The Journal of Systems and Software 141 (2018) 66–84

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

# Towards cognitive support for unit testing: A qualitative study with practitioners

Marllos Paiva Prado[a,b,*], Auri Marcelo Rizzo Vincenzi[c]

[a] Institute of Informatics, Federal University of Goiás – Alameda Palmeiras, Quadra D, Câmpus Samambaia CEP 74690-900, Goiânia, GO, Brazil
[b] Federal Institute of Goiás – Goiânia, GO, Brazil
[c] Computing Science Department, Federal University of São Carlos, São Carlos, SP, Brazil

ARTICLE INFO

*Article history:*
Received 31 March 2017
Revised 15 March 2018
Accepted 21 March 2018
Available online 21 March 2018

*Keywords:*
Unit testing
Cognitive support
Qualitative study
Test Review Practice
Software Testing

ABSTRACT

Unit testing is an important component of software quality improvement. Several researchers proposed automated tools to improve this activity over the years. However, these research efforts have not been sufficient to help the practitioners to address some associated mental tasks. Motivated by this gap, we conducted a qualitative study of professionals with unit testing experience. The goal was to understand how to improve the cognitive support provided by the testing tools, by considering the practitioners' perspective on their unit testing review practices. We obtained the responses from our volunteers through a questionnaire composed both of open-ended and closed questions. Our results revealed some primary tasks which require cognitive support, including monitoring of pending and executed unit testing tasks, and navigating across unit testing related artifacts. We summarize our results in a framework, and based on it, we develop a research agenda as an actionable instrument to the community. Our study's contributions comprise practical improvement suggestions for the current tools and describe further opportunities for research in software testing. Moreover, we comprehensively explain our qualitative methods.



## 1. Introduction

Software testing researchers contribute considerable efforts to solve problems that emerge in testing. The fruit of their research has often taken the form of testing tools, which are made available to industry. Despite this effort, there is a persistent disconnection between industry and research practices (Xie, 2017; Zeng et al., 2016; Fraser et al., 2015; Kasurinen et al., 2010). As part of this, the words "automation", "criteria", "technique" and "software" have frequently been associated with the testing tool research. However, where would the word "human" rank in this list, and what role does it play in guiding research? Although testing tools are built for human use, there is little research about human use of these tools. If we want people to adopt our tools and the underlying techniques, it is advisable to take human factors into consideration.

These considerations came to awareness after observing recurring problems reported in the testing literature and conferences, as well as in informal chats with peers in academia and industry. First, testers face challenges when performing certain mental tasks in testing; this is regardless of the aid provided by the existing supporting tools–e.g. the difficulty of understanding the code under test or the test case itself (Panichella et al., 2016; Ceccato et al., 2015; Fraser et al., 2015; 2013). Second, practitioners are more concerned with defects in the test cases rather than defects in the code (Daka and Fraser, 2014). In the Human-Computer Interaction (HCI) area, such problems are often associated with tools that provide weak cognitive support to their users. Cognitive support can be defined as the assistance that a tool provides to the user in their efforts in thinking, reasoning, remembering, and creating (Walenstein, 2002). As a simple example, consider the LED light that indicates the status of the "CapsLock" key on a keyboard. It helps us to rely on a mechanism other than our own memory, reducing our own mental efforts and the impact of human memory limitations. Although human factor problems are repeatedly indicated in the testing literature, there is a lack of interest in addressing these problems (Prado et al., 2015). This encouraged us to focus our attention on the problems of cognitive support in software testing tools, starting from its lowest level: the unit testing.

Unit testing is popular among professionals concerned with software quality. It involves directly testing the smallest cohesive components of a system. These components include functions,

* Corresponding author at: Institute of Informatics, Federal University of Goiás – Alameda Palmeiras, Quadra D, Câmpus Samambaia CEP 74690-900, Goiânia, GO, Brazil.
*E-mail addresses:* marllosprado@ifg.edu.br (M.P. Prado), auri@dc.ufscar.br (A.M.R. Vincenzi).

https://doi.org/10.1016/j.jss.2018.03.052
0164-1212/

procedures, methods, or classes, depending on the language and paradigm (Perry and Kaiser, 1990; Binder, 1999; Vincenzi, 2004; Colanzi, 1999). The ultimate goal of the activity is to reveal of logical and implementation errors in these elements (Sommerville, 2010; Naik and Tripathy, 2008; Wappler and Wegener, 2006).

Currently, there is no theory on how the professionals perform unit testing activity in practice. Also, the support that the current tools offer are often limited to elimination of repetitive manual effort e.g. test data generation, assertions verification, coverage checking, and so on (Galler and Aichernig, 2014; Ramler et al., 2018; Yang et al., 2009; Alemerien and Magel, 2014). As we further discuss in Section 2, practitioners face problems that go far beyond issues with manual effort. Considering this, two things became apparent since the start of our investigation: (i) the phenomenon of interest was not clearly comprehended; (ii) it required a method of inquiry that could adequately explore the human-related experience from the perspective of those who live them. Thus, we adopted a qualitative approach for our research. According to Creswell (2013a) "In qualitative research, the intent is to explore the complex set of factors surrounding the central phenomenon and to present the varied perspectives or meanings that participants hold." In the present study, the "central phenomenon" is unit testing review. The "complex set of factors" are the tasks associated with the recurring problems. Lastly, the "varied perspective that participants hold" serves as the data that enables us to build a framework using a user-centered approach. This framework will inform future studies, as well as tool design—namely, improvement of cognitive support provided by unit testing tools.

This paper makes the following contributions:

1. A summary of our previous work, which characterizes the problem of cognitive support in unit testing. The present paper is motivated by the findings from our previous studies and extends them.
2. A detailed description of the study design, and the method used for data analysis–Qualitative Content Analysis.
3. A framework of tasks requiring cognitive support for the improvement of the tools regarding the unit testing review practice.
4. A research agenda based upon the findings provided by our framework.

The next Section (2) provides the literature background to: (i) understand the absence of user involvement in testing tool research; (ii) characterize the problem of cognitive support in unit testing activity. Section 3 describes the study design, such as the research questions, the method of analysis applied to the participants' responses, and practical considerations about the survey development. In the Section 4, we report the profile of the volunteers and their perspectives on some problems related to the open-ended questions. Section 5 presents the framework that emerged from our qualitative analysis and a research agenda based on it. Finally, Section 6 concludes and discusses additional study opportunities.

## 2. Related literature

Our choice to take a qualitative approach was motivated by the need to explore the research topic and understand the landscape from the perspective of testers and developers. Very little is written or known about the experiences of unit testing practitioners and their challenges.

Qualitative approaches do not involve the hypothesis formulation and testing that is seen in quantitative approaches i.e. predictions about expected relationships among variables. Rather, researchers use general and broad research questions, so as to not limit the inquiry (Creswell, 2013b). To achieve this goal, researchers use literature and existing studies: (i) to incorporate findings from other studies that are tangential to the investigation; (ii) to compare affirmations of observations from participants on the theme; (iii) to discover problems that have not been studied; and (iv) to understand methodological and theoretical controversies in the field (Flick, 2014; Creswell, 2013c; Corbin and Strauss, 2014). In line with this, the literature review situates our study within the broader context, but it does not set boundaries for interpretation of the participants' perspectives (Creswell, 2013c; Corbin and Strauss, 2014).

In this section, we discuss two previous works in order to frame the present study. Both works reviewed the existing literature, to characterize the problem of human participation in the testing tool research and collect evidence of a cognitive support gap in unit testing.

### 2.1. Humans out of the loop and the cognitive problems in unit testing

The impetus for our present research followed from the publication of (Prado et al., 2015) in a venue recognized by its importance in the software reliability area, with a mixed audience coming from industry and academy. In that work, we used a hunting game metaphor to describe the human aspect of the software testing problem. In our metaphor, the testers are the hunters, bugs are the game animals, testing tools are the weapons and snares, and strategies and tactics underlay both as testing techniques. To succeed, the hunters must catch as many game animals as they can, especially the most valuable ones. A hunter relying only on natural human abilities and strength will not be successful. Specialized weapons and traps are needed for the hunt. But if they are hard to use or not designed to leverage hunter's skills and abilities, they can become cumbersome, time-consuming to master, and in the worst case, useless while offering false expectations. Replacing people with automated weapons is less adaptable, and has weaknesses in finding some skittish and valuable game animals. Promising research into weapon's features or cognitive ergonomics could improve practitioners' performance in discovering bugs.

We found evidence in the literature of the research community working in the opposite direction of the ideas illustrated in our metaphor. For example, in a recent paper, Orso and Rothermel (2014) analyze the state of the art in software testing, spanning the last decade and a half. They summarize a number of research topics and present challenges to be investigated. Their data is from a pair of open ended questions collected from fifty testing researchers. Notably, although there is a wide variety of ideas in the "Challenges and Opportunities" section of their paper, the authors explicitly say that they would not consider "human factors" and "technology transfer" as research challenges, even though they declared that these topics were among those mentioned in their peer's responses. There is no justification for such a decision, and previous sections of their paper present arguments inconsistent with this omission. For example, in the subsection devoted to "Empirical Studies and Support for Them", they mention the need for "user studies" and the predominance of "automation of algorithms and heuristics" in testing research. Also, they show concern about threats to validity that research results are subject to when they fall into the "hands of engineers". Another point of interest in their work is the influence that industry contributions have had on research (a reverse transfer of technology). JUnit (Beck, 2004) is mentioned in their papers as a "straightforward [testing] framework" that exemplifies this influence.

The work of Jia and Harman (2011) is another example of this avoidance of human factors. Although they draw attention to human-related issues, they give little consideration to human participation in the research of testing solutions. They review muta-

tion testing literature and discuss potential research on the topic. The unresolved problem of "barrier[s] to wide application of Mutation Testing" is the "Equivalent Mutant Problem". The "Human Oracle Problem"–the checking of the original program output for each test case generated–is mentioned as another barrier. Also, the authors mention that "more tooling is required to ensure widespread industrial uptake" and that "automated practical tool[s]" for test case generation could be a promising way of creating test solutions. Throughout the paper, the authors discuss connections between automation and mutation testing. However, they make little mention of a human-in-the-loop as part of the solution. Considering that the Equivalent Mutant Problem is recognized as undecidable (Madeyski et al., 2014), and that test case generation requires an as-yet non-mechanized "human oracle", it is hard to presuppose that automation will be a panacea in the vast spectrum of existing software development domains. The implication is that humans are–and will continue to be–a crucial factor to any potential solution for these problems in the short and medium term. Therefore, researching ways to enhance testers' innate abilities in performing these tasks, through tools with cognitive support, would be a way of making the mutation test accessible to practitioners. For example, research could look at what tools and supports would facilitate testers in recognizing mutants that are more likely to be equivalent within a broader set of surviving mutants (perhaps using visual or collaborative supports). This necessitates that we put practitioner in the center of analysis to understand: how the users currently make this kind of identification; which are the attributes that they look for in the mutant when they perform this kind of analysis; which are their primary difficulties, etc. Such detailed studies will inform us of what cognitive supports are required.

Another set of works highlight the disparity between practitioners' needs and existing solutions. Recently, Daka and Fraser (2014) conducted a survey with testers, to investigate problems in unit testing. They found that the principal motivation for conducting tests is the tester's "Own Conviction", which exceeds that of the "Management Requirements" placed upon them. On the other hand, only half of the respondents "attributes a positive feeling to writing unit tests". These findings are aligned with the results of Ng et al. (2004) and the survey carried by Lee et al. (2012). In the former, the "difficulty of using [tools]" was identified as the major barrier to tool adoption, and consequently the testing methods associated with it. In the latter, more than half of respondents indicate that testing is performed based on personal knowledge and without guidance. As indicated by these results, despite being unpleasant work, resilience in the hunt for bugs can be motivated by genuine trust in its purpose.

Altogether, these works demonstrate how the research community has segregated the human aspects from the testing research. In addition, other works from the literature give evidence of cognitive support problems in the unit testing domain.

Atkinson et al. (2010) observed that tools like JUnit still require tests to be written as programs, which is beneficial for its natural integration with the code base. Unfortunately, it also creates dependencies between details of the program language, and the strategy to "test logic, test data, and later test results". Lappalainen et al. (2010) identified difficulties in understanding this interdependence in novice developers, who are concurrently learning unit testing and Test-driven development (TDD). In response to the problem, they proposed a tool, aiming to make test cases more readable and easier to write. Aligned with the previous studies, Daka and Fraser (2014) ask "how could unit testing be improved", and in particular "what makes it difficult to fix a failing unit test". They found that: "the code under test is difficult to understand"; "the test [itself] is difficult to understand"; and "the test reflects unrealistic behavior".

Another remarkable finding from Daka and Fraser (2014) shows that developers tend to treat failing unit tests as defects in the test cases themselves, more often than as defects in the tested code. The study of Daka and Fraser (2014) also reveals that the discovery of "crash" type failures and the complementing of the manual tests are two popular uses of automated test. According to Leitner et al. (2007), despite the effortless nature of automated test generation, automation is not a substitute for manual testing, because developers are experts on defining complex input data and building effective test cases. Both works reveal that manual unit tests can be important in areas with less obvious errors (e.g. "non-crash" types). Moreover, manual and automated testing are current and non-exclusive practices.

The previous observations give us pause for reflection: manual and automated unit testing are both current, important and complementary. They are somehow supported by the existing tools. However, comprehension problems have commonly affected the practitioners. Thus, how do current tools fail to aid the practitioners in unit testing comprehension?

Runeson (2006) conducted a study to characterize unit testing in the industry. The study included one round of focus group with participants and another with their respective companies. One of the findings showed that not even the definition of what constitutes a unit under test is agreed upon. Although the companies' questionnaire responses indicated that the units were formed of individual elements, the focus group revealed that it was fairly common to consider the unit under test as a set of cohesive linked elements. Eight years later, Daka and Fraser (2014) revealed that practitioners still consider the identification of which code to test, and the isolation of the unit under test, as problems that affect the writing of unit tests. The studies of Runeson (2006) and Daka and Fraser (2014) also share similarities regarding the difficulties in evaluating unit tests. The former study indicated a need for clear and measurable test quality criteria, and the second revealed "what to check in a test" as a common difficulty in writing new tests.

