

An Interview Study of GUI Testing Approaches in Brazilian Software Industry

Nailton S. Almeida Jr.  [Federal University of Uberlândia | Federal Institute Baiano | nailton.junior@ufu.br | nailton.junior@ifbaiano.edu.br]

Larissa Rocha Soares  [Bahia State University | larissabastos@uneb.br]

Heitor Augustus Xavier Costa  [Federal University of Lavras | heitor@ufla.br]

Abstract

Context: Users increasingly demand quality in their systems. Software lacking quality can lead to financial losses, legal issues, and even life-threatening risks. To mitigate these problems, testing professionals employ Graphical User Interface (GUI) tests to assess software quality and verify whether applications function as expected by end users. Despite the widespread use of GUI testing by practitioners, few academic studies investigate the actual execution of GUI tests in the industry. **Goal:** This research aims to investigate how GUI tests are conducted in the Brazilian industry from the perspective of software testers. **Method:** Twenty interviews were conducted using a semi-structured script with testing professionals. **Results:** The findings show that experienced testing professionals automate more GUI tests than novice testers. Testers primarily acquire knowledge from their work environment and online resources. Additionally, testers who perform only manual tests generally do not automate due to a lack of programming skills or a preference against programming activities. **Conclusions:** This research contributes to a shared understanding of industry practices in executing GUI tests. The interviews revealed that major reasons for testers conducting manual GUI tests include difficulties with programming-related tasks and a preference for manual evaluations. Challenges related to the testers' organizational context were also evident, particularly concerning deficient project processes such as a shortage of professionals to manage the projects' high workload, issues with the computers provided to testers, and ineffective communication among team members.

Keywords: *Software testing, GUI functional testing, interviews, software testers*

1 Introduction

With the increasing reliance on software, whether for trivial activities like a messaging app or complex tasks such as software used in an industrial process, end-users demand higher quality in the software they use. ISO/IEC 25010 defines software quality as the degree to which it satisfies the explicit and implicit needs of its stakeholders (25000, 2011). Consequently, when software does not meet user acceptance criteria, various issues can arise, such as decreased trust in the software development team, financial losses, legal actions due to contract breaches and risks to people's lives in critical contexts (Alferidah and Ahmed, 2020).

Testers use appropriate testing techniques to analyze whether it meets its defined purpose (Bach, 2022). In this context, there are static and dynamic testing techniques. The first technique aims to prevent errors in software artifacts from causing developers to implement the software incorrectly, usually by reviewing requirements, user stories, flow charts, and the software source code. In the second technique, testers run software to uncover existing inaccuracies. One type of dynamic test is the GUI (Graphical User Interface) test or interface test, which assesses software with graphical interfaces (Banerjee et al., 2013).

In GUI testing, there is a concern to verify whether the events performed on the software screen correctly respond to user actions, such as mouse clicks on a button. The graphical interface facilitates the use of available functionalities; moreover, a substantial portion of the system implementation (approximately 60% of the development effort) is related to graphical features (Jones, 2015). Consequently, GUI testing

enables software evaluation in a manner similar to end-user interaction in a production environment.

GUI testing involves both manual and automated approaches (Amalfitano et al., 2012). Manual testing is essential for verifying functional and nonfunctional aspects through direct interaction (Ramler and Wolfmaier, 2006), and it is recommended for new features prior to automation (Sánchez-Gordón et al., 2020). To overcome manual testing's limitations, automated GUI tests offer better efficiency and reusability across different versions and platforms (Shruti Malve, 2017). Automation is particularly superior for analyzing performance metrics, such as loading times, where manual tracking often yields unreliable results (Molyneaux, 2015).

The literature provides substantial empirical evidence on GUI testing, with numerous studies proposing innovative test automation techniques. For instance, one study introduces a method for automatically extracting interface rules from mobile applications, which serve as test oracles to detect failures without manual verification (Yarifard et al., 2025). Another research effort explores testing to uncover authentication vulnerabilities in Android apps by applying controlled input mutations, this approach effectively reveals flaws even when expected outputs are not explicitly defined (Amalfitano et al., 2025). Additionally, a robotic system has been developed to perform non-intrusive GUI testing on touchscreen devices, simulating real user interactions through computer vision to identify visual and usability issues across multiple platforms (Qian et al., 2025).

The latest systematic reviews conducted by researchers in the field highlight a limitation in relevant studies concerning

knowledge of how GUI tests are conducted in the industry and the identification of the profile of professionals who perform this type of testing. For example, (i) focused on the educational aspect of software testing, grouping publications related to learning in the testing field (Garousi et al., 2020); (ii) brought forth automated techniques for GUI testing in mobile applications through a systematic review (Zein et al., 2016); and (iii) several studies on software interface evaluations were identified by Banerjee et al. (2013). However, upon analyzing these studies, it becomes evident that many do not delve deeply into GUI testing within the industry or fail to centralize their research focus on the testers' perspective.

Between March 10, 2021 and April 9, 2021, we surveyed 222 test professionals in previous work to understand their profiles and how they perform GUI testing (Junior et al., 2021). The results showed that 94% of the participants hold a Bachelor's degree, and the majority work in the Financial Sector, which accounted for 35% of the responses. Additionally, 43% of professionals reported working on projects that follow Agile methodologies. Manual GUI testing remains a widespread practice, with 49% of respondents stating that they test their software exclusively through manual methods. Regarding automation, Java is the most commonly used programming language, adopted by 37% of testers. Selenium leads as the most frequently used tool for GUI test automation, cited by 36% of participants. Nevertheless, around 60% of respondents reported encountering limitations with the tools used in their automation processes.

While previous work offers valuable insights into the practice of GUI testing among industry professionals, many gaps remain and new questions continue to emerge. Considering that GUI testing is essential for ensuring software quality (Tanaka, 2019), and that there is still a lack of comprehensive understanding regarding how it is conducted in real-world industry settings, this study aims to deepen the investigation into GUI testing practices. To that end, we propose conducting interviews with testing professionals to explore their experiences, challenges, and strategies in greater detail. In summary, our contributions are:

1. We conducted a qualitative study that further explores the profile of software professionals who perform GUI testing, with a particular focus on their educational background and how they were trained to work with GUI tests.
2. We highlight the main motivations that lead software testers to perform GUI testing manually.
3. We present a broad overview of GUI test automation based on the accounts of software testers, including the resources used, technologies employed, motivations for automation, and the challenges encountered throughout the process.

Due to the nature of this topic, this work may serve as the first step in encouraging the development of new solutions to address the limitations that testers encounter with manual or automated GUI testing and identify areas for improvement in testing processes.

2 Background

2.1 Towards a Definition of GUI Testing

In formal software engineering taxonomy, such as the guidelines provided by the International Software Testing Qualifications Board (ISTQB), clear distinctions are made between test levels, test types, and testing mechanisms (Klaus Olsen and Ulrich, 2019). A test level defines the target of the test execution, such as system testing or integration testing. A test type refers to the objective of the testing, such as functional or non-functional (e.g., usability) testing. Meanwhile, the testing mechanism or interface relates to how the test is executed, such as through an Application Programming Interface (API) or a Graphical User Interface (GUI). Theoretically, GUI testing is strictly a mechanism used to interact with the software.

However, for the purposes of this research, it is important to establish an operational definition that reflects the reality of the software industry, particularly in the Brazilian context. While formal academic literature maintains strict boundaries between functional system testing and interface-specific validation, our empirical observations indicate that practitioners frequently conflate these concepts.

