.5281/zenodo.4548406`.

Kitchenham, B. A., Budgen, D., and Brereton, P. (2015). *Evidence-based software engineering and systematic reviews*, volume 4. CRC press.

Melegati, J. and Wang, X. (2020). Case survey studies in software engineering research. In *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '20, New York, NY, USA. ACM.

Meszaros, G., Smith, S. M., and Andrea, J. (2003). The test automation manifesto. In Maurer, F. and Wells, D., editors, *Extreme Programming and Agile Methods - XP/Agile Universe 2003*. Springer Berlin Heidelberg.

Miles, M. B., Huberman, A. M., and Saldaña, J. (2014). *Qualitative Data Analysis*. SAGE Publications, fourth edition.

Palomba, F., Di Nucci, D., Panichella, A., Oliveto, R., and De Lucia, A. (2016). On the diffusion of test smells in automatically generated test code: An empirical study. In *9th International Workshop on Search-based Software Testing*. ACM.

Peruma, A. S. A. (2018). *What the Smell? An Empirical Investigation on the Distribution and Severity of Test Smells in Open Source Android Applications*. PhD Thesis, Rochester Institute of Technology.

Pfleeger, S. L. and Kitchenham, B. A. (2001). Principles of survey research: part 1: turning lemons into lemonade. *ACM SIGSOFT Software Engineering Notes*, 26(6):16–18.

Santana, R., Martins, L., Rocha, L., Virgínio, T., Cruz, A., Costa, H., and Machado, I. (2020). Raide: A tool for assertion roulette and duplicate assert identification and refactoring. In *Proceedings of the 34th Brazilian Symposium on Software Engineering*, SBES '20, page 374–379, New York, NY, USA. Association for Computing Machinery.

Silva Junior, N., Rocha, L., Martins, L. A., and Machado, I. (2020). A survey on test practitioners' awareness of test smells. In *Proceedings of the XXIII Iberoamerican Conference on Software Engineering, CIbSE 2020*, pages 462–475. Curran Associates.

Singer, J., Sim, S. E., and Lethbridge, T. C. (2008). Software engineering data collection for field studies. In Shull, F., Singer, J., and Sjøberg, D. I. K., editors, *Guide to Ad-*

vanced Empirical Software Engineering, pages 9–34, London. Springer London.

Smeets, N. and Simons, A. J. (2011). Automated unit testing with Randoop, JWalk and μJava versus manual JUnit testing. Research report, Department of Computer Science, University of Sheffield/University of Antwerp, Sheffield, Antwerp.

Spadini, D., Schvarcbacher, M., Oprescu, A.-M., Bruntink, M., and Bacchelli, A. (2020). Investigating severity thresholds for test smells. In *Proceedings of the 17th International Conference on Mining Software Repositories, MSR*.

Tufano, M., Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A., and Poshyvanyk, D. (2016). An empirical investigation into the nature of test smells. In *31st International Conference on Automated Software Engineering*. IEEE.

Van Deursen, A., Moonen, L., Van Den Bergh, A., and Kok, G. (2001). Refactoring test code. In *Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP)*.

Van Rompaey, B., Du Bois, B., and Demeyer, S. (2006). Characterizing the relative significance of a test smell. In *22nd International Conference on Software Maintenance*, ICSM'06. IEEE Computer Society.

Virgínio, T., Martins, L., Rocha, L., Santana, R., Cruz, A., Costa, H., and Machado, I. (2020). Jnose: Java test smell detector. In *Proceedings of the 34th Brazilian Symposium on Software Engineering*, SBES '20, page 564–569, New York, NY, USA. Association for Computing Machinery.

Virgínio, T., Martins, L. A., Soares, L. R., Santana, R., Costa, H., and Machado, I. (2020). An empirical study of automatically-generated tests from the perspective of test smells. In *SBES '20: 34th Brazilian Symposium on Software Engineering*, pages 92–96. ACM.

Virgínio, T., Santana, R., Martins, L. A., Soares, L. R., Costa, H., and Machado, I. (2019). On the influence of test smells on test coverage. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*. ACM.