Lee et al. (2012) found that the usage of testing methods and tools in unit testing is especially weak, compared to other levels of testing. In fact, unit testing frameworks do not impose the application of any particular testing strategy or criterion. In this sense, they are like a blank slate where developers may define test cases in their own way. Runeson (2006) also found that due to the lack of strategies at companies, test cases were defined using the personal judgment of developers, "leading to varying practices". Half of the respondents of the study conducted by Lee et al. (2012) corroborate Runeson's finding by declaring that they "perform their software testing based on individual's know-how or personal knowledge without standardized guidance". Daka and Fraser (2014) found similar results. According to them, "developers claim to write unit tests systematically and to measure code coverage, but do not have a clear priority of what makes an individual test good".

The concerns reported in the studies of Runeson (2006); Lee et al. (2012); Daka and Fraser (2014) are related to uncertainty about how to perform unit testing. This leads us to ask whether the tools could be changed to improve cognitive orientation towards strategies and methods, in order to mitigate such problems. Other opportunities for investigation are the apparent lack of enjoyability and ease-of-use of unit testing tools'.

Runeson (2006) acknowledged the difficulty of motivating developers to perform unit testing. Daka and Fraser (2014) corroborate this idea, revealing that only about half of developers have a positive feelings about writing unit tests. They suggest tool improvement as a solution to this. Kasurinen et al. (2009) revealed in his study–dedicated to understand the problems of software testing in practice–that the usability and applicability of testing tools is identified, by the companies surveyed, as one of the factors that

affect the efficiency of testing activity. In particular, larger companies reported that false positives generated by complicated or defective tools cause additional expenses. Wiklund et al. (2014), point to difficulties in configuration and use of the IDE (Integrated Development Environments)–in particular Eclipse–as a barrier commonly faced by testers when writing test code in automated testing environments. In addition, Lee et al. (2012) claim that the relationship between software testing tools and their corresponding activities should be made more clear.

Delahaye and du Bousquet (2015) defined guidelines for selecting and comparing software engineering tools. They chose mutation testing–one of the techniques for unit testing–as their case study. Their usability criteria consisted of four categories: compatibility, interface, documentation, and surviving mutants management. In the "interface" category they highlight the difficulty faced by users when operating these tools. Among the eight mutation tools they investigated, only two presented proper Graphical User Interfaces (GUI), the lack of which can, for example, hamper the inspection of the mutants. Additionally, most of them do not integrate with popular IDEs. In "documentation" they looked for clear descriptions of the operations and processes executed by the tools. Only three out of the eight tools analyzed are ranked as having "very good" documentation, whereas five are ranked at the lowest level, for the inferior quality of the "surviving mutant report" generated.

Based on the reported problems, tool usability could affect user performance in unit testing. If so, we need to explore what attributes are related to these problems, and how they could be evolved to improve the overall ease of use of the tools.

Based on these lessons learned from the literature, we summarized the cognitive demands into an initial framework, to offer direction to our investigations. The framework is composed of the following three dimensions:

(i) "Tester Artifact Comprehension" dimension–that addresses factors which affect users in the task of abstracting and understanding unit testing artifacts;
(ii) "Tester Orientation and guidance" dimension–that addresses factors which influence user's decision on unit testing tasks; and
(iii) "Tool Interaction Usability" dimension–that addresses usability aspects in the tool's interface.

### 2.2. A closer look at the unit testing tools and cognitive supports

After characterizing of cognitive support problem, we expanded our analysis towards unit testing tools that offer visualization facilities (Prado and Vincenzi, 2016). Visualizations leverage the strength of human vision and mental image processing to overcome barriers related to data manipulation, and is associated with cognitive advantages (Card et al., 1994). Thus, we made a retrospective analysis of visualizations (in unit testing tools that provided them), to see how they addressed the cognitive support issues we identified.

We identified works for analysis using a systematic mapping. In the systematic mapping (see Fig. 1), we identified 64 works that applied visualization to software testing in general. From this total, we selected the 12 works that matched the unit testing level.

We grouped the selected works by similarity, and analyzed them under the framework dimensions indicated in Section 2.1. We discovered some cognitive support related problems. For example, some visualizations that addressed structural unit testing, like Muto et al. (2011) and Karam and Abdallah (2005), were susceptible to *expressiveness* problems–i.e. they unintentionally transmitted more information than intended, causing misinterpretation of results by users.

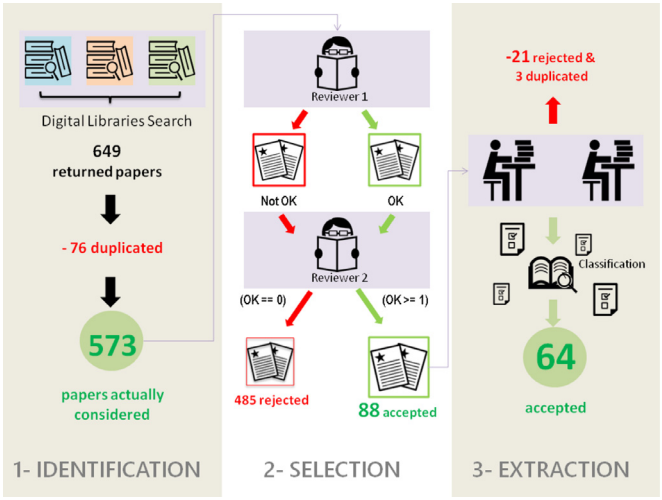


**Fig. 1.** Infographic summarizing the systematic mapping process.

There was a lack of user studies early research stages, in all the proposals with cognitive support problems. Despite that, we also found two exceptions–Cottam et al. (2008) and Lawrence et al. (2005)–that exemplify the importance of human involvement in the different stages of the research process.

In the work of Cottam et al. (2008) for example, the target audience was considered in the initial design steps. This involved: understanding how information was previously obtained from raw data; classifying users into groups; and understanding the information needs for each group and prioritizing them. The users' opinion impacted many choices made during the process of visually mapping the data, such as: the visual metaphor applied; colors and color tones adopted; and the double-encoding of the data to solve particular cases where the information comprehension could be affected. The authors also evaluated the visualization regarding its expressiveness and effectiveness for the target users. This evaluation included questions to: (i) compare user interpretation using that visualization versus the pre-existing textual mode; (ii) discover user insights facilitated by the visualization; (iii) determine the capacity of the user to reach goals using the visualization; (iv) check improvements in data representation and user assistance.

Lawrence et al. (2005) carried out an experimental study with users to understand the influence of visualization on structural unit testing, assessing user performance and behavior. One of their result did not show significant differences in the number of faults found between the treatment group (using visualization of block coverage) and the control group (not using visualization for the same criterion). A second result showed that the number of test cases generated was not affected by the adoption of a visualization, although the variability in the amount of test cases wrote was reduced in the treatment group. Finally, another result suggested that the visualization can affect the users' behavior by encouraging them to overestimate the effectiveness[1] of their tests. According to the authors, this was possibly due to the positive visual feedback obtained when the criterion adequacy was reached.

### 2.3. Other works related to cognitive problems in software testing activity

There are other works in the literature that evidence cognitive problems associated with testing activity and the importance of user involvement in the research of supporting solutions.

[1] Used as in the original work i.e. to indicate the capacity of finding errors.

Ricca et al. (2007) discuss the difficulties faced by programmers in understanding requirement change requests during software maintenance. According to them, these change requests are often inadequately reported and prone to result in software faults. In their study, they ran an experiment to assess if FIT tables—conceived to facilitate acceptance tests—affects the level of understanding and productivity in understanding the requirements. Their results align with Melnik et al. (2004) and show that although FIT tables help improve requirement understanding, the benefit comes at additional effort for users.

Ceccato et al. (2015) notice that most of the literature regarding experimental studies with testing and debugging does not involve human subjects. Ceccato et al. use the work of Frankl and Weiss (1991) as an example: although their work compared the effectiveness of different structural criteria to reveal faults, it did not consider the intensive human effort involved in pinpointing the faults after their discovery.

The experiments conducted by Fraser et al. (2015) aimed to shed light on the assumption that automatically generated unit tests facilitate the task of testing for developers. The result of their studies revealed that despite the improvement of code coverage by automatically generated tests, there was no indication that these tests help to find more bugs, compared to manual tests. Another finding of the study was that developers spent considerable time analyzing the poor tests generated by the automatic tool–to improve them or to realize some should be discarded. According to the authors, this result indicates a need for change in the way these tools are designed. They call for tools capable of producing more "understandable" tests and warn for the "lack of human subject study [... ] to determine what factors impact human understanding of tests".

Part of the work of Rojas et al. (2015) comprised a think-aloud observation of five professional programmers. The researchers investigated how the participants interacted with an automated unit test generation tool (EvoSuite) during development and testing activities. As part of their results, the researchers identified the need to improve the tool's usability and its integration into the software development environment–e.g. "by providing an automated mechanism to combine existing test suites with automatically generated ones".

The work of Panichella et al. (2016) proposed and empirically evaluated a tool (TestDescriber) which generates natural language summaries of automatically generated JUnit test cases, in order to facilitate their comprehension by developers. The participants of their study were both professionals, researchers and students that had at least three years of prior experience with Java and JUnit. The findings of their study indicated that such approach had a substantial impact in helping the participants to better comprehend the test cases and find more bugs.

Recently, the work of Jahangirova et al. (2016) addressed the problem of supporting tester in the definition of more accurate oracles. They proposed a tool and technique for "the generation of counterexamples as test cases that demonstrate incompleteness and unsoundness [of the oracle], which the testers use to iteratively improve the assertion oracle". Although their proposal looks promising, their study depends on further replication to improve the generalizability of their results. The authors recognize "human tester in the loop" as an important component of their tool and technique for the solution of the oracle accuracy problem. In line with this, it would also be important to know if–and how–the authors considered the "tester in the loop" would inform design of their tool and technique.

Finally, (Burnett et al., 2009) discuss the use of "distributed cognition" as cognitive support improvements for programming tools, as used by end-users. Distributed cognition "extends the reach of what is considered cognitive beyond the individual to encompass interactions between people and with resources and materials in the environment" (Hollan et al., 2000). One of the proposals of Burnett et al. is a mechanism to assist user's cognition about things to test, and things tested, during spreadsheet programming. In this scheme, the user checks off (validates) cells that have correct values. Then, these validations are processed in the background to evaluate the testedness state of other spreadsheet cells that are dependent on the validated value (using a test adequacy criterion). The result is presented to the user in the form of colored cells that indicate values that are correct (blue), cells that need attention (red) and values that depend on both correct and untested values (purple). They aimed to encourage end-users to have a disciplined behavior of debugging and testing during spreadsheet programming. In another work, the authors proposed a prototype to support end-user to debug machine-learning logic of applications that run on their computers. Their prototype involved an application that automatically files the user's incoming email messages into correct folders. In their prototype, the user "debugs" the logic by viewing and changing answers to questions given by the system. This feedback mechanism is used to update the result of the input (email message) that the user is working with and "apprises him/her of how other messages in their email system would be categorized differently given these changes".

## 3. Study design and execution

In this section, we discuss our choice of methods for our study, and provide details on the research planning, design and execution.

### 3.1. The Qualitative Content Analysis

The qualitative approach is appropriate when a research problem needs exploration, in order to understand a group of people, to identify variables that are not known (or easily measured) and to hear to silenced voices. It is suitable for contexts where theory is scarce or nonexistent, and a complex and detailed understanding of phenomena is necessary. The researchers attempt to make sense of these phenomena from the participant's perspective. The findings of a qualitative study are richly descriptive. They are usually in the form of categories, themes, typologies, tentative hypotheses or substantive theory rather than derived postulates or the acceptance/rejection of hypotheses (as in positivist research) (Creswell, 2013a; 2007; Merriam, 2002; MORSE, 1991). In contrast to quantitative research, qualitative research rarely defines an explicit hypothesis in advance. In fact, the hypotheses are generated and matured over time–with the data collection and analysis process–or they emerge as part of the study's results (Jr and Unrau, 2010; Henderson and Carter, 2009).

In our study, we adopted the Qualitative Content Analysis (QCA) to analyze practitioner responses. QCA is an established qualitative method used to examine and extract meaning from a collection of textual data (Mayring, 2014). It adheres to the naturalistic paradigm by iteratively developing and revising understanding–in contrast to the positivist approach of checking previous conclusions of theories (Brod et al., 2009). Its fundamental features include the systematic coding of data, as well as the interpretation and the derivation of categories or themes, to elucidate a phenomenon of interest in a social context. Textual data can come from different sources such as narrated data, open-ended surveys, interviews, focus groups, and other types of textual media. While one could find similarities between QCA and Grounded Theory (GT), we instead highlight some fundamental differences between them, both for the research process as well as for the types of results they produce. We will describe these differences in response to the discussion raised by Stol et al. (2016) regarding the misuse

of the grounded theory–named by them as "grounded theory *à la carte*".

Grounded theory is a whole methodology with a strong philosophical background. In practice, it usually requires a lengthy process, operates inductively, subsumes varied sources of data and can be applied from at least three main paradigms, each one with its nuances (Adolph et al., 2011). As the name indicates, its final goal is to build a theory that is grounded in the process adopted in a study's early stages, in particular during the study's design and data collection. Some of its distinctive features are the ability to observe, analyze and adjust the questions while the answers are obtained (Theoretical sampling) (Glaser and Strauss, 1999).