In this study, GUI Testing is understood as the process of assessing software quality through direct interaction with its graphical interface. This perspective includes not only the verification of visual elements, layout, and responsiveness (i.e., the "look and feel"), but may also involve the validation of end-to-end business rules and functional requirements, especially when the GUI serves as the primary entry point for system-level evaluation.

This broader interpretation is adopted in order to better align with observed industry practices. From the practitioner's perspective, the GUI can often represent the system as experienced by the end user. As a result, references to "GUI testing" in professional contexts may encompass not only interface validation but also broader system-level checks performed through the interface.

In this sense, the GUI can be seen as a practical access point to system behavior, mediating interactions between users and underlying application logic. Consequently, GUI testing may involve the simultaneous consideration of usability, functionality, and integration aspects, even if these are not always explicitly distinguished in practice.

By acknowledging this dual role—both as validation of visual components and as a means of exercising functional system behavior—this work seeks to provide a representation that is more closely aligned with how software quality is assessed in practice. This perspective may be particularly relevant in the Brazilian software industry, where contextual factors such as resource constraints and tooling limitations can influence the adoption of more integrated testing approaches.

2.2 Manual GUI Testing

Testers commonly conduct tests by interacting with the software, analyzing, and reporting the results to the software team (Klaus Olsen and Ulrich, 2019). According to Shruti Malve (2017), one of the advantages of using the man-

ual approach is associated with its low implementation cost, since the tester only needs the software to be evaluated and a device (such as a mobile phone, desktop, or tablet) to access it.

However, manual GUI testing presents a number of disadvantages. One notable limitation is the significant effort required, as it demands considerable tester time, particularly when multiple test iterations are necessary, ultimately impacting the overall efficiency of the testing process (Alégroth et al., 2013). It is also an error-prone approach, as it relies on humans for execution (Grechanik et al., 2009). Additionally, manual tests can become tedious due to repetitive executions (Leitner et al., 2007).

There are two ways to perform manual GUI tests:

- **Exploratory manual execution** are conceived and evaluated dynamically during their execution, meaning they are carried out in a more informal and non-predefined manner (Klaus Olsen and Ulrich, 2019). Because the execution is only partially planned, there is no detailed writing of what should be done in the tests or the expected results. This type of testing is recommended when there is little or no documentation for the software being tested (Klaus Olsen and Ulrich, 2019). The advantages include encouraging the tester's critical thinking and providing an opportunity to learn more about the software's functionality.
- **Planned manual execution** requires the tester to prepare the execution in advance. In this approach, the tester identifies and documents what needs to be tested for each test scenario. Then, in test cases, the steps needed to achieve the test's purpose are detailed (Rothermel et al., 2001). One benefit is the possibility of reusing the test in other evaluation cycles, even if done manually. To achieve this, it is recommended to create high-level test cases without concrete values for input and output data.

2.3 Automated GUI Testing

Due to deficiencies in manual testing, there is a commitment to developing new solutions to automate GUI tests (Canny et al., 2020). Testers develop automated GUI tests in various programming languages (e.g., Java, JavaScript, Ruby, or Python) based on test cases created in natural language. Automated GUI testing allows for quick software validation, reducing the time needed to assess software quality (Karhu et al., 2009) and enabling test reuse and execution on different platforms (Dallal, 2009). Additionally, automated processes can increase test coverage (Sharma, 2014).

Despite the previously mentioned advantages, automating any level of testing, including those performed on the graphical interface, is not a trivial task. It usually requires the tester to possess various skills, such as experience with tools and different programming languages, as well as a clear definition of what will be automated (Tramontana et al., 2019).

With the advancement of research and development related to GUI testing, there have been changes in test automation solutions. The literature categorizes these changes into three generations (Alégroth et al., 2016). The first generation

of GUI test automation relied on absolute screen coordinates (x, y) and emerged in the 1990s to reduce manual testing effort (Alégroth et al., 2015). However, this approach was highly sensitive to changes in screen resolution and window size, leading to frequent script failures.

The second generation shifted to identifying interface elements through logical attributes, such as HTML properties or CSS selectors (Alegroth et al., 2015). Currently considered the industry standard (Alégroth et al., 2015), it is supported by tools like *Selenium*¹, *Cypress*, and *Puppeteer*, as well as commercial solutions. Despite being more robust, this approach still requires manual effort to ensure cross-device and cross-browser compatibility.

Third Generation. Visual GUI Testing (VGT) identifies elements using image recognition techniques (Alégroth et al., 2015). Tools such as *Sikuli*² enable visual validation and user-like interaction but suffer from performance limitations and potential recognition errors.

2.4 Related Work

The literature presents studies that address the topic of GUI testing. Among them, one can find both qualitative and quantitative studies based on the analysis of industry case studies. Below, focusing on works related to GUI testing, the main differences between this research and related studies on the subject are discussed.

Kertusha et al. (2025) provided a comprehensive survey on the evolution of web testing practices, with a focus on the integration of AI techniques in industrial contexts. The authors noted that AI-driven approaches are increasingly being used to address the complexity of modern web interfaces, although practical challenges such as scalability and interpretability remain.

Cui et al. (2024) explored the use of Large Language Models (LLMs) for generating textual input in GUI testing for Android applications. The empirical results showed that certain LLMs achieved page pass rates between 50.58% and 66.67% and were capable of detecting real bugs in open-source apps, demonstrating the growing viability of LLMs as tools in automated GUI test generation.

Amalfitano et al. (2026) surveyed industry practitioners to assess the role of Artificial Intelligence in GUI-based testing. The study revealed that while AI-powered tools are gaining adoption, a lack of deep understanding among users hinders their effective use. Despite this, the potential for AI to support test creation and maintenance was recognized by the participants.

Yu et al. (2023) reviewed over 270 publications on vision-based techniques for mobile GUI testing. The study identified significant advantages of vision-based methods over traditional approaches, including greater adaptability, ease of use, and broader applicability in areas such as test generation, recording and replaying, and the development of testing frameworks.

Polepalle et al. (2022) conducted an exploratory study that gathered evidence on how GUI test automation is perceived and practiced by professionals. The findings highlight

¹<https://www.selenium.dev/selenium-ide/>

²<http://sikulix.com/>

both benefits—such as improved defect detection—and limitations, including high maintenance effort and tool-related issues that can impact the reliability of automation processes.

Ye et al. (2021) also investigated GUI tests on mobile devices, focusing on difficulties in testing entertainment applications. In this study, the authors conducted interviews and applied questionnaires to company development team members. The results showed that the location of graphical elements in the interface is one of the most significant challenges in tests performed on mobile platforms using automated scripts.

Alégroth et al. (2021) proposed a good practice guide for developing third-generation automated GUI tests for a company in Sweden. For this, five test cases previously executed manually were automated using the tools *Sikuli* and *JAutomate*³, aiming to identify the impact of the best practice guide on the test cycle. From the perception of industry testing professionals, the qualitative results showed that testers considered the recommended guidelines inappropriate, superfluous, or unnecessary. Based on the analyses carried out, the authors concluded that the guide produced was ineffective during the software testing phase, as the practices proposed in the document generated extra effort and costs in creating automated test scripts.

Alégroth et al. (2015), driven by industry needs for developing automated GUI tests, with a focus on a Turkish software company, empirically evaluated the *Sikuli* and *JAutomate* tools, aiming to identify the best solution for the company's project demands. Through a static analysis of the functions available in each tool, observation of how professionals used the solutions, and consideration of the opinions of test automation experts, it was concluded that both solutions have limitations that could negatively impact software development, such as generating false negatives during test execution on these software platforms.