Wiederseiner, C., Jolly, S. A., Garousi, V., and Eskandar, M. M. (2010). An open-source tool for automated generation of black-box xunit test code and its industrial evaluation. In Bottaci, L. and Fraser, G., editors, *Testing – Practice and Research Techniques*. Springer Berlin Heidelberg.

Yusifoğlu, V. G., Amannejad, Y., and Can, A. B. (2015). Software test-code engineering: A systematic mapping. *Information and Software Technology*, 58.

A Appendix A

Block 1: Respondents' Profile

- **Q1.** What is your gender?
- **Q2.** What is your age?
- **Q3.** Which course do you have an academic background in?
- **Q4.** What is the highest degree or level of education you have completed?
- **Q5.** Which Brazilian state do you currently work?
- **Q6.** How long have you been working with software testing?
- **Q7.** How long have you been working with software development?
- **Q8.** Which activity do you perform daily?
- **Q9.** What is the platforms of the projects that you have worked on?
- **Q10.** What is the application domain of the last project that you worked on?
- **Q11.** Which test technique do you execute?
- **Q12.** Are the tests executed more often manually or automated?
- **Q13.** How do you describe your expertise with coding?

Block 2: Test Creation

- **Q14.** What is the source for creating the test cases for the projects in which you work?
- **Q15.** Is there verification to detect duplicate tests (with the same writing or with different writing and the same objective)? *More than one option could be selected.*

Evaluate the following statements according to your daily activities:

- **Q16.** "I usually create test cases using some configuration file (or complementary file) as a backup"
- **Q17.** "When creating a test, I analyze whether it can be executed at the same time with others or if it should be executed in isolation, due to the availability of external resources ."
- **Q18.** "I analyze the possibility of a test failing because it uses a resource that is being used at the same time by another test."
- **Q19.** "I have a habit of creating tests with a high number of parameters (number of files, database record, etc.)."
- **Q20.** "I group different test cases into one (that is, combine tests that could be run separately)."
- **Q21.** "I create tests that depend on resources that may not have their own tests for validation (eg a test that involves retrieving information from the database, but there is no test to validate database research). "
- **Q22.** "I have already created a test to validate some feature that will not be used in the production environment"
- **Q23.** "I have already created a test with a high value for a specific parameter (eg number of records in the database, number of files in folder) even that makes it difficult to repeat. "
- **Q24.** "I have already created a test with a conditional or repetitive structure."
- **Q25.** "I have already created an empty test, with no executable instructions."
- **Q26.** "I usually create tests using some data from a configuration file."
- **Q27.** "I usually create tests with printing or displaying results in a redundant way, or without need."
- **Q28.** "I have already created a test considering the existence of a resource, without checking its existence or availability."

Q29. "I already changed a test by identifying one of the previous points."
Q30. If you answered "always", "frequently" or "rarely" in the previous questions, why were the tests created with these standards?
Q31. If you changed any tests according to the design standards above, why were they edited?
Q32. What problems in the test structure have you encountered?
Q33. What difficulties do you often encounter when creating test cases?

Block 3: Test Execution
Evaluate the following statements according to the frequency found in daily activities:

Q34. "A test case fails due to unavailability of access to a configuration file."
Q35. "Repeat a test case because it previously failed due to competition with some other test case that was running at the same time."
Q36. "Execute tests that could be executed performed more quickly, when modifying the contents of the configuration file."
Q37. "Run a test without understanding its purpose."
Q38. "Some test fails and it is not possible to identify the cause of the failure."
Q39. "Run a test that depends on an external resource that does not have a test for direct validation."
Q40. "A test case fails due to unavailability of access to any external resource."
Q41. "Run test with a high value for a specific parameter (eg: number of records in the database, number of files in folder) even if it makes it difficult to repeat."
Q42. "Run a test to validate a feature that will not be used in the production environment."
Q43. "Find duplicate test (with the same or different writing)."
Q44. "Run test with conditional or repetitive structure."
Q45. "Find empty test, with no executable instruction."
Q46. "Run test with printing or display of results in a redundant way, or unnecessary."
Q47. "Run a test considering the existence of a resource, without checking the existence or availability of it." What difficulties do you usually encounter when running test cases?