The work of Stol et al. (2016) is very informative, and it discusses the inadequate use of "grounded theory" label for many publications in software engineering area. They explain the high complexity involved in adequate execution of Grounded Theory (GT), and the nuances of its three main variations. They indicate that GT is suffering from arbitrary and confused use in software engineering research. Throughout their paper, they show how publications are not meeting the criteria to be designated as GT, and provide useful hints on what should be considered when using different GT variations, and when presenting such research results.

In contrast, QCA is a method to interpret a collection of textual data. It is recognized for its utility in reducing data, as well as for being systematic and flexible (Flick, 2013). QCA has a rather straight-forward process in comparison to GT. It accepts both inductive ("conventional") and deductive ("directed") coding for the process of category development. It allows for interpretation of the manifest or latent content of the textual data i.e. coding the explicit or implicit meaning of the textual data (Mayring, 2014; Hsieh and Shannon, 2005). It does not aim to reach the full development of a theory–as intended by GT–but to reveal categories from data that could be helpful for answering the proposed research questions.

Despite the fact that QCA is a qualitative method, it was partially inspired by Quantitative Content Analysis, and preserves some benefits of that method, such as: (i) determining what aspect of the communication should be considered during the process of analysis e.g. communicator experiences, context where the text was produced, participants' personal background; (ii) adopting a pre-established process to analyze the material and fragmenting it into units of analysis; (iii) conducting a careful category-oriented analysis through feedback loops; (iv) adopting criteria of reliability to make the results trustworthy and comparable (Mayring, 2014).

### 3.2. Participants contextualization and the mechanism of inquiry

The different testing levels demand different kinds of analysis from the testing practitioners. For example, for unit testing, the practitioners focus their attention on the units and their internal contents, while in integration testing they attend to the interfaces of these elements and how they orchestrate their communication. The responsibility of performing unit testing is commonly assigned to the developers. The assumption that developers have a fine-grained knowledge of the units under test motivates this choice. Thus, this understanding of the units abbreviates the cycle of testing, reporting, debugging, fixing, and re-testing. Considering this, our study admits both professionals exclusively dedicated to quality assurance tasks and as well as developers who perform in that role. The only requirement is that the volunteers have previous experience with unit testing.

The maximization of the number of participants and valid answers is a major concern in our study. This strategy contributes to the analysis of a more robust and insightful set of data. As a consequence, we need to make a compromise between allowing many volunteers to join our study, and the quantity of information that we capture and interpret from each of them. An online survey that could be freely accepted and answered–whenever and wherever the volunteers decided–would be adequate for volunteer participation. However, it also impacts the level of interpretation used in the coding process, since we cannot capture aspects such as the body language of the participants, or the intonation in their speech. Therefore, the main aspect of communication to be considered during our analysis is the textual information directly provided by the participants. We use the closed questions to obtain information about the participants' profile and to discover their perspective on the problems.

#### 3.2.1. Survey elaboration

The creation and application of a survey is a laborious process. The researchers are often involved in the investigated topic for a long time, and develop particular considerations. Conversely, they need to develop a questionnaire that is easy to be answered by respondents who do not have the same background. In this subsection, we share some of our experience accumulated during this important part of the research. The preparation of a mechanism of inquiry is an important part of any qualitative study. We will offer a clear view of the process that we followed to develop our survey. Moreover, it can be useful for researchers who may face these common issues in similar studies.

The first–and possibly hardest–issue was the wish to hasten the fine-tuning process of the survey questions. Qualified volunteers are usually hard to find, and therefore are a precious resource to expend, so we do not want to waste the opportunity to optimize our questions. As a consequence, we needed to adjust the survey questions in a way that they were sufficiently open to let the participants feel free to be expressive their answers. On the other hand, the questions should be constrained enough to prevent respondents from rambling. Our initial concern was to define this trade-off in advance for each proposed question. However, after the first pilot run, we observed that such concern was too restrictive to result in good questions. Rather, we realized that we needed to allow for more freedom. We decided that our priority should be to create insightful questions i.e. questions that indeed addressed the problem of interest. As a consequence, the number of questions, their type (open-ended or closed-ended), the conciseness of wording, or other structural concerns, were not priorities at first. Instead, the questions should look interesting and catch the volunteers' attention. This approach enabled us to develop new questions and pilot test them again. Fortunately, the answers obtained after that showed promise: while we certainly needed to remove and refine many of the questions, the respondents were clearly understanding the inquiries and providing rich and unambiguous answers relevant to our interests.

Our second concern was to determine the role of the closed questions. Closed questions are quick to answer, the researchers can specify possible answers, and respondents only have to choose which answer best fits their opinion. However, such ease of use hides a problem: this type of question frequently quantifies a phenomenon rather than revealing its characteristics. Our final goal was not to understand how the participants' opinions were distributed. Instead, we wanted to understand an unexplored phenomenon, based on the respondents' perspective. As a consequence, the questions should lead to data that can offer us insights. With this goal in mind, we decided that the closed questions would only be used to learn the general demographics of the volunteers, and their general perceptions. The purpose was two-fold: (i) to enable future related research with participants having a similar profile—e.g. for the purpose of comparison or triangulation; (ii) to get an idea of the participants' point-of-view on the issues before the qualitative analysis;

We also had to keep the volunteers engaged during the survey answering process. This issue was important in our study because the questionnaire was provided online, and volunteers were free to end their participation at any time. Also, any decrease in answer quality, over the course of the survey, could risk the quality and trustworthiness of the coding phase. After several refinements (e.g by merging similar questions, making them more cohesive and concise) the final version of our questionnaire had 24 questions—half open-ended, half closed questions. We divided the survey into four pages. The three first pages contained the core questions. The last one was for comments, suggestions and contact information, which were not to be used for analysis. We also positioned closed questions between open-ended whenever possible, instead of dividing them into two distinct groups. The underlying strategy was to reduce fatigue for each kind of question. Moreover, we wanted to offer the opportunity for the volunteers to take pause and elaborate their open-ended answers, to prevent them from getting into a repetitive rhythm.

The survey questions can be accessed in: (Prado and Vincenzi, 2017).

### 3.3. Research questions

Unit testing can involve several different sub-activities—called practices—e.g. the creative process for the test case formulation; the strategies for choosing and applying different testing techniques (structural, error based) over the units; the methods for reporting the unit testing results; the collaboration tasks among practitioners for unit testing management in a project; the strategies to integrate unit tests into the project code base, etc. Moreover, there are a multitude of tools associated with each different practice, making it unfeasible to address all of them here.

In this study, flaky test identification is one of the problems that we want to explore. A flaky test is a test that can pass or fail somewhat unpredictably, making it untrustworthy, and can be the result of mistakes made during test generation. The problem of test flakiness has received considerable attention from the research community (Luo et al., 2014; Gyori et al., 2015; Eloussi, 2015). Thus, we want to know if tools could leverage practitioner skill in identification of flaky tests, after such unit test have been created.

We focused and limited our study on exploring unit testing review practice. Unit testing review includes any task performed by the practitioners that involves checking or verifying existing test case code, their results, and their properties.

Research question elaboration is fundamental for planning of qualitative studies. We formulated the following research question and sub-questions to orient our research:

RQ1. What are the unit testing review tasks that require cognitive support fromtools, to aid testing practitioners?

- RQ1.1 What are the tasks (if any) that require tool support for comprehension of unit testing artifacts?
- RQ1.2 What are the tasks (if any) that require tool support for orientation and guidance during unit testing review?
- RQ1.3 What are the tools' features (if any) that can be improved to better support user interactions with the GUI during unit testing review?
- RQ1.4 What are the tasks (if any) that require tool support for identification of flaky tests during unit testing review?

### 3.4. The coding planning and implementation

The coding process in QCA is not guided by a linear and exacting process such as hypothesis-testing within a positivist-quantitative approach. Unlike quantitative methods, the goal of a qualitative study like QCA is to use data to discover the dimensions of the studied phenomenon–as opposed to understanding how the data is distributed in those dimensions of a previously understood phenomenon. When using QCA, the analysis pathway is most influenced by the experience of the research team with qualitative methods, the self-criticism maintained during the coding process, and the quality of the obtained responses. Nevertheless, QCA is more systematic than grounded theory, and practices that are valid for grounded theory are likely inadequate for QCA. Considering this, we adopted a set of high-level steps (Fig. 2) inspired by Mayring (2014) inductive category development process.

As stated in Section 1, the main purpose of our study is to reveal cognitive support needs on the basis of the perspectives of practitioners. As a consequence, the categories should emerge directly from the participants' responses, in an inductive form. However, as observed by Mayring (2014), even in an inductive approach "the level or theme of categories to be developed must be defined previously. There has to be a criterion for the selection process in category formation. The criterion is a deductive element and established within theoretical considerations about the subject matter and the aims of analysis". In this sense, our previously defined framework (Prado et al., 2015) sets the theoretical foundations of our inductive category development.

For the first step of the process, we define one central research question with four associated sub-questions (see Section 3.3). The question formulation follows the recommendations of Creswell (2013a) and Miles et al. (2013) which include:

- the definition of the central question in a broad and exploratory way;
- a focus on the concept that we wanted to understand most;
- the use of exploratory verbs in the research questions;
- a specification of who are the participants;
- the association of the sub-questions to the central question for the purpose of clarity and specificity.

The cognitive dimensions from our previous framework (Prado et al., 2015) underlay the formulation of our sub-questions. The proposed sub-questions highlight important aspects of our study, and simultaneously leave the questioning opened. In this sense, they can be seen as "lenses" that help us to remember and notice important characteristics of the researched problem, rather than "boundaries" that restrict our analysis.

For the second step, we structured our coding-frame according to the following recommendations from the literature:

- the codes to be included in a given category should have "similar meaning and connotations" (Weber, 1990);
- an emerged category must be part of the phenomenon addressed in the research questions (Mayring, 2014);
- the subcategories included in a given category should be "mutually exclusive and exhaustive" (Crowley and Delfico, 2013).

Moreover, we establish the following category definition and level of abstraction for our category system:

Category definition–*tasks related to: difficulties involving decisions about unit testing artifacts; issues involving reasoning about unit testing artifacts; necessity to cope with unit testing review issues outside the testing environment; friction using the testing environment GUI; problems regarding flaky test occurrences*

Level of abstraction–*difficulties regarding personal and concrete experience; issues that could or should be subjected to tool support; problems affecting the unit testing level, even if not exclusively.*

#### 3.4.1. Coding of the material

The third step is the stage when the coding starts. In our study, we analyzed 58 valid surveys. We considered the first 10 responses

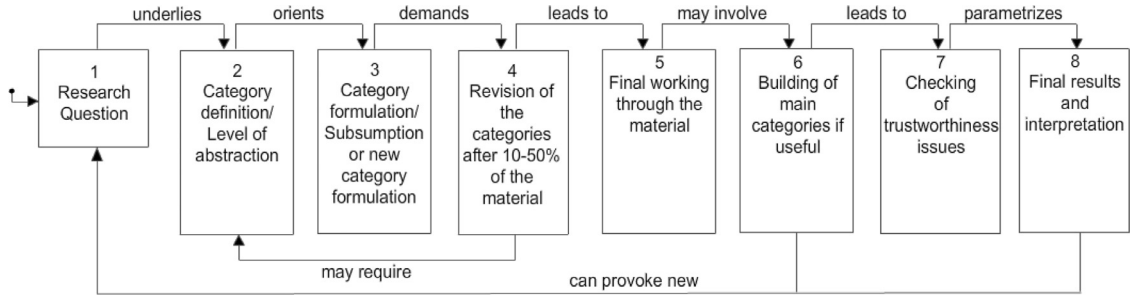


**Fig. 2.** Process model for inductive category development (redrawn and adapted from Mayring, 2014).

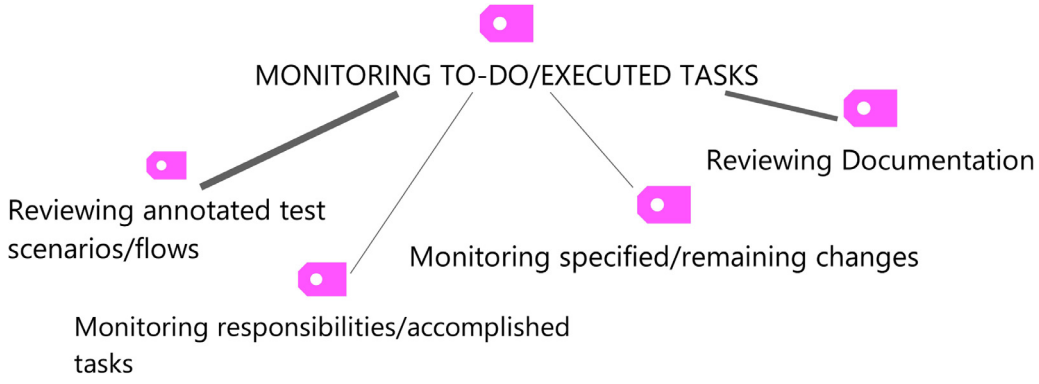


**Fig. 3.** A branch of the final category system, containing one upper-level category and 4 subsumed categories.

in each survey–from the total of 12 open-ended answers–for coding purposes. The two last open-ended questions only collected personal information from the participants. We read the set of 10 answers of all participants as well as the results from the closed-ended questions, before starting the coding process. We took this approach to become immersed in the material collected. We defined our coding unit as any clear semantic element in the text–be it a single phrase or a sequence of clauses. We chose this "dynamic" way of segmentation because each participant elaborated their answers with differing levels of detail.

We coded responses from each question of the questionnaire for the whole group before proceeding to the next question. This allowed us to contrast the participants' perspectives on the problems, as they were raised by each question. It also helped us to perceive any common factors present for each issue, and it was especially useful in the initial iterations–when the first categories emerged. This strategy is aligned with the idea of "build[ing] the coding frame[2] for one 'chunk' [of the material] after another" (Schreier, 2013).