Coppola et al. (2020) interviewed ten developers from different companies in Italy, seeking to identify the most recurrent problems in running GUI tests on mobile platforms. From the results obtained, the authors listed several issues associated with running GUI tests, such as the difficulty of executing the same test on different platforms (e.g., *iOS* and *Android*), considering the peculiarities of each operating system. In addition, difficulties were observed in maintaining automated test scripts.

Although the surveys mentioned above are related to GUI testing in the industry, they primarily focus on automated tests—mainly third-generation approaches—while excluding other generations of GUI testing from consideration.

Note that a large portion of the cited works investigates the tools used in the testing process, particularly those related to test automation. Other publications aimed to portray the general landscape of testing activities in the software industry; however, they did not focus specifically on GUI testing practices (or system-level testing). In this sense, both Melo et al. (2022) and Dias-Neto et al. (2017) adopted a quantitative approach to provide a broad overview of the testing context in the Brazilian software industry rather than an in-depth analysis of GUI testing practices themselves. Moreover, these

studies are based on data collected in 2018 and 2006/2013, respectively, which reinforces the need for a new and updated empirical investigation that reflects current industrial practices.

In contrast, the objective of this work is to analyze practices related to the creation and execution of GUI tests, considering both manual and automated activities, as performed by industry professionals. This analysis is conducted through interviews with practitioners, with the aim of capturing the individual perceptions, experiences, and viewpoints of each tester regarding GUI testing in their daily work.

3 Interview Study Settings

In order to collect additional qualitative data and further explore some topics raised in the online survey (Junior et al., 2021), interviews were conducted with testing professionals. The following guiding questions were used.

- **RQ1. What are the motivations for testing GUI manually or automatically?** The objective is to identify the reasons that lead a tester to execute only manual GUI tests and the reasons that lead the tester to automate GUI tests.
- **RQ2. How are manual GUI tests planned?** The objective is to deepen the understanding of the planning performed by testers for the manual execution of GUI tests, seeking to identify the objectives of the tests, what supporting documentation is used for the creation of the tests, the format in which the tests are written, and the reuse of test scenarios created previously.
- **RQ3. What are the main limitations of the manual GUI testing process?** The objective is to identify the most significant limitations that negatively impact the creation and execution of manual GUI tests.
- **RQ4. How is the automated GUI testing process?** The objective is to understand the GUI test automation process performed by the testers, how to validate the tools used for automation activities, the support inputs used to automate GUI tests, the triggers for performing maintenance on automated scripts, and sound design practices for creating automated GUI tests.
- **RQ5. What are the main limitations of the automated GUI testing process?** The objective is to identify which limitations hinder the GUI test automation activity in software development projects.

3.1 Interview Protocol

A protocol was developed for conducting the interviews, based on the design outlined by Adams (2015). The protocol was developed to guide the pre and post interview activities with the participants, also defined the Informed Consent Form, which aimed to inform the interviewees about the research details, their rights, and data protection measures. Access to the interview protocol and informed consent form is provided in Section 8 - Dataset Availability.

During the protocol design phase, it was determined that the interviews would follow a semi-structured format,

³<https://jautomate.com/>

thereby providing interviewees with the opportunity to discuss topics beyond the predefined questions. The decision to conduct semi-structured interviews instead of alternative methods, such as surveys or focus groups, was driven by the need to investigate the root causes of the phenomena identified in our previous quantitative study. While surveys effectively provide a general overview of the industry, they lack the flexibility to explore the “why” and “how” behind the participants answers. Interviews allowed for a deeper investigation into sensitive topics, such as the lack of programming skills or toxic organizational cultures, which participants might be reluctant to discuss in a focus group setting due to social desirability bias. This individual approach was essential to fully address the Research Questions, particularly those regarding personal motivations (RQ1) and organizational limitations (RQ3 and RQ5), ensuring that the data reflected the honest and individual reality of each tester.

3.2 Interview Design

The interviews aimed to identify the professionals’ profiles and motivations, alongside project limitations impacting their testing activities. Accordingly, the 24 questions were structured into three sections following a funnel sequence designed to establish rapport and ensure a natural flow of conversation. This approach began with a “warm-up” phase to collect demographic data before narrowing the focus to technical practices, as follows:

1. General Questions (G): Designed to identify the interviewees’ profile, including education, professional background, and motivations for entering the field;
2. Manual Testing (MT): Specific inquiries for professionals running manual GUI tests, focusing on planning, documentation, and process challenges;
3. Automated Testing (AT): Technical questions for testers involved in automation, covering tools, script maintenance, and implementation difficulties.

This structured progression allowed for a comprehensive understanding of each tester’s experience while maintaining a consistent methodology across all participants. Table 1 presents questions asked in the interviews:

3.3 Pilot Application

Before sending invitations to potential participants, pilot interviews were conducted with testers to identify and implement necessary adjustments to the interview protocol. Three interviewees with different profiles were selected to respond to the questions in the interview script:

- One respondent who performs both manual and automated GUI tests;
- Two respondents who perform manual only GUI testing.

The pilot interviews allowed for the refinement of the questions based on participants’ feedback. Additionally, it was observed that the average time to complete the interview script

Table 1. Interview Questions

#	Q*	Question Description
Q1	G	How many years have you been working in the software testing field?
Q2	G	How did you start working with GUI testing?
Q3	G	What is your level of education?
Q4	G	Where did you acquire the knowledge to work with software testing?
Q5	G	Do you think it’s important for this content to be covered in university education?
Q6	G	Currently, do you conduct testing manually or through automation?
Q7	MT	What is your process for creating manual GUI tests?
Q8	MT	Is your process for creating manual GUI tests documented? Is the documentation reviewed after creation?
Q9	MT	What are the main challenges in creating manual GUI tests? How could these problems be addressed?
Q10	MT	What are the main challenges in test execution? How could test execution be made easier?
Q11	MT	Do you perform any other type of manual testing apart from GUI testing?
Q12	MT	Are you familiar with the technologies used for GUI test automation?
Q13	MT	What prevents you from automating GUI tests currently? Do you have a desire to engage in this activity?
Q14	MT	Does your work environment encourage manual or automated test development?
Q15	MT	How do you feel about executing manual GUI tests in your project?
Q16	AT	What is your process for creating automated GUI tests?
Q17	AT	How is the maintenance of created tests handled later on?
Q18	AT	Is your process for creating automated GUI tests documented?
Q19	AT	What are the main difficulties in creating automated tests?
Q20	AT	How could the process of creating automated GUI tests be made smoother?
Q21	AT	What were the motivations behind automating GUI tests?
Q22	AT	What technologies have you become familiar with for automating GUI tests? What guided these choices?
Q23	AT	How does your work environment encourage the development of automated tests?
Q24	AT	Do you prefer manual or automated test execution? Why?

G = General Questions, MT = Manual Testing Questions, AT = Automate Testing Questions

was approximately 40 minutes. Another aspect analyzed during the pilot phase was the efficiency of the digital platforms, from the digital collection of participants’ signatures to the videoconferencing process.

3.4 Distribution of Invitations

The invitation to participate in the interviews was sent to the target audience composed of testers who perform manual and automated GUI tests. The order in which invitations to participate in the interviews were sent was:

- An invitation was sent to the 222 participants of the online questionnaire previously study.
- Send an email to 100 technology companies;
- Manually send direct messages to over 900 testers on LinkedIn;
- Publish a post in 22 technology related groups on Facebook and LinkedIn.

From the respondents who agreed to participate, 20 interviewees were selected to ensure diversity in seniority levels, professional backgrounds, and testing experience. The

recruitment process employed a multi-stage sampling approach. First, we used criterion sampling by prioritizing professionals who had participated in our previous survey. Then, we applied purposive sampling through direct outreach on LinkedIn and technology-related groups, specifically targeting testers with varying levels of seniority and professional backgrounds.

3.5 Data Collection

The interviews took place between February 3, 2022, and May 7, 2022, via videoconferencing using the Google Meet platform. Data collection was concluded with 20 participants based entirely on logistical constraints. Specifically, no additional potential participants were found, or they were uninterested or unavailable to participate within the stipulated data collection period. Consequently, the data collection phase was closed with exactly 20 participants, driven by timeframe limitations and participant availability.

At the beginning of the interview, the interviewees were informed about the research objectives, asked if they had any questions, and requested permission to record the interview for subsequent data coding. The conversations were recorded using the free software OBS Studio⁴ on two different computers to prevent data loss in case of a failure on one of the researchers' devices (interviewers). The recordings were stored on an external hard drive and encrypted cloud platforms to create reliable backups. Access to both storage locations was restricted to the researchers, preventing any potential "leakage" of sensitive information from the interviewees.

In accordance with the regulations of the Research Ethics Committee for Human Subjects⁵ of the Universidade Estadual de Feira de Santana (UEFS), which follow the Brazilian National Health Council Resolution CNS No. 510/2016⁶, this study was not submitted for ethical review, as it falls under the exemption criteria established by the resolution. Specifically, the research aims at the theoretical deepening of situations that emerge spontaneously and contingently from professional practice and does not involve the disclosure of data that could lead to the identification of human subjects.

3.6 Data Analysis

To begin data analysis, the recorded interviews were transcribed. In the initial version of the transcripts, the interviewees' speeches were fully transcribed in plain text format, followed by a revision of the texts. The interviewees' responses were then organized in an online spreadsheet alongside the respective questions to facilitate data understanding. Subsequently, with the assistance of a second researcher (a master's student), and to ensure better data quality, coding was initiated using the *Delve*⁷ platform.

For data analysis, we adopted *Thematic Analysis* as the overarching qualitative framework to identify, analyze, and

report patterns within the interviews. To ensure a rigorous and structured analysis, the thematic development was operationalized using specific coding procedures derived from *Grounded Theory* (Corbin and Strauss, 2008). Specifically, the analytical process was executed in two systematic phases: (i) open coding, where the interview transcripts were segmented and labeled inductively, and (ii) axial coding, which allowed us to identify relationships between codes and systematically organize them into broader conceptual categories and themes.

Open coding aimed to segment the data into distinct parts and then label them with codes that represent their content. The approach used in open coding was inductive, where codes are created after data collection and processing. The priority order of coding methods was:

1. In Vivo Coding: The interviewees' own words were used to categorize the analyzed data;
2. Descriptive Coding: The interviewees' statements were summarized into a short phrase or term. This method was used only when In Vivo coding was not feasible.

The qualitative analysis was conducted using a systematic coding process to ensure the reliability and validity of the findings. To maintain rigor during the analysis, we employed a negotiated agreement approach. Initially, two researchers independently performed open and axial coding on the interview transcripts.

Upon completion of the individual coding, the researchers held several meetings to compare their results and reach a consensual final result, resolving any discrepancies through dialogue and shared understanding of the participants' statements. To further strengthen the study's objectivity and minimize researcher bias, we conducted peer debriefing sessions, in which two additional researchers—not involved in the initial coding—reviewed the identified categories and their respective definitions to ensure logical consistency and clarity.

4 Results

This section presents the results obtained from the interviews. All information was derived from statements provided by professionals in the software testing field and therefore reflects the participants' perceptions and viewpoints, not the researchers.

Aspects related to the interviewees' profiles, such as academic background, sources of knowledge, and the motivations that led them to become testers, are presented using a quantitative approach. Therefore, regarding the presentation of percentages, values were rounded to whole numbers to improve readability. The remaining subsections, which address the study's research questions (RQ1–RQ5) concerning manual GUI testing and automated GUI testing, adopt a more qualitative approach. In Figure 1, the main categories identified with the codification of the interview data are presented.

4.1 Interviewee profile

Twenty software testers were interviewed. Regarding their experience, there is a significant difference between the pro-

⁴<https://obsproject.com/>

⁵<https://cepuefs.wixsite.com/cepuefs>

⁶<https://www.gov.br/conselho-nacional-de-saude/pt-br/acao-a-informacao/atos-normativos/resolucoes/2016/resolucao-no-510.pdf/view>

⁷<https://delvetool.com/>

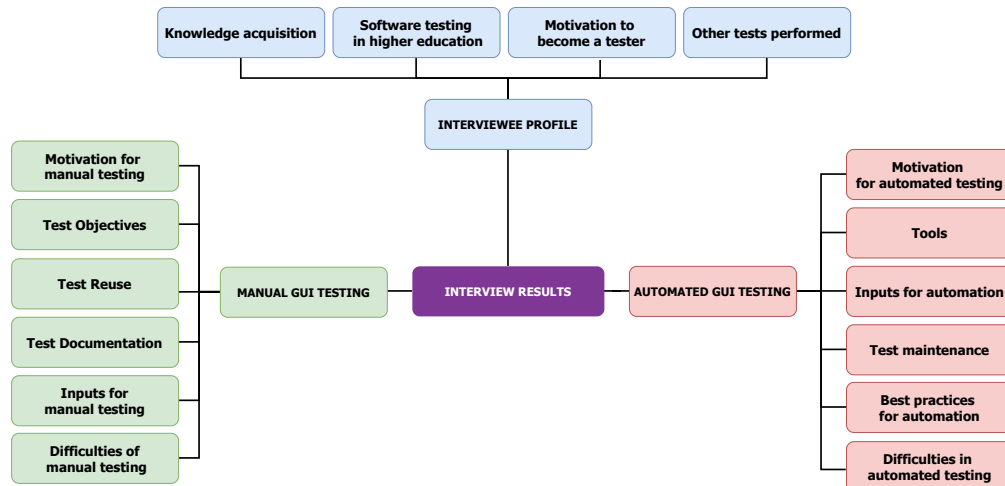


Figure 1. General aspects investigated with the interviews

files of the interviewees. While the 6 out of 20 interviewees who perform only manual tests have been working with GUI for an average of 4 years and 5 months, 13 out of 20 interviewees who perform automated and manual tests have an average of 7 years and 6 months of experience. The only 1 interviewee who performs exclusively automated tests has been in the field for 3 years.

Among the interviewees, three types of software testers were observed: (i) 1 interviewee (5%) performs only automated tests; (ii) 6 interviewees (30%) perform only manual tests; and (iii) 13 interviewees (65%) perform both manual and automated tests. Regarding schooling, 5% (1 interviewee) has only a high school diploma, 10% (2 interviewees) has an incomplete undergraduate degree, 14% (3 interviewees) has an undergraduate degree in progress, 57% (11 interviewees) has a complete undergraduate degree, and 14% (3 interviewees) has a graduate degree. Of the interviewees who have a bachelors degree and postgraduate studies, it was identified that 62% (8 interviewees) are graduates in Information Systems, 15% (2 interviewees) in Systems Analysis and Development, 15% (2 interviewees) in Computer Science, and 8% (1 interviewee) in Process Management.

4.1.1 Interviewees contact with software testing during graduation

Respondents with higher education were asked how the software testing content was taught at graduation. In response, it was observed that the interviewees experienced four contexts during higher education: (i) There was a discipline focused on graduation tests; (ii) Software testing was not covered during graduation; (iii) Topic covered in related course; and (iv) Topic neglected by the institution (Figure 2).