We use one branch of our category system diagram to illustrate how part of the coding process evolved (see Fig. 3). Initially, some coded segments led inductively to the categories represented by the leaf nodes–"Reviewing annotated test scenarios/flows", "Monitoring responsibilities/accomplished tasks", "Monitoring specified/remaining changes" and "Reviewing documentation". However, these categories did not emerge all at once. We iteratively and collaboratively reviewed each newly proposed category. We double-checked whether a coded segment fit any of the existing categories in the entire category system. If not, we used the criteria set by our pre-defined category definition and level of abstraction to ensure that the new category would be appropriate for the category system. If we saw that categories were closely related, we considered whether we should create an upper-level category–in this example "Monitoring to-do/executed tasks" – which could subsume them. This also required verification that the categories to be subsumed had reached some level of consistency; that they were indeed cohesive regarding the upper-level concept proposed; and that they were mutually exclusive.

We reviewed the coded material as our fourth step, i.e., when a fair number of categories emerged in our first coding attempt. At that point, we noticed that some categories needed to be fixed due to an overlapping of concepts—we were unconsciously interpreting two distinct ideas as a single one. Mayring (2014) recommends that the category definition and level of abstraction should be reviewed when overlapping concepts or categories are detected. If they need modification, analysis of material should be started from the beginning. Thus, we adjusted the level of abstraction, dropped away the set of categories obtained up to that point, and restarted the process. This decision was helpful to re-establish our confidence in obtaining a consistent category set. Moreover, it helped reinforce the importance of reviewing the category formation during coding. We took some additional steps to mitigate the chances of having the same problem of overlapping concepts again: (i) we waited a few weeks to recommence the process; (ii) we analyzed each question in a different order than during the first attempt; (iii) we randomized the answer set for each question. This strategy helped us to clear the previous coding process from our minds—or at least make it less vivid.

The coding process was repeatedly reviewed and checked (steps 5 and 6) until we reached saturation i.e. when the emerging categories were exhausted and there was nothing to add or refine in the category system. Fig. 4 shows the category system that emerged from our analysis. A copy of the coded responses is available online at: (Prado, 2017).

The steps 7 and 8 are further discussed in Section 5, which discusses the study results and trustworthiness considerations.

[2] In our work we refer to the coding frame as the category system, following the nomenclature of Mayring (2014).

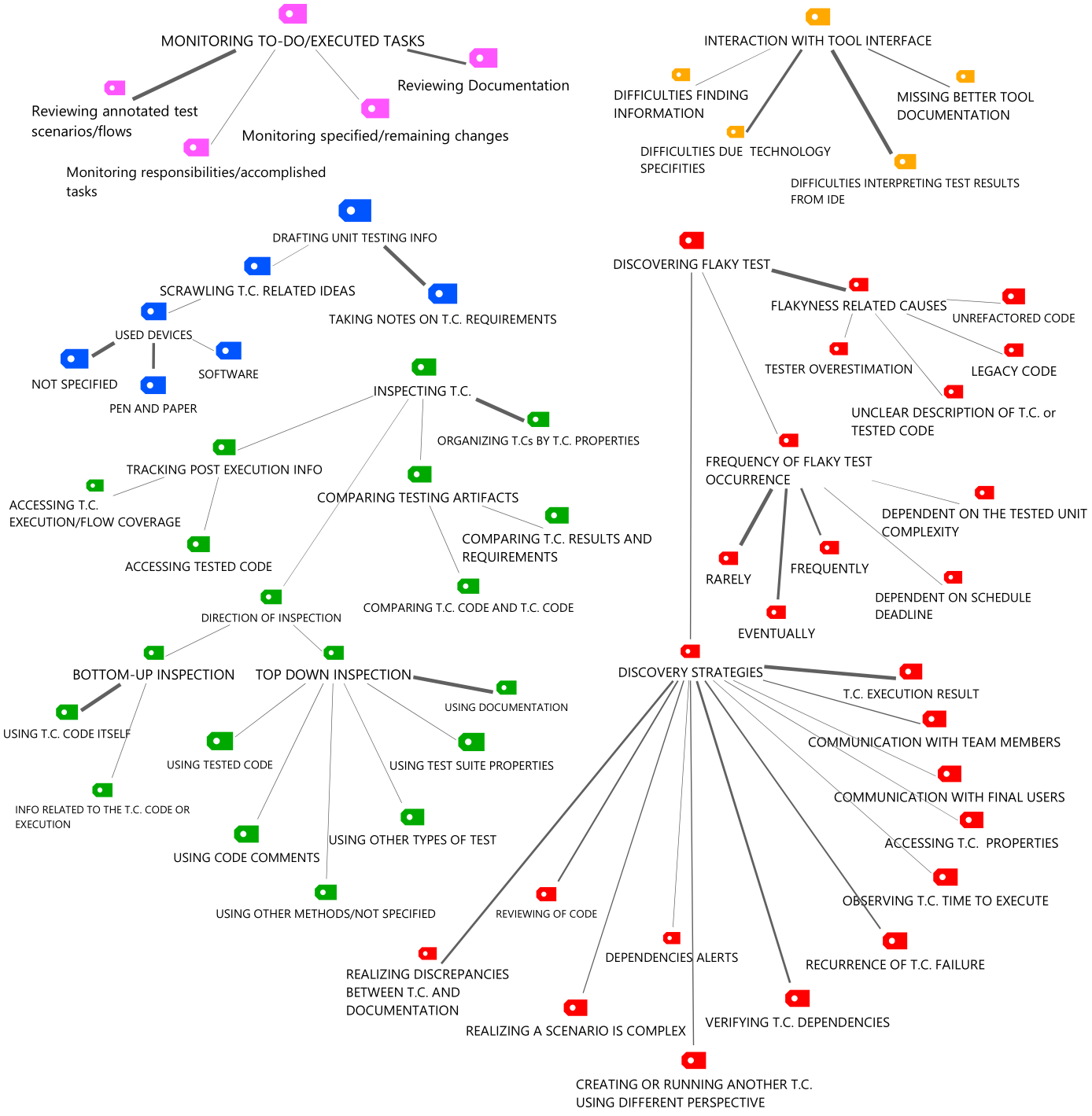


**Fig. 4.** The final category system. The thicker the edges, the more coded segments exist inside the child node category.

## 4. Profile and perspective of the participants

An analysis of the closed-ended question results gave us an overview of how the participants, taken as a group, face some problems related to the open-ended questions. This analysis, plus a careful reading of all open-ended answers, helped us to get immersed in the data before proceeding to the coding process. As noted before, our only sampling criteria for participants was that they have previous experience with unit testing. However, future studies wishing to extend or triangulate with our results may want to select participants that share a similar opinion to our volunteers. In this sense, the closed-ended question results can be reused as criteria to choose participants that are comparable to ours. We present the results from our closed-ended questions in the following subsections.

### 4.1. Participants' demographic information

We recruited participants by following two different approaches. For the first approach, we posted invitations in a professional social network and online forum for developers. For the second, we sent direct e-mails to our peers in the Information Technology sector, so that they could forward the letter of invitation to potential candidates in their circles. We received a total of 44

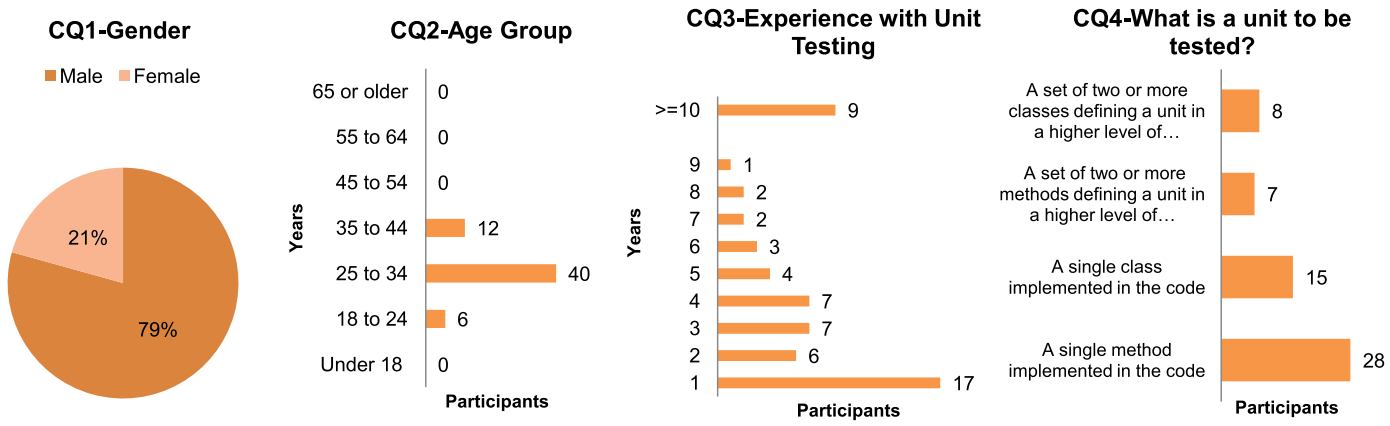


**Fig. 5.** Charts with the demographics of the volunteers that participated in our study.

replies via the first approach and 17 in the second. However, three responses from the second group were considered as invalid and were discarded–summing a total of 58 valid replies.

The charts in Fig. 5 show the demographics of our valid participants. The majority of them are male (79%). Most of the respondents (68.9%) are between 25 to 34 years old and none of the participants is more than 44 years old. Almost one-third of the volunteers (29.3%) have one year of experience with unit testing. The number of respondents with higher experience diminished as the total years of experience with unit testing increased. Since we could not predict the maximum number of years of experience, we simplified by not distinguishing the participants with 10 years or more of experience (15.5%). We also wanted to know how they usually define a unit to be tested. The majority of the participants opted for a more simplistic definition, defining the unit to be tested as a single method or class implemented in the code (total of 74.1%) rather than defining it as a set of these elements.

### 4.2. Participants' perspective on the problems

Problems with motivation during unit testing were reported by Runeson (2006) and Daka and Fraser (2014). We wanted to know how our volunteers perceived their motivation while reviewing their unit tests. The chart in Fig. 6–CQ5 shows that reviewing unit tests is not a cause for excitement, nor is it a tedious practice. Looking at the CQ5 and CQ6 data (Fig. 6–CQ5 x CQ6), the chart reveals that, in general, the participants have an evenly distributed opinion on how much the tool's GUI contributes to their motivation. The participants that answered "Very Motivated" had more clear attribution of their motivation to the tool's GUI. Although not surprising, the previous observations gave us some confidence regarding the remaining responses' reliability. The middling answers to the first two observations above showed us that most volunteers were not likely to be positive or agree with whatever we asked them. This gave us confidence that an "acquiescence" bias Knowles and Nathan (1997) was not affecting the participants. The "sink or swim" attitude noted in the third observation showed us that the "Very Motivated" participants were not concerned with presenting themselves in the best possible light. This observation gave us confidence that a "social desirability" bias Dodou and de Winter (2014) was not affecting the "very motivated" participants.

In the programming psychology area, "mental model" is the concept describing how the programmer develops understanding when reading program code. According to this idea, "understanding a program is equivalent to constructing a detailed representation of the situation" (Detienne, 2001).

Code comments are resources that support the reader in building a mental model of the program. Unit tests are constructed in code and are therefore programs, so we proposed two closed-questions to understand if, in the participants' opinion, the test case implemented by themselves (CQ7) and by others (CQ8) are usually commented. The responses were slightly balanced between "yes" and "no" for commenting the own test case code (CQ7). On the other hand, the answers for CQ8 showed a different perception: 71.9% responded that the test cases not created by themselves, in general, are not commented. The previous two observations could be more precise if we were able to make a direct measurement of the code comments present in each volunteers' routine. However, such measurement requires access to a fair amount of code from our volunteer's organizations and their co-workers during an extended time, and creates privacy and intellectual property concerns. This measurement is beyond the scope of our study, which focuses on practitioner experiences. Considering that the primary purpose of the closed-questions is the characterization of our participants–and not their artifacts–the obtained results are sufficient to understand how our volunteers: (i) perceive the use of this immediate support in their own test cases; and (ii) perceive the use of the same support when the test cases come from third-parties.