In the context **There was a discipline focused on graduation tests**, interviewees (2 out of 20) reported having an exclusive discipline on software testing, though only at the end of the course. They stated that the content had already been acquired through professional experience and that the discipline repeated topics from other subjects, such as Software Engineering, being taught with superficiality". Interviewee #8 presents: *[...] in college, I later took a Software Test-*

ing subject, but as I already knew how everything worked in practice, it ended up being irrelevant because I already had all that knowledge because of work. It was super superficial because it taught the same contents as the Software Engineering subject, only with a different name; even the test cases were shown much more similar to use cases than anything else, so that is the way I saw it, I do not think important [...]".

In the context **Software testing was not addressed during graduation**, some interviewees (17 out of 20) stated that the theme about quality and software testing were not passed on while they were in higher education, as the graduation had a greater focus on the area of programming, leading them to specialize in testing with other sources of knowledge, as reported by Interviewee #4: *"[...] during my graduation I had contact with programming logic, SQL and even usability testing, but nothing directly to software testing [...]"*.

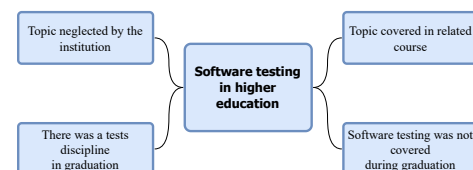


Figure 2. Interviewees contact during their undergraduate studies with the topic of software testing

The **Topic addressed in a correlated discipline** category, interviewees (1 out of 20) reported contact with software testing through the "Software Engineering" discipline, but only at a superficial level, limited to basic concepts like white box and black box testing, without exploring other relevant aspects. Interviewee #9 answer defines this context: *"[...] in college I saw just a hint of testing, the professor in the Software Engineering class took some printed papers talking about development and there was a column small in the middle of the sheet that defined what software testing was, white and black box testing, integration and these things, so it was a very basic thing [...]"*.

In the context **Topic neglected by the institution**, interviewees 3 out of 20 reported a lack of respect for software testing by some professors. Interviewee #11 stated: *[...] when*

the professor commented on the professional testers referred to them as the people over there in the corner testing [...]”. Interviewee #9 mentioned: [...] this teacher I mentioned was also director of the company where I was interning, he did not give any importance to the test area, and when he found out that there were testers in the classroom, he turned his back on us and also went against us [...]”. Interviewee #15 added that the institution indirectly discourages internships in testing, as shown in the comment: [...] several of my colleagues were unable to eliminate the mandatory internship from the course with the experiences they have in the testing area because the course does not allow it, I only managed because I work with automation, I think this is absurd [...]”.

Still addressing the educational aspects, the interviewees were asked what would be the possible benefits for graduates if the topic of software testing were presented with more emphasis in higher technology courses. Respondents (17 out of 20) reported that a discipline with this theme could contribute to the presentation of the software testing area to new professionals since, in some cases, many testers are unaware of its existence, what activities are performed, or its importance in software projects.

Other interviewees (6 out of 20) also emphasized the need to prepare professionals to carry out more quality deliveries in their software development projects in the labor market. As identified in the responses, some developers learn to test only when required by their companies. In addition, for some interviewees (5 out of 20), spreading the culture of quality in their projects is necessary.

4.1.2 Knowledge Acquisition by the Interviewees for GUI Testing

It was observed that testers, in order to perform their testing activities, use different sources of knowledge to learn more about GUI tests. These sources were organized into four categories: (i) Online Content; (ii) Work; (iii) Formal Education; and (iv) Others (Figure 3).

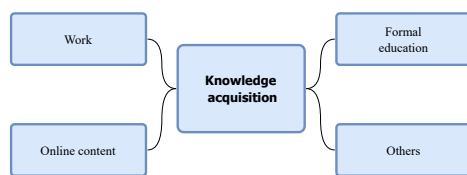


Figure 3. Sources of knowledge for interviewees to act as testers

Among the most cited categories are **Online Content** and **Work**, both with 35% of responses (Table 2). For the type of learning obtained online, the source of knowledge “courses” was the most mentioned with 55% of respondents (11 interview’s) (mainly associated with the *Udemy* platform), 15% of respondents mentioned blogs (3 interview’s), 10% of respondents mentioned search engines (2 interview’s), 10% of respondents mentioned *Youtube* videos (2 interview’s), 5% (1 interview) of respondents mentioned articles and 5% (1 interview) of respondents mentioned technical documentation.

In the **Work** category, 40% of respondents (8 interview’s) said they learned about tests from tacit knowledge (with another professional passing on their experience in the area),

40% of respondents (8 interview’s) mentioned that they learned about tests by executing past demands in the day to day at work and 20% of respondents (4 interview’s) learned by participating in training at the company they worked for.

Table 2. Sources of Knowledge of the Interviewees

Acquisition Type	Source	%
Work	Tacit Knowledge	35%
	Daily Activities	
	Internal Company Training	
Online Content	Search Engines	35%
	Articles	
	Courses	
	Blogs	
	YouTube Videos	
Others	Technical Documentation	20%
	Self-taught	
	Certifications	
	Books	
	Handouts	
Formal Education	Technical Course	10%
	Undergraduate Degree	
	Postgraduate Degree	

The last mentioned category is related to **Formal Education**, for example, through a Higher Education Course. Among the interviewees, 65% (13 out of 20 interview’s) said they learned about tests while taking a Graduation, 20% (4 out of 20 interview’s) reported that they learned in a Postgraduate Degree and 15% (3 out of 20 interview’s) said that they learned in a Technical Course. Knowledge sources that do not express the other types of acquisition were categorized as **Others**. Of the responses stated for this category, 40% (8 out of 20 interview’s) of respondents learned self-taught, 40% (8 out of 20 interview’s) of respondents learned through certifications, 10% (2 out of 20 interview’s) of respondents learned from books, and 10% (2 out of 20 interview’s) of respondents learned from handouts.

4.1.3 Motivations for the respondent to become a software tester

In the professional aspect of the interviewees profile, the reasons that impacted professionals in choosing the role of software tester were investigated. Nine reasons were identified: (i) Friend recommendation; (ii) Self-interest in the area; (iii) Dissatisfaction with current job; (iv) Company demand; (v) Financial reasons; (vi) Difficulties with programming; (vii) Invited by a company; (viii) Wrong recruitment; and (ix) Failed to enter another area (Figure 4).

The **Friend recommendation** category was the most mentioned by the 9 interviewees, in which they stated that they started in the Software testing area by encouragement or invitation from someone they knew who worked in the software development industry as a tester. As can be seen in the interviews, this Recommendation for the software testing area occurred both at the beginning of higher education and in the professionals careers and for respondents who worked in the technology area in some other context.

The **Self Interest** category motivated respondents to test software. From the content about tests seen in graduation (even superficially as informed by the 4 interviewees) and the self-taught study carried out, the interviewees were enthusiastic, leading them to look for vacancies to act as testers.

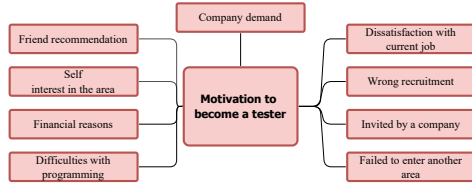


Figure 4. Reasons that led the interviewee to start working in the software testing area

In the **Dissatisfaction with current job** category, it was observed that 4 respondents were dissatisfied with the activities they carried out in their teams, either due to lack of knowledge to perform them or because they did not see value in the activities. In addition, there was dissatisfaction as a whole with the progress of the project in which they were allocated since, in some cases, there were idle moments not used for the professional development of the interviewees, with no incentive to expand their competence.

Due to the lack of testers in the software development teams, it was identified that, due to the **Demand of the company** in which they worked, 4 interviewees were transferred between sectors and allocated to carry out activities related to the software testing area. Respondents stated that they did not have the necessary skills to perform the activities at the time of this internal sector transfer. However, the company still made its allocation to assess the quality of the software.

The search for better salaries composes the **Financial motives** category. For respondents dissatisfied with their salaries, the promise of better pay in software testing made them look for a position to work as a software tester. This motivation was cited by 3 interviewees with careers in the technology sector and graduates in other sectors (e.g., Performing Arts), making them migrate to software testing.

In the **Difficulty with programming** category, 2 respondents reported that because they did not know how to perform the software developer role, they decided to specialize in assuming the role of software tester in technology projects. Furthermore, it was mentioned by the interviewees that they do not like to code, encouraging their participation in the area of software testing.

The **Invited by a company** category reflects responses referring to 2 respondents who initially did not think of acting as a tester but were invited by companies that had vacancies for this position and that they had not applied for, as informed by Interviewee # 20: “[...] To be honest I didn’t have a specific motivation, it just happened. I was about to join an IT company because I had a lot of experience with support and maintenance, I had an interview with the manager, but the vacancy I was going to fill ended up closing. Three months later, the manager called me asking if I was interested in a vacancy to work with software testing in one, and I accepted the offer [...]”.

The **Wrong recruitment** category refers to 2 respondents who participated in selection processes believing that the va-

cancy was for a sector of the company; however, the selection was aimed at filling positions for the software testing area. As informed by Interviewee #14, only when the company human resources team was hired it was notified that the vacancy was to be a tester: “[...] at the time of selection, it was informed that the vacancy was not for database as I expected, but for software testing [...]”. In another report, the respondent reported that he was taking a course to specialize in technology; however, the respondent was undertaking an internship program and was not aware of this.

The search for an internship vacancy and the lack of opportunity contemplate the category **Couldn’t get into another area**. The 1 interviewee, who did not obtain an internship vacancy in another technology area, accepted to act as a tester in the first invitation they received, as informed by Interviewee #5: “[...] I entered the testing area through an internship. All the internships I was looking for required experience beyond what I had, and this one in the software testing area was the only one I passed [...]”.

4.1.4 Other tests performed by respondents

In addition to GUI tests, respondents usually perform other types of tests (Table 3), the most cited being the API test by 15 respondents (49%). The other types are load testing, cited by 6 respondents (20%), usability testing cited by 4 respondents (13%) and compatibility, performance, unit, accessibility, smoke and component tests, cited by 1 respondent (3%) each. The API test may have had more citations because of the popularity of the use of interfaces for exchanging information by companies, making testers need to assess whether the requests sent or received have the expected data. The total number of reported test types exceeds the number of participants, since one respondent indicated performing multiple types of quality assessment as part of their testing process.

Table 3. Other Types of Tests Performed by the Interviewees

Test Type	#Interviewees	%
API	15	49%
Load	6	20%
Usability	4	13%
Compatibility	1	3%
Performance	1	3%
Unit	1	3%
Accessibility	1	3%
Smoke	1	3%
Component	1	3%

4.1.5 Characterization of the Companies

The positioning of the companies where the interviewees work in relation to the execution of GUI tests was investigated. It was observed that some companies prioritize the manual execution of GUI tests, according to the interviewees who perform manual tests (19 out of 20). According to them, there is no allocation of hours in the project for automation, explaining that management prefers manual tests. Furthermore, when products under development are new, it requires manual evaluation at first.

In addition, a tester mentioned that, in terms of results, the organization he works for looks for activities that bring short-term returns, forcing the test team only to perform manual tests. Also, he was informed by another interviewee that, due to the lack of testers in his company, the priority is to “put out fires”. Thus, the focus is on testing urgent fault corrections, with no room for the evolution of software quality processes.

Finally, one respondent stated that manual tests are carried out without formality in his project, as their execution is done by the project manager, not by a qualified professional in the testing area.

4.2 RQ1 - What are the motivations for testing GUI manually or automatically?

In the study, it was identified that most professionals still perform GUI tests manually. Thus, the motivations that encourage testers to choose the manual or automated approach when performing GUI tests were investigated during the interviews. First, the reasons for the exclusive execution of manual GUI tests are presented, and then the reasons that led testers to automate GUI tests are discussed.

4.2.1 Reasons for manually testing only

Even with the growing popularity of tools and techniques for automating GUI tests, many professionals still perform GUI testing manually. Categories related to the reasons why respondents do not perform automated tests were identified: (i) Professional preference; (ii) Educational issues and (iii) Project context (Figure 5).

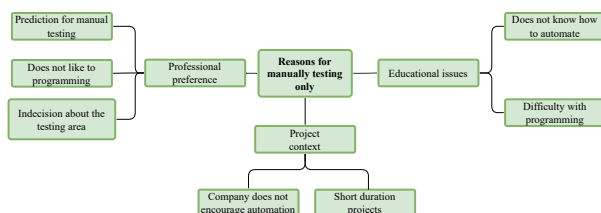


Figure 5. Reasons why respondents only perform manual GUI testing

1. Professional preference

Below are the personal and professional reasons that make respondents not automate tests. Among the types found are:

- **Predilection for manual testing:** 3 interviewees prefer to perform tasks associated with manual testing rather than automated ones; that is, they feel more comfortable performing activities more linked to the interpretation of business rules and specification of requirements than the coding of automated test scenarios;
- **Does not like programming:** As informed by 1 interviewee, the act of programming, be it an automated test or the coding of software, is not an activity that gives pleasure or satisfaction to him. Because of this preference, the respondent only performs manual tests;

- **Indecision about the testing area:** 1 interviewee recently started in the testing area and informed that he is still getting to know the universe around software quality. He was undecided whether to continue performing only manual tests or whether to go professional for test automation.

2. Educational problems

In the Educational issues category, the limitations and lack of knowledge that prevent the production of automated GUI tests by the interviewees are discussed. The problems are detailed below:

- **Does not know how to automate:** 2 respondents are still becoming familiar with manual tests and do not have knowledge about test automation, impacting the execution of automated GUI test scripts;
- **Difficulty with programming:** 5 interviewees listed having problems understanding and using programming logic and applying it with some programming language in creating automated tests. Terms such as “resistance” and “dread” were associated by respondents with the action of programming.

3. Project context

In the Project context category, it is detailed what are the organizational variables that impact the production of automated GUI tests, among them are:

- **Company does not encourage automation:** According 10 interviewees, companies allocate their testers only to activities related to the creation and execution of manual tests, not allowing them to automate GUI tests, even if they have the necessary skills for the service. Furthermore, as testers are always investing their efforts in carrying out manual tests, they are overwhelmed by the repetitive and tiresome activities, having little time to automate any tests;
- **Short duration projects:** Projects with a short deadline to be delivered and the team responsible for creating the software that will not maintain the source code later tend not to receive GUI test automation, according to 1 respondent. In this case, the software development cycle does not support the test automation process, which needs more time to be efficient and bring benefits.

4.2.2 Reasons for automated testing

Three categories were identified related to the reasons that lead respondents to develop skills to perform GUI test automation tasks in their projects: (i) Educational reasons; (ii) Project reasons and (iii) Professional reasons (Figure 6).