The mental model distinguishes two types of mental representations during the program comprehension: the program model and the situational model. In short, the program model representation corresponds to the superficial and explicit information in the textual code and its structure (implementation aspects). This can be seen at two levels: micro and macro. At the micro-level, the program model represents the elementary operations and the control-flow between these operations. At the macro-level, it represents the methods and larger structures. The situational model representation corresponds to high-level aspects (functionality aspects) that can be static or dynamic, for example: the problem domain objects; the relationship between the objects (e.g. inheritance, composition); the goals of the program; and inter-object communication. Considering this, we proposed an otherwise hidden open-ended question (Q7.1) for those that answered "yes" to CQ7. The question asked if there were any details—implementation or functionality details—that the participant usually prioritized when commenting their unit test code. We classified the answers into four distinct groups: "Functionality prioritization (Fun)" (17/33), "Same priority for implementation and functionality (ImpFun)" (5/33); "Implementation prioritization (Imp)" (5/33); and "Not clearly answered (NCA)" (6/33). According to these results, the majority of our participants that comment their own test case code prioritize the functionality of the test case in

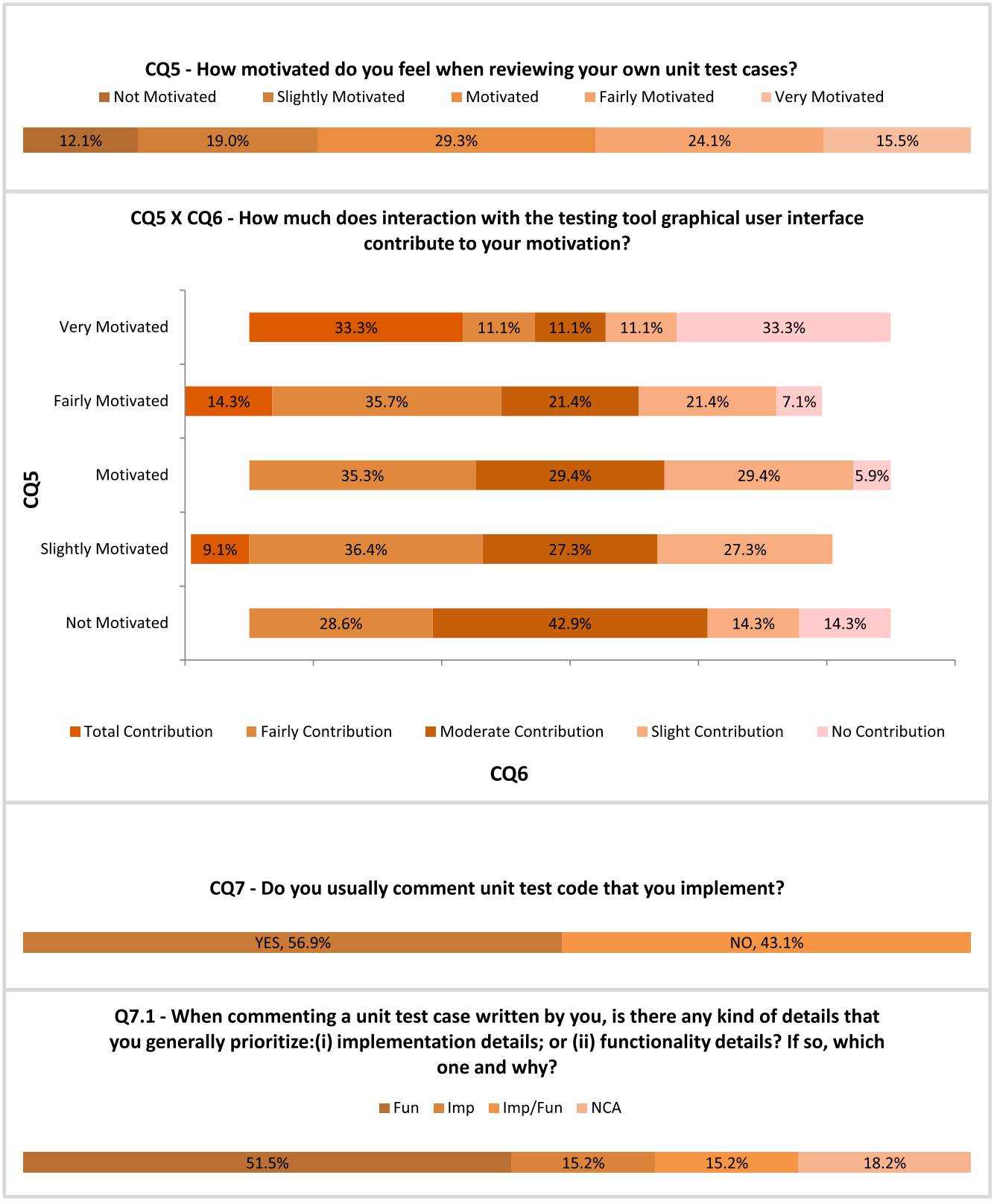


**Fig. 6.** Closed-ended questions and results (first part).

their comments. Since functionality aspects are related to domain elements and test case goals, we interpret this result as a willingness to write comments reflecting their situational model of the test case code.

The CQ9 results highlight how disturbing "flaky tests" are for our participants: 72.4% of them classified the problem as either "disturbing", "fairly disturbing" or "very disturbing". In the work of Luo et al. (2014), three features are commonly associated with the flaky tests: dependency between test cases; tests associated

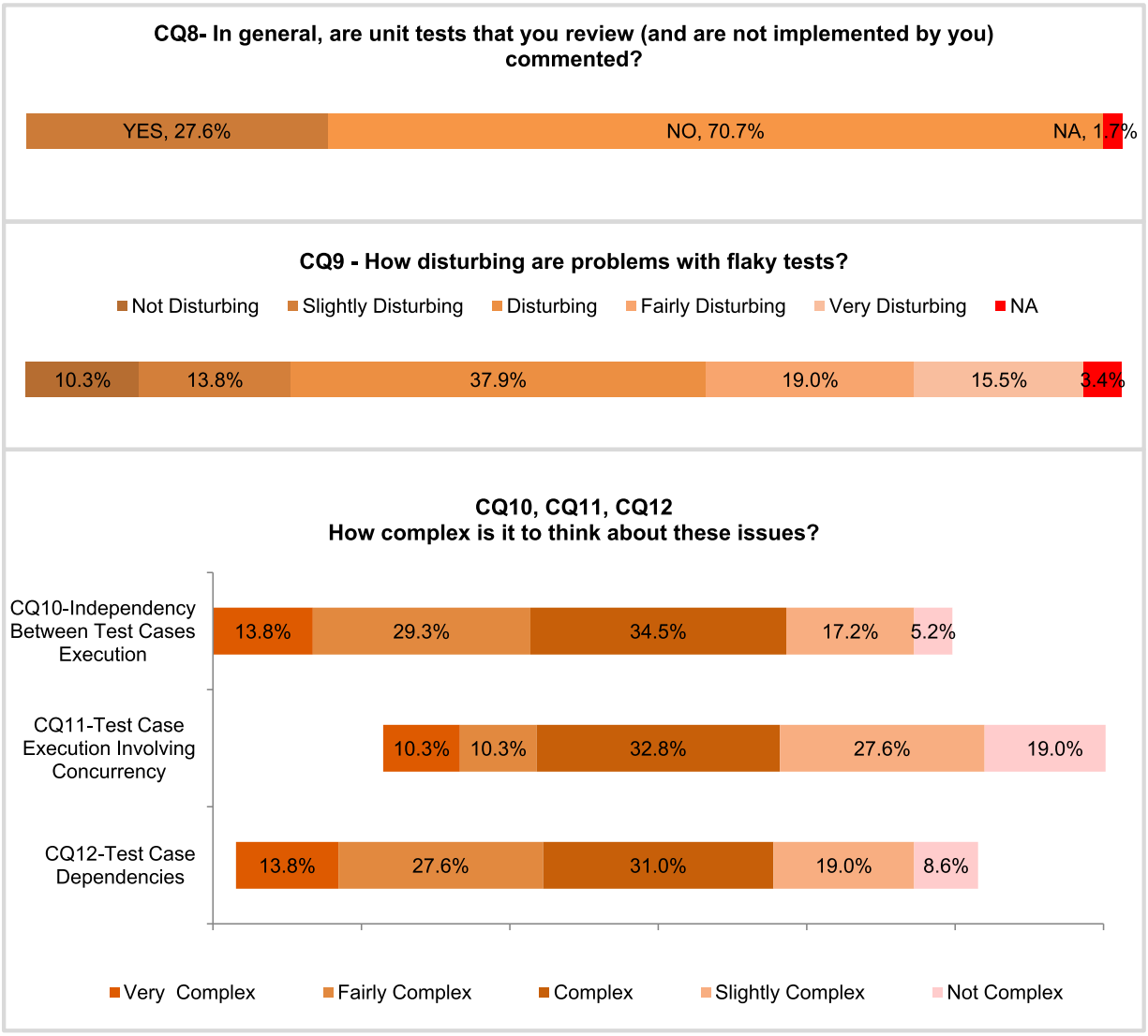


**Fig. 7.** Closed-ended questions and results (second part).

with concurrent execution; and test case dependencies (e.g. on an external service, file content, network resource, etc). Hence, we proposed three questions (CQ10, CQ11, CQ12) to evaluate the perception of these problems in unit testing review. In the Diverging Stacked Bar chart of the Fig. 7–CQ10, CQ11, CQ12 we can observe that the "Independence between test cases execution" and "test case dependencies" seem more complicated to them than "test case execution involving concurrency".

## 5. Framework of tasks requiring cognitive support

In this section, we present the framework of tasks requiring cognitive support. We made the framework description as specific as possible, based on the categories that emerged from our analysis.

The framework contains five groups: self-monitoring, test case inspection, drafting of unit testing information, flaky test discovery, and interaction with the tool's interface. Each group involves one or more tasks and they do not necessarily occur in isolation. For example, the drafting of unit test information may occur in parallel with the test case inspection tasks.

- *Self-monitoring*–support the user's self-management during the unit testing review tasks
  1. *Support the monitoring of to-do/executed tasks*–The practitioners plan and monitor their own progress during the unit testing review. This self-monitoring practice is heterogeneous and currently done through memorization, side notes, and reviewing of media like documents, diagrams, and text files. The self-monitoring problems are related to: (i) monitoring the responsibilities and the tasks that he/she accomplished; (ii) reviewing notes to remember the unit testing goal and to determine which scenario or execution flow remains to be covered; (iii) monitoring which changes were specified for the tests; (iv) reviewing documentation to remember the unit test purpose. Some remarkable coded segments regarding this issue were: *"I have to constantly keep in mind that unit tests exist to test MY work, and not some other 3rd party code or even language's features."; "I use to note in a text file what scenarios I need to do"; "I generally write down some notes about the inputs and outputs of the tested unit to help understanding the objectives of the unit"; "I always use notes to keep in memory and organize the purpose of test creation of truth tables"; "Check documentation and compare expected scenarios and results [to remember unit test purpose]"*
- *Test case inspection*–Support the understanding, organization and comparison of the tests

2. *Support the top-down and bottom-up inspection of the context/test case*–The practitioners inspect test cases alternating between two different directions of analysis: top-down and bottom-up. In the top-down inspection, understanding of test case context precedes understanding of the test case itself, which facilitates the interpretation of the test's behavior and goal. Some examples of this kind of analysis are: the review of the code being exercised by the test case; the reading of the software documentation and related diagrams; the recalling of test case or test suite properties i.e. naming convention, criterion that the test satisfies; the consultation of side notes, code comments or even other types of tests e.g. integration and system test cases. In the words of some of our participants:*"I try to review the code/functionality that is being tested (for contextualization), and then describe how the test is conducted (for ease of reading it)"; " test suite name of test context (Rspec case) is very import. Also, the test file name helps to understand the test purpose."*, or even *"I sometimes take a lot of time understanding the test scenario"*. During bottom-up inspection, the practitioners analyze a given test case to get an understanding of its purpose and context. Coded segments that exemplify this kind of analysis are: *"I have to refactor the test because it has to be clear enough to describe the software."; "I use a lot of JUnit inside Eclipse and I have experience with jasmine/mocha for javascript tests and the latter support contextualize the tests in a better way (with describe/it for example). It would be nice if JUnit could do the same."; "I generally take some notes on the problems that were found while executing the tests [to remember their purpose]"*
3. *Support the comparison between unit testing artifacts*–Testers need to draw on different sources of information to evaluate if the testing results are correct or not (oracle checking). These sources vary e.g., information that was directly given by final users, prescribed use case scenarios, test plans etc. Moreover, the professionals often compare two or more unit tests (or associated artifacts) during the refactoring of test case code. As indicated in some coded segments: *"I check documentation and compare expected scenarios and results"; "Sometimes I consult other test cases to see if I have forgotten some scenario that is recurrent in all projects; [I use] verification of assertions and evaluation of similarity between test cases to avoid redundancies."*.
4. *Facilitate the tracking of test post-execution information*–Visualizations of the tested unit and its coverage are important data-sources that practitioners seek out after test case execution. According to some of our volunteers: *"A link with the tested code should be more interactive and dynamic in order to inspect the code in case of error/failure; "the execution flow of the test case could be described graphically, with a graph or activity diagram [to facilitate interpretation]."*
5. *Assist the search and organization of the test cases according to properties of interest*–During the review, professionals need to pinpoint and rearrange the test cases according to properties of interest. In the participant's words: *"[... ] the IDEs are not very specific in indicating where the test classes are."* or *"[The] organization of the tests is very confusing. It would be great to organize them in a way that the developer/tester could define what to sort.group by classes, methods... giving 'tips' of similar tests"*.

- *Drafting of unit test information*–Support the drafting of relevant information for the unit test analysis
  6. *Support the handling of unit test related drafts*–The practitioners make annotations and sketches in the form of diagrams and drawings. These notes and sketches are created on paper or in software. These drafts work as a memory aid to support them in tasks like: reasoning about test case requirements and test case execution; understanding the purpose of the unit under test. Some coded segments that illustrate concern with these issues are: *"A graphical representation of inputs and outputs of a module as well as the own module, could be useful in the review of test cases"; "When performing unit tests, in functional aspect I use paper and pen to list equivalence partitions and decision graph. In structural aspect, I use paper and pen to mount a CFG to help"; Generally [I] write down some notes about the inputs and outputs of the tested unit to help understanding the objectives of the unit; "Drawings connecting boxes help organize reasoning and identify the inputs and outputs of the tests"*. These tasks share some similarity with those covered in the previous item "Support the monitoring of to-do/executed tasks". However, these diagrams and notes are oriented around the comprehension of the test behaviour rather than the self-management of the practitioners.
- *Flaky test discovery*–support the revealing of flaky tests
  7. *Support the application of strategies to reveal the flaky unit tests*–Practitioners mention the reuse of legacy code, inaccuracy of test case description, and overestimation of testing professionals during test creation as factors that contribute to flaky test occurrence. Moreover, practitioners adopt some strategies that assist discovery of flaky tests, such as: (i) verification and organization of test cases, according to properties such as time to execute each test and complexity of the test case; (ii) communication with users and team members; (iii) observation of alerts triggered by tested units or test case dependencies; (iv) noticing if the tested unit is executed under a complex scenario; (v) creation of another test case–e.g., a test case for the same unit in another scenario; (vi) observation of the failures' recurrence in a test case execution; (vii) observation of the number of test case's dependencies and their details; (viii) observation of discrepancies between the test case and its documentation (e.g. due to frequent changes, new scenarios, unclear description).
- *Interaction with the tool's interface* – support a better experience of the user interacting with the tools.
  8. *Simplify the testing results exhibited by the IDEs and improve tools' documentation*–Unit testing practitioners face troubles with interpretation of unit test results presented by IDEs. These difficulties vary from the high volume of data produced in the tests execution, to the data presentation. Moreover, we found problems associated with tool documentation. As expressed by some of our coded segments: *"When an error occurs, the IDEs could provide a consolidated information instead of so much detailed [information]. If the tester wants to detail more, it could be done in a second step."; "I like the JUnit plugin in Eclipse. However, plugins for Maven/Ant suck. You have to read all those crazy log files to understand something when it breaks."; "Sometimes the output can become difficult to read because of exception's stack traces printed as a log message"; "Some tools do not offer documentation detailed in an appropriated manner"*.