1. Educational Reasons

The following is the educational motivation for automating GUI testing:

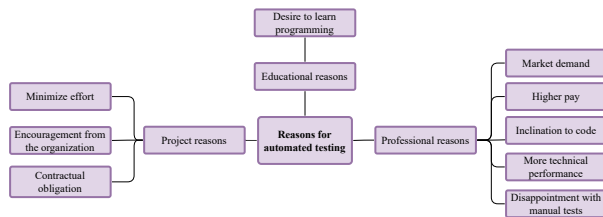


Figure 6. Reasons to be a tester who automates GUI testing

- **Desire to learn programming:** Learning to code in a programming language motivated 1 interviewee to get to know the area of software development, leading him to use the experience obtained with software creation for the test automation activity of GUI.

2. Project Reasons

The specific context of the software development project in which the tester operates plays a significant role in shaping how GUI tests are performed. The motivations related to the project, as reported by the interviewees, are outlined below:

- **Minimize effort:** Decreasing the time to run tests and increasing the number of validations performed in the software were the main reasons given by 5 respondents for them to start in the area of test automation. With automation, in addition to achieving the advantages above, the test team can perform other activities that are less repetitive than manual GUI testing;
- **Encouragement from the organization:** 2 some interviewees were encouraged by the companies they work for to learn about automation in order to apply the knowledge in the company projects, aiming at the gains that test automation provides in the software development flow;
- **Contractual obligation:** it was informed that by 1 interviewee some contracts with customers would only be carried out by creating automated tests to assess the quality of the software to be produced. Since automation was a must for the project owner, the company required testers to create automated GUI tests.

3. Professional Reasons

Many testers prioritized running GUI tests for reasons related to their professional career. Below are the motivations reported by professionals in the testing area:

- **Market demand:** For 8 interviewees, the job market itself required testers to have the necessary skills for test automation. According to them, more and more job openings published list, among the mandatory requirements, knowledge in programming and test automation, in addition to company managers emphasizing the need for the candidate to prove that they know the frameworks for GUI test automation;

- **Higher pay:** 4 respondents noted that salaries for testers who know how to automate tests are higher than for testers who run manual GUI tests. Even in companies with no space to automate, there is greater salary appreciation for professionals who, at least, have the necessary skills for automation. As a result, some respondents reported that salary was one of the reasons for focusing on automated testing;
- **Inclination to code:** 6 interviewees stated that they like to code, and the test automation area was the space they found to contemplate this desire;
- **More technical performance in the project:** disconnecting from activities related to project management and manual testing was one of the reasons identified in the interview. For 1 interviewee, the desire to be a technical reference in the development project and to know the tools and the entire automation process would be the reasons for automating GUI tests;
- **Disappointment with manual tests:** 14 interviewees also reported a lack of interest in performing manual tests, as performing only manual validations on the interface can be tiring. In addition, it can generate a feeling of career stagnation due to repetitive and “non-technical” activities.

4.3 RQ2 - How are manual GUI tests planned?

To understand in more depth how manual GUI tests are planned for their later execution in software development projects, the respondents responses were separated into categories referring to the planning activity. The following sections cover the categories: (i) Objectives of the test; (ii) Inputs for the tests; (iii) GUI test documentation format and (iv) Reuse of documentation produced for GUI tests.

4.3.1 Objectives of the test

When planning and executing the GUI test, it is necessary to identify the purpose of the tests, that is, the focus of the quality assessment. Five types of objectives were identified: (i) Essential features; (ii) Prototype validation; (iii) Elements available in the interface; (iv) Software responsiveness and (v) Software usability (Figure 7).

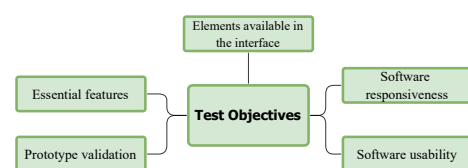


Figure 7. Objectives of Manual GUI Testing

Among the types of objectives, the **Essential Functionality** objective stands out. By 1 respondent, prioritize the

validation of business rules with basic test scenarios and a smaller number of alternative scenarios, as mentioned by Interviewee #15: “[...] if I took a login feature to test, I created a test informing a correct password, another for an incorrect password or just one with the wrong user”.

The objective **Elements available in the interface** represents the evaluation of the availability of use of the fields displayed on screen and the spelling of the respective associated texts, such as the text of a placeholder or the title of an input. In addition, 1 interviewee observed other characteristics not directly related to the GUI test. An example is the responsiveness of the software due to the diversity of devices on the market, as it is necessary to check the software on screens of different sizes, such as computers or smartphones.

Regarding the objective **Software usability**, it was observed that the tester often needs skills beyond those required for carrying out GUI tests, such as being responsible for verifying the usability of the tested software. Interviewee #4 comment discusses this: “[...] Here where I work there is a UX Design, but in my past experiences not so, so I also had to think about the usability of the software and even accessibility during scenario planning [...]”.

In the objective **Prototype validation**, the 3 interviewees explained that they need to assess whether the prototypes developed as the first version of the software meet the documentation that maintains the previously specified requirements. Regarding the **Software Responsiveness** objective, the interviewees evaluate in the test whether the software, when used on different devices, respects the screen dimensions. The evaluation of software responsiveness occurs due to the diversity of devices with different sizes that could access the application, especially if it is Web software.

4.3.2 Inputs for manual tests

Respondents use various types of documentation as support material for the creation of GUI tests, for example: (i) User story; (ii) Business rule; (iii) Developer Report; (iv) Requirements; (v) Acceptance criteria; and (vi) Refinement participation (Figure 8).

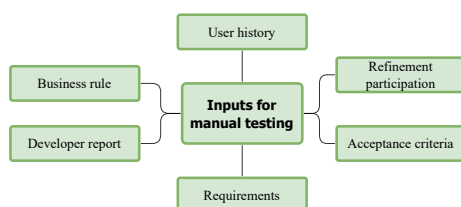


Figure 8. Inputs to create manual tests

A majority of the participants (13 out of 20) reported using user stories as supporting artifacts for planning GUI tests.

Another input related to Agile Methods is the **Refinement Participation**, in which 5 testers related obtain details regarding the new sprint functions in the requirements refinement meeting with the product owner. Interviewee #3 informs that: “[...] in the technical refinement with the developers, the technical point of view of the story is explored and how it will be done, and in this section, I already define what

the scenarios will be of tests we want to explore and how the story will be tested [...]”.

The input **Acceptance criteria** is complementary to the input **User stories** in the process of creating GUI tests (AltexSoft, 2023). By 2 interviewee’s, with these criteria, it is possible to have a correct definition for the task, as informed by Interviewee #3: “[...] business refinement is focused with the Product Owner and the Business Analyst to understand how from a business point of view, what are we going to do in the sprint; in this part, we focus on creating well-defined acceptance criteria, so that we can later prove the completion of the user story [...]”.

In the view of the 4 interviewees, agile methodologies may have some disadvantages, for example, the lack of details that impact the understanding of the software and the performance of tests. This was highlighted by Interviewee #6: “[...] what I miss today is more explanatory documents to support tests in the Agile methodology, today there is no information about the features or just the basics which can be used for writing tests, so I think it has this deficit [...]”.

More “traditional” inputs were also found, such as **Requirements** and **Business rules**, formatted to suit the Cascata development model. In addition, Interviewee #18 reported that, in order to create and run the tests, he asked the developer who changed the software what was done in the new function, as he reported: “[...] asking that the developer let me know what he wanted me to test because the application here is robust and I need to know what he implemented for me to test. From the developer report about his change, I would do the tests and deliver an execution report [...]”.

4.3.3 Form of documentation for GUI tests

Regarding the format in which GUI tests are written, Figure 9 presents the formats most cited by respondents to document GUI tests: (i) Gherkin; (ii) Mind map; (iii) Test topics; and (iv) Step by Step. The format in which GUI tests are written is often impacted by the methodology used in the software development process. Six Interviewees who work with an Agile methodology often use Gherkin as the standard format for writing GUI tests.