### 5.1. Discussion and trustworthiness considerations

The perspective adopted in this research was not—and could not be–sufficient to capture all the aspects of the unit testing phenomenon. Rather, as previously stated, we restricted our analysis to capture only cognitive support issues of the review practice, our primary interest in this research.

We tried to be as accurate as possible in describing tasks requiring cognitive support, based on the emerged categories and their included coded segments. Some framework items like "Sup-

port the top-down and bottom-up inspection of the context/test case", "Support the comparison between unit testing artifacts" and "Assist the search and organization of the test cases according to properties of interest" show the importance of retrieving and accessing multiple types of information during unit testing review. Other items like "Support the monitoring of to-do/executed tasks" or "Support the handling of unit test related drafts" exemplify how unit testing review practice relates to the particular way that each practitioner manages and reasons the activity. These tasks are related to *enrichment* processes under the perspective of Information Foraging Theory (Pirolli and Card, 1999) and correspond to activities that the practitioners execute for modeling the environment to fit available strategies (e.g., creating patches to help process information quickly (Burnett et al., 2009)). By supporting these tasks, tools could empower users to focus their attention on critical decisions of the testing review process–and shorten the path between reasoning about a problem and working on a solution.

Next, we consider the credibility, transferability, dependability and confirmability (Guba, 1981) of our research.

#### 5.1.1. Credibility

Our survey resulted from a refinement process (see Section 3.2.1), to facilitate understanding of the questions by participants–this enhanced our data gathering credibility. The process included: the pilot run of the survey; the progressive refinement of the questions with simplification of vocabulary; and the explanation of terminology before their use in the questionnaire to avoid misinterpretation (e.g. the definition of "flaky test"). Also, the volunteers' participation and our data use policy passed scrutiny of the research ethics committee of our institution. Our approved ethical commitments contained several terms that aimed to encourage our volunteers' honesty: the promise to make the volunteer's participation anonymous and confidential; the freedom of the volunteer to accept, reject and ignore involvement in the study; the assurance that we would safely store their responses under our responsibility; the guarantee that their results would be made anonymous previous to any possible publication; and the exclusive use of their data only for the stated purposes. Lastly, the QCA is a qualitative method that has been successfully applied in a wide range of scientific areas, especially in areas where the subjects' claims are the focus of attention–e.g. nursing and psychology. QCA is a flexible and concise method that allows different variants for category development depending on the study purpose. The choice of inductive category formation let us develop the categories directly out of the material. Such characteristics underlie the suitability of this method to our research goals, and contributed to the analysis credibility.

#### 5.1.2. Transferability

Regarding transferability, some observations should help to convey the boundaries of our study. As observed by Ritchie and Lewis (2003) "Inferential generalisation in social research must similarly be a matter of judgement, and the role of the researcher is to provide the 'thick description' of the researched context and the phenomena found (views, processes, experiences etc.) which will allow others to assess their transferability to another setting. As with theoretical generalization, the inference must rest as a hypothesis until proved or disproved by further evidence". Our volunteers came from a fairly diverse set of organizations, a total of 29 institutions, based on participants that opted to give this information. We did not restrict participation in our study based on any criteria other than having unit testing experience. To maximize the chances of obtaining insightful answers, we had to reach as many potential volunteers as we could. Fortunately, the invitation strategy was successful. From the total of 61 replies, 58 were considered valid–we had two blank replies and one that admitted the lack of unit testing experience in the responses. Moreover, the quality of the valid responses was enough to let the categories reach saturation during the coding process. Thus, we consider the number and quality of the obtained responses adequate for the goal and scope defined for our research. Furthermore, we strove to detail our framework description in a precise narrative form, based on the emerged categories and subcategories.

#### 5.1.3. Dependability and confirmability

We provided in-depth information about the processes and decisions taken in our study (Section 3), to enhance the dependability and confirmability of the results. This can be observed, for example, in Section 3.4.1 where we described the additional steps taken to mitigate the problem of overlapping concepts during our coding process. Also, since our participants came from different organizations, we had no control of their context. Thus, we proposed closed-ended questions–related to the problems discussed in the open-ended questions–to capture characteristics that could make our participants more comparable as a group (see Section 4). Hence, future studies wishing to replicate or triangulate with our study can reuse our closed-ended questions and results to select participants with a perspective closer to our volunteers' opinions. Our data gathering instrument also contributed to the improvement of the dependability of the findings. The online questionnaire is a neutral mechanism of inquiry. Therefore, it is less subject to unintended factors that may influence the participant's responses, e.g.: body language, voice intonation, or eye-contact of the interviewer. Another advantage of the online questionnaire is that the responses are textual, and response transcription is unnecessary. Further studies can make an immediate use of our questionnaire to make a more consistent comparison of the findings (Prado M.P. and Vincenzi A.M.R., 2017).

#### 5.1.4. Other considerations

Despite our concerns with trustworthiness, our work has limitations that leave room for improvement. For example, a triangulation study using other forms of inquiry (e.g. semi-structured interview), conducted by other researchers and considering another set of participants could contribute to the enhancement of credibility and confirmability. Additional volunteers and the opportunity to collect data directly from their workplace would help to make our findings even more representative and favor transferability. In a qualitative paradigm, it's important to recognize that the textual data needs interpretation, which can vary more than interpretation of measurable numbers, and researchers assume the unavoidable responsibility of interpreting responses during the coding process. That is the reason why we discuss generalization issues under "transferability" concerns instead of discussing "external validity". Thus, such trustworthiness considerations cannot be compared on the same basis as validity considerations in a quantitative paradigm.

Finally, we highlight that our results represent unprecedented contributions in the area, even with these limitations. It puts the human in the center of analysis, to research the evolution of unit testing tools. Our framework is a theoretical model that categorizes tasks, from the practitioner's routine, which should be supported by tools so as to leverage the practitioners' own skills. Moreover, the study size allows straightforward replication and extension. In this sense, it paves the way for new research concerned with the cognitive support problem in other software testing areas.

### 5.2. Research agenda

In this section we provide a research agenda based on the results of our study. The agenda serves as an actionable instrument

to guide the development of the framework findings into new cognitive support studies and applications.

We structured our agenda as follows: each subsection of the agenda corresponds to a different research theme i.e. further studies to be performed or applications of the framework findings. For each heading, we indicate the research theme title followed by the framework tasks (FTs) associated with the research theme.

*5.2.1. Research the specifics of using paper and pen versus software tools on the unit testing note-taking and sketching—{FTs: 1 and 6}*

Note-taking can help people handle the cognitive load and amplify their innate cognitive abilities (Dror and Harnad, 2008; Makany et al., 2009). According to Kiewra (1987) "note-taking encourages organization, and organization during note-taking increases achievement". However, there is still much debate on the benefits and drawbacks of note-taking using pen and paper versus software tools, regarding their effect on problem-solving and learning activities (Friedman, 2014). The educational field has been the natural environment for such investigations. However, problem-solving and learning are also part of software testing activity. As shown in our results, the unit testing practitioners use both types of devices in their note-taking and sketching to manage their tasks and solve unit testing review problems. In this sense, further research should be conducted to assess the intrinsic advantages and disadvantages of using both types of tools for the unit testing practitioners. This research could include experimental studies (e.g. Oviatt et al. (2012, 2007, 2006)) designed to compare how both types of tools, as well as hybrid technologies (e.g. pen-enabled devices), facilitate or hinder common unit testing review problems, including but not limited to: reasoning about unit testing requirements; delimitation of responsibilities; monitoring of accomplished tasks and flows to be tested; modeling of unit and test case behavior; and understanding test case goals.

*5.2.2. Development of a new apparatus to support unit testing sketching and note-taking—{FTs: 1 and 6}*

Tool designers could propose a new supporting apparatus to leverage the current toolset capabilities, based on results coming from previous research theme (5.2.1). Also, as observed in the coded segments, sketches and notes from unit testing review often refer to people and development artifacts– teammates, test cases, use cases etc. In this sense, a tool could, for example, enhance support by enabling detection and linkage of notes and sketches to artifacts and stakeholders that were registered in the annotated information–e.g. binding the notes to the corresponding users or files at Github, or to a calendar reminder. However, this and new ideas coming from further studies should be co-validated with users. Thus, researchers would need to choose practitioners willing to give continuous feedback on this and other interaction design issues–as shown in Muto et al. (2011). That includes but is not limited to: deciding whether the intended apparatus should be integrated into or separated from existing IDEs; determining the preferred ways to retrieve and organize notes and sketches; choosing appropriate metaphors to use in the interface; defining input and output methods. Also, it would be important that users could be sampled from diverse backgrounds–i.e. users with varying levels of unit testing expertise; with different socio-demographic profiles; experienced in different languages/technologies–since personal factors can influence interaction design decisions (Chalmers, 2003; Burnett et al., 2009). In case the implementation of such an apparatus involves input/output technologies that developers are not yet using in their work environment (e.g. a digitizer), technological probes could offer some help. A technological probe is an instrument used to explore the unknown about new technology. It promotes users' criticism and gives the users the freedom to adapt the proposed technology in creative new ways. Moreover, it is appropriate for complex and private settings where learning about the users' attitude towards technology can be challenging (Hutchinson et al., 2003).

*5.2.3. Research of supporting mechanisms that mitigate cognitive overload risk during unit testing bottom-up and top-down analysis—{FTs: 2 and 3}*

According to Cognitive Load Theory, impediments, distractions, and unnecessary information faced during the learning process increase extraneous load in the working memory and contribute to cognitive overload (Sweller, 1988; 1999; van Merriënboer and Sweller, 2005). Considering that learning processes are involved in understanding of units and their test cases, erratic navigation of unit test information sources can hamper bottom-up and top-down inspection activities. An example of this is having to rummage through diverse artifacts to find information, or digging through multiple redundant descriptions in order to learn about a tested unit and its dependencies. Thus, we propose supporting mechanisms aimed at facilitating the association and access to unit testing contextual information, and recommend research to test how well those mechanisms mitigate such problems. For example, would immediate retrieval of specific documentation and diagrams associated with a given unit test case, without losing location in the test case code view, reduce cognitive overload? What are the positive and negative effects in the practitioner's comprehension of tested units and their test cases? The works of Paas et al. (2003); Chen et al. (2013) and Oviatt et al. (2006) provide useful information for the design of an experimental setup to assess cognitive load issues.

*5.2.4. Design of contextual information retrieval mechanisms from and for unit tests—{FTs: 2 and 3}*

Developing any cognitive support mechanisms, as proposed in the previous research theme 5.2.3, requires some initial research. For example, how do practitioners usually combine the artifact's information during a top-down analysis? What tools and techniques are involved in this process? These and other technical aspects may vary between development communities. However, these are necessary elements for an effective interaction design proposal, and therefore new user studies are necessary. As a suggestion, we think that a three-stage user study composed of a focus group, followed by a field study, then contextual interviews, could help to address this need. In the first stage, the researchers should try to understand the common artifacts, technologies and associated methods adopted by the users from a given community–e.g. the unit testing techniques and strategies used by android app developers. Moreover, the first stage would serve to strengthen ties with participants and to prepare for subsequent stages. In the field study, researchers should observe how users perform bottom-up and top-down analysis in their natural setting, under the conditions discovered in the previous stage. Based on these observations, the researchers should compile the common challenges faced by the users as they perform both top-down and bottom-up analysis. The contextual interview will help solve questions that emerge during the field study and to explore different ideas of tool support. For the design process, researchers could apply rapid prototyping to mature the support requirements with participants.

*5.2.5. Research and development of co-adaptative tools to assist the search, comparison, organization and tracking of unit tests and their results–FTs: 3, 4, 5, 7 and 8*

Practitioners regularly need to locate, compare and rearrange test cases and tested units in different orders during unit testing review. Also, they alternate heavily between viewing test case and the corresponding results. Finding a pattern to support these

tasks is difficult because of the different practitioners' contexts–development toolset, adopted methodologies, processes and testing strategies (e.g., test driven development *versus* traditional *versus* hybrid (Rojas et al., 2015; Erdogmus et al., 2005)). Moreover, each individual may face these problems differently. Thus, any supporting mechanism for these tasks should be customizable and learn from the user's particular workflow. In this sense, the user can smoothly adapt to the new support and *vice-versa*. Machine learning can contribute to this co-adaptation. For instance: (i) based on the test cases and tested units properties that the user commonly accesses, the tool could suggest how to sort or compare the test cases–e.g., test cases associated with the same dependencies, exercising the same path in the code or revealing the same kind of errors; (ii) the tool could suggest unit tests that are likely to be refactored if it learns that extensive access and modification on tested unit dependencies leads to modification on the associated unit tests; (iii) the tool could suggest deprecated unit tests based on criteria that can be correlated like the low frequency of user's view and the unit test being included by a more effective test–e.g., a unit test that covers the same path and reveals more errors. The work of Oviatt (2006) discusses several user-centered design principles associated with improved human performance. The work of Mackay (1990) provides the theoretical foundation to understand the co-adaptive phenomenon between users and customizable software. Mathur et al. (2015) demonstrate how some adaptive support mechanisms and machine learning methods can be used e.g., to facilitate the execution of user interface driven testing.

*5.2.6. New studies to support flaky test discovery strategies—{FT: 7}*

Several strategies that practitioners use to help reveal flaky tests emerged in our study. Although some of the previous research themes support those strategies–e.g., theme 5.2.5 "Research and development of co-adaptive tools (...)" for the strategy "(i) the verification and organization of the test cases according to properties such as the time to execute each test and the complexity of the test case;"– further research is necessary to elucidate other forms of support. For example, considering that *the communication with the users and the team members* is an emerged strategy in our study for revealing flaky tests, one could investigate how collaboration can offer leverage. Such studies could inform improvements to collaborative tools and communication channels, in order to ease flaky test discovery–not only for unit testing, but also for the other testing levels. Research in this sense is very timely. Companies like Google, for example, have recently demonstrated their interest in the problem by designating a team dedicated to informing developers of the harms caused by test flakiness (Micco, 2016).

*5.2.7. Re-shape the testing results presentation—{FT: 8}*

The way that the IDEs currently present unit testing results was a major complaint related to usability that emerged in our study. The tool designers should support testers with a better experience here, which includes but is not limited to: the reformulation and presentation of test execution information into more consolidated forms; the option to explore further execution details; and the ability to subdivide this exploration into successive steps. Some methods that researchers could apply to address these problems: (i) moderated usability tests (Vasalou et al., 2004; Barnum, 2010) and eye-tracking (Pernice and Nielsen, 2009) studies could help understand and find patterns of user interaction with the unit testing toolset e.g.: to identify what draws attention in the GUI, and associated causal factors (to determine what erroneously catches user's attention, so as to optimize recognition of the desired testing result information); (ii) think aloud technique could help discover how users inspect unit testing execution logs–and whether the information found coincides with their expectations; (iii) card sorting technique can be used to rearrange the log information structure into an order that makes sense to users and meets their needs.

## 6. Conclusions

This paper is the third in a series that investigates cognitive support problem in unit testing activities.

Our first paper, Prado et al. (2015) overcame some challenges: we defined a new research problem in the testing area; we highlighted how the testing community was neglecting human aspects in tool research; the new problem required a different research perspective than was commonly used in testing tool research. To support our claims, we discussed works in the literature that exemplified the disconnection between human aspects and testing solutions. Also, we proposed an initial framework composed of three dimensions of cognitive demands for unit testing. The framework was based on problems of practice reported in the literature.

The second paper, Prado and Vincenzi (2016) opened the research problem in the domain of visualization. We wanted to know how unit testing tools that used visualization resources were addressing cognitive support problems. We applied systematic mapping to select works of interest. Then, we examined each selected work under the cognitive dimensions proposed in our initial framework. The findings indicated several cognitive support gaps and improvement opportunities. Moreover, we found that the lack of user involvement in research was a common factor among all the works with cognitive support problems.

Motivated by the results from our previous publications, we decided to perform our current study of practitioners with unit testing experience. We developed a qualitative survey with 24 questions to understand how tools could improve cognitive support to users during unit testing review. A total of 61 volunteers replied our survey–58 were considered valid. Then, we applied the Qualitative Content Analysis method–in its inductive form–to elicit the categories of cognitive support problems that are common to the volunteers. Based on the categories that emerged in the analysis, we proposed a framework of tasks requiring cognitive support. The analysis process was discussed in-depth, from survey planning through to coding of responses. Subsequently, we discussed the trustworthiness considerations of our work. In addition, we proposed a research agenda based on the framework, as an actionable instrument for the testing community.

Altogether, our framework and our research agenda offer a valuable set of information for evolution of current and future unit testing tools–be it in the form of standalone testing tools, IDE plugins, or adaptations to the testing environment. The framework takes into consideration real problems that affect real-world practitioners. Researchers, third-party or in-house tool developers can use our contributions as guidelines to address user needs that are neglected or poorly supported by current tools. These improvements have the potential to impact unit testing activity and, ultimately, the final quality of the software under test. As highlighted in themes of our research agenda, any proposal aiming to improve cognitive support must involve co-participation of real-world users. Such involvement is necessary to ensure an user-centered approach and a satisfactory cognitive support improvement process.

Our study provides original contributions for understanding and addressing a largely neglected problem in software testing research. Besides the themes proposed in our research agenda, we envision that future studies will address cognitive support for other unit testing practices and test levels, such as the creative process, during unit test case planning and implementation; integration and system testing; performance testing practices, etc. For example, one of our participants mentioned in one of the re-

sponses: “When there is already some similar test [case], I use it as a basis to do the new one. But when it’s a totally unusual test, I start working from scratch”. In case this is a common approach among practitioners, one could investigate the clues that testing professionals use to determine the existence of similar tests during test creation. Do they have to read, interpret and mentally compare each test from the suite with the one that they are considering to implement/generate? How do the tools currently support the tester in this process? How can the tools be improved (from the practitioners’ perspective) to help with the test implementation/generation process?

Finally, new studies wishing to replicate or extend our work are welcome. They would help to reinforce the importance of involving the human in testing tool research. As a result, they would contribute to making software testing a more pleasant and efficient activity for the primary stakeholders–those that battle for software quality on the front line.

## Acknowledgments

The authors would like to thank the Brazilian funding agency: FAPEG. The first author is a Ph.D. student and received a scholarship from FAPEG (grant no. 2016/10267000693). We would like to thank each volunteer that chose to participate in our study for their inestimable contribution to our results. Special thanks to Dr. Margaret-Anne Storey for her precious contributions to our research since its initial stages, and to Eric Verbeek for his invaluable friendship and feedback on our paper’s written presentation.

## References

Adolph, S., Hall, W., Kruchten, P., 2011. Using grounded theory to study the experience of software development. Emp. Softw. Eng. 16 (4), 487–513. doi:10.1007/s10664-010-9152-6.

Alemerien, K., Magel, K., 2014. Examining the effectiveness of testing coverage tools: an empirical study. Int. J. Softw. Eng. Appl. 8 (5), 139–162.

Atkinson, C., Barth, F., Brenner, D., 2010. Software testing using test sheets. In: Proceedings of the Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW), pp. 454–459. doi:10.1109/ICSTW.2010.21.

Barnum, C.M., 2010. Usability Testing Essentials: Ready, Set...Test!. Elsevier.

Beck, K., 2004. JUnit Pocket Guide, first ed. O'Reilly Media, Beijing ; Sebastopol, Calif.

Binder, R.V., 1999. Testing Object-Oriented Systems: Models, Patterns, and Tools, 1. Addison Wesley Longman, Inc.

Brod, M., Tesler, L.E., Christensen, T.L., 2009. Qualitative research and content validity: developing best practices based on science and experience. Qual. Life Res. Int. J. Qual. Life Aspects Treatm. Care Rehabil. 18 (9), 1263–1278. doi:10.1007/s11136-009-9540-9.

Burnett, M., Bogart, C., Cao, J., Grigoreanu, V., Kulesza, T., Lawrance, J., 2009. End-user software engineering and distributed cognition. In: Proceedings of the ICSE Workshop on Software Engineering Foundations for End User Programming, pp. 1–7. doi:10.1109/SEEUP.2009.5071696.

Card, S.K., Mackinlay, J.D., Shneiderman, B., 1994. Morgan Kaufmann Publishers, pp. 1–35.

Ceccato, M., Marchetto, A., Mariani, L., Nguyen, C.D., Tonella, P., 2015. Do automatically generated test cases make debugging easier? an experimental assessment of debugging effectiveness and efficiency. ACM Trans. Softw. Eng. Methodol. 25 (1), 5:1–5:38. doi:10.1145/2768829.

Chalmers, P.A., 2003. The role of cognitive theory in human–computer interface. Comput. Hum. Behav. 19 (5), 593–607.

Chen, F., Ruiz, N., Choi, E., Epps, J., Khawaja, M.A., Taib, R., Yin, B., Wang, Y., 2013. Multimodal behavior and interaction as indicators of cognitive load. ACM Trans. Interact. Intell. Syst. 2 (4), 22:1–22:36. doi:10.1145/2395123.2395127.

Colanzi, T.E., 1999. Uma abordagem integrada de desenvolvimento e teste de sofware baseada na UML. ICMC-USP, São Carlos, SP Master's thesis.

Corbin, J., Strauss, A., 2014. SAGE Publications, Inc, pp. 31–56.

Cottam, J.A., Hursey, J., Lumsdaine, A., 2008. Representing unit test data for large scale software development. In: Proceedings of the Fourth ACM Symposium on Software Visualization. ACM, New York, NY, USA, pp. 57–66. doi:10.1145/1409720.1409730.

Creswell, J.W., 2007, second ed.. Sage Publications, Inc, p. 40.

Creswell, J.W., 2013. Research design: Qualitative, quantitative, and mixed methods approaches, fourth ed SAGE Publications, Inc, Thousand Oaks.

Creswell, J.W., 2013, fourth ed, 4, pp. 129–143.

Creswell, J.W., 2013, fourth ed, 1, pp. 25–50.

Crowley, B.P., Delfico, J.F., 2013. Content Analysis: A Methodology for Structuring and Analyzing Written Material: PEMD-10.3.1. BiblioGov.

Daka, E., Fraser, G., 2014. A survey on unit testing practices and problems. In: Proceedings of the IEEE Twenty-Fifth International Symposium on Software Reliability Engineering (ISSRE), pp. 201–211. doi:10.1109/ISSRE.2014.11.

Delahaye, M., du Bousquet, L., 2015. Selecting a software engineering tool: lessons learnt from mutation analysis. Softw. Pract. Exp. doi:10.1002/spe.2312.

Detienne, F., 2001. Software Design – Cognitive Aspect. Springer, London ; New York. Softcover reprint of the original 1st ed. 2002 edition

Dodou, D., de Winter, J.C.F., 2014. Social desirability is the same in offline, online, and paper surveys: a meta-analysis. Comput. Hum. Behav. 36, 487–495. doi:10.1016/j.chb.2014.04.005.

Dror, I.E., Harnad, S., 2008. Offloading Cognition onto Cognitive Technology. In: Dror, I.E., Harnad, S. (Eds.), Benjamins Current Topics, 16. John Benjamins Publishing Company, Amsterdam, pp. 1–23. doi:10.1075/bct.16.02dro.

Eloussi, L., 2015. Determining flaky tests from test failures. University of Illinois at Urbana-Champaign Ph.D. thesis. http://hdl.handle.net/2142/78543.

Erdogmus, H., Morisio, M., Torchiano, M., 2005. On the effectiveness of the test-first approach to programming. IEEE Trans. Softw. Eng. 31 (3), 226–237. doi:10.1109/TSE.2005.37.

Flick, U. (Ed.), 2013. The SAGE Handbook of Qualitative Data Analysis, 1, 1 edition. SAGE Publications Ltd, Los Angeles.

Flick, U., 2014. SAGE Publications Ltd, pp. 66–69.

Frankl, P.G., Weiss, S.N., 1991. An experimental comparison of the effectiveness of the all-uses and all-edges adequacy criteria. In: Proceedings of the Symposium on Testing, Analysis, and Verification. ACM, New York, NY, USA, pp. 154–164. doi:10.1145/120807.120821.

Fraser, G., Staats, M., McMinn, P., Arcuri, A., Padberg, F., 2013. Does automated white-box test generation really help software testers? In: Proceedings of the 2013 International Symposium on Software Testing and Analysis. ACM, New York, NY, USA, pp. 291–301. doi:10.1145/2483760.2483774.

Fraser, G., Staats, M., McMinn, P., Arcuri, A., Padberg, F., 2015. Does automated unit test generation really help software testers? A controlled empirical study. ACM Trans. Softw. Eng. Methodol. 24 (4), 23:1–23:49. doi:10.1145/2699688.

Friedman, M.C., 2014. Harvard University.

Galler, S.J., Aichernig, B.K., 2014. Survey on test data generation tools. Int. J. Softw. Tools Technol. Trans. 16 (6), 727–751. doi:10.1007/s10009-013-0272-3.

Glaser, B.G., Strauss, A.L., 1999. The Discovery of Grounded Theory: Strategies for Qualitative Research. Aldine, New Brunswick u.a.

Guba, E.G., 1981. Criteria for assessing the trustworthiness of naturalistic inquiries. ECTJ 29 (2), 75. doi:10.1007/BF02766777.

Gyori, A., Shi, A., Hariri, F., Marinov, D., 2015. Reliable testing: detecting state-polluting tests to prevent test dependency. In: Proceedings of the International Symposium on Software Testing and Analysis. ACM, New York, NY, USA, pp. 223–233. doi:10.1145/2771783.2771793.

Henderson, D.L., Carter, S., 2009. Qualitative Research Design. Sage Publications Ltd, London.

Hollan, J., Hutchins, E., Kirsh, D., 2000. Distributed cognition: toward a new foundation for human-computer interaction research. ACM Trans. Comput. Hum. Inter. (TOCHI) 7 (2), 174–196.

Hsieh, H.-F., Shannon, S.E., 2005. Three approaches to qualitative content analysis. Qual. Health Res. 15 (9), 1277–1288. doi:10.1177/1049732305276687.

Hutchinson, H., Mackay, W., Westerlund, B., Bederson, B.B., Druin, A., Plaisant, C., Beaudouin-Lafon, M., Conversy, S., Evans, H., Hansen, H., Roussel, N., Eiderbäck, B., 2003. Technology probes: inspiring design for and with families. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, New York, NY, USA, pp. 17–24. doi:10.1145/642611.642616.

Jahangirova, G., Clark, D., Harman, M., Tonella, P., 2016. Test oracle assessment and improvement. In: Proceedings of the Twenty-Fifth International Symposium on Software Testing and Analysis. ACM, New York, NY, USA, pp. 247–258. doi:10.1145/2931037.2931062.

Jia, Y., Harman, M., 2011. An analysis and survey of the development of mutation testing. IEEE Trans. Softw. Eng. 37 (5), 649–678. doi:10.1109/TSE.2010.62.

Jr, R.M.G., Unrau, Y.A., 2010. Oxford University Press, p. 57. Google-Books-ID: sYC_WI_COd8C

Karam, M.R., Abdallah, A.A., 2005. Assisting in fault localization using visual programming constructs. In: Proceeding of the Canadian Conference on Electrical and Computer Engineering, pp. 856–860. doi:10.1109/CCECE.2005.1557114.

Kasurinen, J., Taipale, O., Smolander, K., 2009. Analysis of problems in testing practices. In: Proceedings of the Asia-Pacific Software Engineering Conference, 2009. APSEC '09, pp. 309–315. doi:10.1109/APSEC.2009.17.

Kasurinen, J., Taipale, O., Smolander, K., 2010. Software test automation in practice: empirical observations. Adv. Soft. Eng. 2010, 4:1–4:13. doi:10.1155/2010/620836.

Kiewra, K.A., 1987. Notetaking and review: the research and its implications. Instr. Sci. 16 (3), 233–249. doi:10.1007/BF00120252.

Knowles, E.S., Nathan, K.T., 1997. Acquiescent responding in self-reports: cognitive style or social concern? J. Res. Pers. 31 (2), 293–301. doi:10.1006/jrpe.1997.2180.

Lappalainen, V., Itkonen, J., Isomöttönen, V., Kollanus, S., 2010. ComTest: a tool to impart TDD and unit testing to introductory level programming. In: Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education. ACM, New York, NY, USA, pp. 63–67. doi:10.1145/1822090.1822110.

Lawrence, J., Clarke, S., Burnett, M., Rothermel, G., 2005. How well do professional developers test with code coverage visualizations? An empirical study. In: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), pp. 53–60. doi:10.1109/VLHCC.2005.44.

Lee, J., Kang, S., Lee, D., 2012. Survey on software testing practices. IET Softw. 6 (3), 275–282. doi:10.1049/iet-sen.2011.0066.

Leitner, A., Ciupa, I., Meyer, B., Howard, M., 2007. Reconciling manual and automated testing: the autotest experience. In: Proceedings of the Fortieth Annual Hawaii International Conference on System Sciences, HICSS 2007, p. 261a. doi:10.1109/HICSS.2007.462.

Luo, Q., Hariri, F., Eloussi, L., Marinov, D., 2014. An empirical analysis of Flaky tests. In: Proceedings of the Twenty-Second ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, New York, NY, USA, pp. 643–653. doi:10.1145/2635868.2635920.

Mackay, W.E., 1990. Users and customizable software : a co-adaptive phenomenon. Massachusetts Institute of Technology Thesis.

Madeyski, L., Orzeszyna, W., Torkar, R., Józala, M., 2014. Overcoming the equivalent mutant problem: a systematic literature review and a comparative experiment of second order mutation. IEEE Trans. Softw. Eng. 40 (1), 23–42. doi:10.1109/TSE.2013.44.

Makany, T., Kemp, J., Dror, I.E., 2009. Optimising the use of note-taking as an external cognitive aid for increasing learning. Br. J. Educ. Technol. 40 (4), 619–635.

Mathur, R., Miles, S., Du, M., 2015. Adaptive automation: leveraging machine learning to support uninterrupted automated testing of software applications. CoRR abs/1508.00671. arXiv:1508.00671. [Online] http://arxiv.org/abs/1508.00671.

Mayring, P., 2014. psychopen.eu, p. 143.

Melnik, G., Read, K., Maurer, F., 2004. Suitability of FIT user acceptance tests for specifying functional requirements: developer perspective. In: Zannier, C., Erdogmus, H., Lindstrom, L. (Eds.), Extreme Programming and Agile Methods - XP/Agile Universe 2004: Fourth Conference on Extreme Programming and Agile Methods, Calgary, Canada, August 15–18, 2004. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 60–72. doi:10.1007/978-3-540-27777-4_7.

Merriam, S.B., 2002, first ed. Jossey-Bass, San Francisco, p. 5.

van Merriënboer, J.J.G., Sweller, J., 2005. Cognitive load theory and complex learning: recent developments and future directions. Educ. Psychol. Rev. 17 (2), 147–177. doi:10.1007/s10648-005-3951-0.

Micco, J. Flaky tests at Google and how we mitigate them, 2016, [Online] https://testing.googleblog.com/2016/05/flaky-tests-at-google-and-how-we.html

Miles, M.B., Huberman, A.M., Saldaña, J., 2013, third ed. SAGE Publications, Inc, Thousand Oaks, Californinia, pp. 41–43.

Morse, J.M., 1991. Approaches to qualitative-quantitative methodological triangulation. Nurs. Res. 40 (2). http://journals.lww.com/nursingresearchonline/Fulltext/1991/03000/Approaches_to_Qualitative_Quantitative.14.aspx.

Muto, Y., Okano, K., Kusumoto, S., 2011. Improvement of a visualization technique for the passage rate of unit testing and static checking and its evaluation. In: Proceedings of the Joint Conference of the Twenty-first International Workshop on Software Measurement and the Sixth International Conference on Software Process and Product Measurement (IWSM-MENSURA), pp. 279–284. doi:10.1109/IWSM-MENSURA.2011.27.

Naik, K., Tripathy, P., 2008. Software Testing and Quality Assurance: Theory and Practice, first ed. Wiley-Blackwell, Hoboken, N.J.

Ng, S., Murnane, T., Reed, K., Grant, D., Chen, T., 2004. A preliminary survey on software testing practices in Australia. In: Proceedings of the Australian Software Engineering Conference, pp. 116–125. doi:10.1109/ASWEC.2004.1290464.

Orso, A., Rothermel, G., 2014. Software testing: a research travelogue. In: Proceedings of the on Future of Software Engineering. ACM, New York, NY, USA, pp. 117–132. doi:10.1145/2593882.2593885.

Oviatt, S., 2006. Human-centered design meets cognitive load theory: designing interfaces that help people think. In: Proceedings of the Fourteenth ACM International Conference on Multimedia. ACM, New York, NY, USA, pp. 871–880. doi:10.1145/1180639.1180831.

Oviatt, S., Arthur, A., Brock, Y., Cohen, J., 2007. Expressive pen-based interfaces for math education. In: Proceedings of the Eighth International Conference on Computer Supported Collaborative Learning. International Society of the Learning Sciences, pp. 573–582. http://dl.acm.org/citation.cfm?id=1599600.1599708.

Oviatt, S., Arthur, A., Cohen, J., 2006. Quiet interfaces that help students think. In: Proceedings of the Nineteenth Annual ACM Symposium on User Interface Software and Technology. ACM, New York, NY, USA, pp. 191–200. doi:10.1145/1166253.1166284.

Oviatt, S., Cohen, A., Miller, A., Hodge, K., Mann, A., 2012. The impact of interface affordances on human ideation, problem solving, and inferential reasoning. ACM Trans. Comput. Hum. Interact. 19 (3), 22:1–22:30. doi:10.1145/2362364.2362370.

Paas, F., Tuovinen, J.E., Tabbers, H., Van Gerven, P.W., 2003. Cognitive load measurement as a means to advance cognitive load theory. Educ. Psychol. 38 (1), 63–71.

Panichella, S., Panichella, A., Beller, M., Zaidman, A., Gall, H.C., 2016. The impact of test case summaries on bug fixing performance: An empirical investigation. In: Proceedings of the Thirt-Eighth International Conference on Software Engineering. ACM, New York, NY, USA, pp. 547–558. doi:10.1145/2884781.2884847.

Pernice, K., Nielsen, J., How to conduct eyetracking Studies, Nielsen Norman Group Report, 2009, Nielsen Norman Group https://www.nngroup.com/reports/how-to-conduct-eyetracking-studies/.

Perry, D.E., Kaiser, G.E., 1990. Adequate testing and object-oriented programming. J. Object Oriented Program. 2 (5), 13–19.

Pirolli, P., Card, S., 1999. Information foraging. Psychol. Rev. 106 (4), 643–675. doi:10.1037/0033-295X.106.4.643.

Prado, M.P., Resources_tcsut, 2017, https://app.box.com/s/sk52eu5akh1f0o8dj0qkh4pvp5r2p6lk.

Prado, M.P., Verbeek, E., Storey, M.A., Vincenzi, A.M.R., 2015. WAP: cognitive aspects in unit testing: the hunting game and the Hunter's perspective. In: proceedings of the IEEE Twenty-Sixth International Symposium on Software Reliability Engineering (ISSRE), pp. 387–392. doi:10.1109/ISSRE.2015.7381832.

Prado, M.P., Vincenzi, A.M.R., 2016. Advances in the characterization of cognitive support for unit testing: the bug-hunting game and the visualization arsenal. In: Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 213–220. doi:10.1109/ISSREW.2016.11.

Prado M. P. and Vincenzi A.M.R., Link to the survey applied with the volunteers. 2017, [Online] http://survey.sogosurvey.com/k/RQsTTWSQsRTsPsPsP. (Accessed: 11-March-2017).

Ramler, R., Buchgeher, G., Klammer, C., 2018. Adapting automated test generation to GUI testing of industry applications. Inf. Softw. Technol. 93, 248–263. doi:10.1016/j.infsof.2017.07.005.

Ricca, F., Torchiano, M., Ceccato, M., Tonella, P., 2007. Talking tests: an empirical assessment of the role of fit acceptance tests in clarifying requirements. In: Proceedings of the Ninth International Workshop on Principles of Software Evolution: In Conjunction with the Sixth ESEC/FSE Joint Meeting. ACM, New York, NY, USA, pp. 51–58. doi:10.1145/1294948.1294962.

Ritchie, J., Lewis, J., 2003, first ed. SAGE Publications Ltd, Los Angeles, Calif., p. 264.

Rojas, J.M., Fraser, G., Arcuri, A., 2015. Automated unit test generation during software development: a controlled experiment and think-aloud observations. In: Proceedings of the International Symposium on Software Testing and Analysis. ACM, New York, NY, USA, pp. 338–349. doi:10.1145/2771783.2771801.

Runeson, P., 2006. A survey of unit testing practices. IEEE Softw. 23 (4), 22–29. doi:10.1109/MS.2006.91.

Schreier, M., 2013, first ed. The SAGE Handbook of Qualitative Data Analysis, Ch. Qualitative Content Analysis, 1, p. 175.

Sommerville, I., 2010, Software Engineering, ninth ed. Pearson, Boston. Ch. Software testing, p. 213.

Stol, K.-J., Ralph, P., Fitzgerald, B., 2016. Grounded theory in software engineering research: a critical review and guidelines. In: Proceedings of the Thirty-Eighth International Conference on Software Engineering. ACM, New York, NY, USA, pp. 120–131. doi:10.1145/2884781.2884833.

Sweller, J., 1988. Cognitive load during problem solving: effects on learning. Cogn. Sci. 12 (2), 257–285. doi:10.1207/s15516709cog1202_4.

Sweller, J., 1999. Instructional Design in Technical Areas. ACER Press, Camberwell, Vic.

Vasalou, A., Ng, B.D., Wiemer-Hastings, P., Oshlyansky, L., 2004. Human-moderated remote user testing: protocols and applications. In: Proceedings of the Eighth ERCIM Workshop, User Interfaces for All, Wien, Austria, pp. 1–10.

Vincenzi, A.M.R., 2004. Orientação a Objeto: Definição e Análise de Recursos de Teste e Validação. ICMC/USP, São Carlos, SP Ph.D. thesis.

Walenstein, A., 2002. Cognitive Support in Software Engineering Tools: A Distributed Cognition Framework. School of Computing Science, Simon Fraser University Ph.D. thesis.

Wappler, S., Wegener, J., 2006. Evolutionary unit testing of object-oriented software using strongly-typed genetic programming. In: Proceedings of the Eighth Annual Conference on Genetic and Evolutionary Computation. ACM, New York, NY, USA, pp. 1925–1932. doi:10.1145/1143997.1144317.

Weber, R.P., 1990, first ed. SAGE Publications, Inc, Newbury Park, Calif, p. 37.

Wiklund, K., Sundmark, D., Eldh, S., Lundvist, K., 2014. Impediments for automated testing – an empirical analysis of a user support discussion board. In: Proceedings of the IEEE Seventh International Conference on Software Testing, Verification and Validation (ICST), pp. 113–122. doi:10.1109/ICST.2014.24.

Xie, T., 2017. Transferring software testing tools to practice. In: Proceedings of the IEEE/ACM Twelfth International Workshop on Automation of Software Testing (AST), p. 8. doi:10.1109/AST.2017.10.

Yang, Q., Li, J.J., Weiss, D.M., 2009. A survey of coverage-based testing tools. Comput. J. 52 (5), 589–597. doi:10.1093/comjnl/bxm021.

Zeng, X., Li, D., Zheng, W., Xia, F., Deng, Y., Lam, W., Yang, W., Xie, T., 2016. Automated test input generation for android: are we really there yet in an industrial case? In: Proceedings of the Twentieth ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, New York, NY, USA, pp. 987–992. doi:10.1145/2950290.2983958.

**Marllos P. Prado** is Associate Professor at Federal Institute of Goiás, Brazil and Ph.D. candidate at Federal University of Goiás, Brazil. He received his M.Sc. degree in Computer Science from University of São Paulo (USP), Brazil (2009). He started his career in 2000 as Multimedia Programmer and teaches computer science since 2009. His research interests are software testing, human-computer interaction, information visualization and empirical software engineering.

**Auri M. R. Vincenzi** is Associate Professor of Department of Computing at Federal University of S~ Carlos. Concluded his bachelor in Computer Science at Universidade Estadual de Londrina –UEL (1995) and his master (1998) and doctor (2004) degrees in Computer Science and Computational Mathematics at Universidade de São Paulo – ICMC/USP. He is a member of the Brazilian Computer Society (SBC), the Association for Computing Machinery (ACM) and the Institute of Electrical and Electronics Engineers (IEEE). His research interests include software testing, static and dynamic analysis, and software evolution.