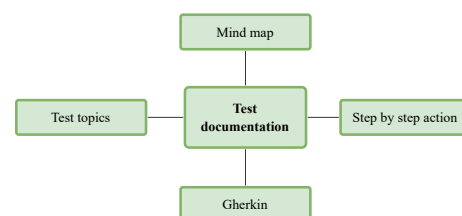


Figure 9. Formats for creating manual GUI tests

The **Mind Map** format is the most popular format for writing among 3 respondents. This format facilitates the validation process, as reported by Interviewee #10 “[...] I have also used mind maps to help me think about scenarios, identifying the test points I needed to perform not to get something so extensive [...]”. Seeking agility in writing tests, Interviewee #9 said he tries to reduce the effort in this part of the process: “[...] I usually do not describe all the test cases because with

the Agile methodology, you do not have to write all the details; I make the topics to remember what I need to test [...]”.

On the other hand, 3 respondents who use the Waterfall model define test scenarios with all the steps in detail for execution. For example, Interviewee #15 reported that: “[...] I prefer the traditional method of writing steps because if the test failed, the developer would know at what point the test had broken.” In addition, to support the writing of GUI tests, respondents use support techniques for defining scenarios, such as: (i) Pareto Rule, (ii) Analysis of limit values, (iii) Equivalence partition; (iv) Decision tables, and (v) Traceability Matrix.

4.3.4 Reuse of documentation produced for GUI tests

As for the reuse of GUI tests, Figure 10 presents the situations in which respondents use the tests previously created: (i) For regression tests; (ii) No reuse; (iii) Sporadic use; (iv) Personal control; (v) Test automation; and (vi) As documentation.

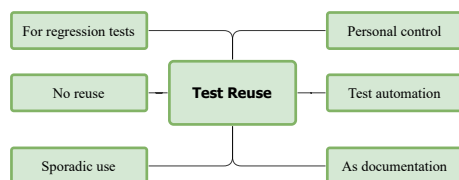


Figure 10. When manual tests are reused in the project

Manual GUI tests, once created, can be used again in the software development project. Thus, testers can use the documented test scenarios more than once. Of the 20 respondents, 8 respondents (40%) reuse tests for regression evaluation, 4 respondents (20%) do not reuse created tests, 2 respondents (10%) partially reuse existing tests, 2 respondents (10%) consult the tests in a personal control sporadically, 2 interviewees (10%) reuse the tests in the automation process and 2 interviewees (10%) use the tests as a documentation of the software under evaluation, making the tests available as a manual for users.

4.4 RQ3 - What are the main limitations in the manual GUI testing process?

Considering that performing manual GUI tests is done in two steps, the first being the design of test scenarios and, subsequently, the execution, the limitations identified for manual GUI tests were categorized in order to contemplate the two steps informed.

4.4.1 Difficulties and solutions in creating manual GUI test

The enormous difficulties that the interviewees find in their projects in writing the test scenarios and the possible solutions for these problems were organized into two categories: (i) Process Limitations; and (ii) Professional Limitations (Figure 11).

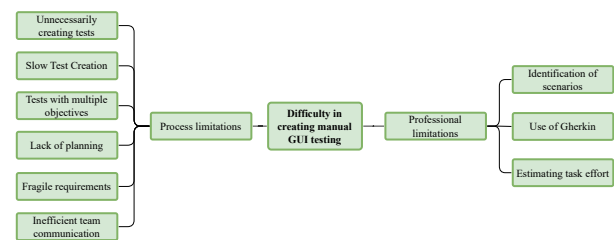


Figure 11. Issues related to creating GUI tests

1. Professional Limitations

The Professional Limitations category references the respondents lack of ability to create GUI tests, such:

- **Identification of scenarios:** Finding the software scenarios that should be tested is a challenge for 4 interviewee's, mainly due to the complexity of the software under tests that generate a long learning curve and the lack of documentation in the projects. The consequence is tests with poorly detailed actions and poorly defined assertiveness. To avoid the problems above, a commonly used way to identify requirements is through the practical use of the software to validate the output returned by the tester interaction, as it is common to have difficulty writing test scenarios until you have a good understanding of the product being used. Customer/user involvement during requirements creation was also pointed out as a possible solution to facilitate the test creation process. For example, the customer close to the project can review the information and see if anything is incorrect. In addition, testers can be included in refinement meetings and help resolve any doubts about the new function to be developed in the software;
- **Using Gherkin⁸:** Creating a test block in Gherkin format that is not extremely specific can be challenging for three software testers. The test scenario needs to be general enough to be reused in other test evaluations; still, the test must contain reasonable detail for the tester to read it and perform the expected actions using the software;
- **Estimating task effort:** Identifying the effort to be spent on designing and planning tests is another difficulty related to the testing process for one tester, especially when you have no idea of the software functions to be evaluated.

2. Process Limitations

The Process limitations category refers to problems that occur in the project in which the interviewees are allocated, the following points are detailed:

- **Unnecessarily creating tests:** 6 testers create tests in the *Gherkin* format, but project people do not consult the produced inputs. For 3 interviewees, this is considered a waste of effort

⁸<https://cucumber.io/docs/gherkin/reference/>

and consumes time that could be used in other software quality activities. The use of automated GUI tests as living documentation of the project is one of the interviewees recommendations to solve the problem in question. Instead of having a requirement, a manual test scenario and the same coded test for automation, it is possible to minimize the team effort invested in creating and maintaining various artifacts with the centralized use of automated scripts for understanding the software;

- **Tests with multiple objectives:** Testing different aspects of software in a single evaluation can impact the quality of the test. Planning the evaluation of business rules and several interfaces in the same test can impact the focus of the activity and, consequently, the quality of the final test result;
- **Slow Test Creation:** Documenting multiple test scenarios is a challenge in projects with many functions to be tested. Therefore, 7 interviewees suggest the use of solutions based on artificial intelligence for identifying and writing test scenarios, aiming to reduce the effort of these activities. Also, another suggestion is to change the test writing format by replacing detailed test scenarios with mind maps. Generally, they are easier to understand because they are more visual, and the production time of this material is shorter compared to tests written in a more traditional way. Storing test scenarios to be reused later can also minimize the work of writing tests, especially in projects that require a large volume of creation and execution of tests, such as in regression tests;
- **Lack of planning:** The activities of the software quality team may not be considered in the final deadlines for product delivery. Sometimes, it may be required that the design, writing, and execution of test scenarios must occur in a short time (e.g., a single day of work), generating effort on the part of testers and impacting the final quality of the product ;
- **Fragile requirements:** One of the biggest problems related (11 interviewee's) to the process is the fragile requirements of the project. Documentation of these requirements often lacks sufficient detail to provide satisfactory test coverage. For example, Interviewee #2 reported that there is a lack of knowledge even by the professionals who should specify the requirements “[...] usually the requirements team does not know the business rules and even informing which needs they need to be remedied with the functionality, they still leave gaps in the documentation that make it difficult to plan the tests and even the development of the feature. In addition, customers

also do not know how to inform what they expect from the software.” Incomplete or poorly written documentation can impact the process in several ways, for example, rework in writing/executing tests, as the software does not meet the customer wishes, and bugs can go unnoticed by the tester, as he does not have enough information to carry out the tests. One solution to solve this problem is to use mockups. This can make it easier to understand how the final software should be or what functions should be available in the user interface, especially if the software is still in its infancy. Also, there is the possibility of identifying the maximum amount of information when specifying the requirements, such as what needs to be done and the users needs, so that the tests take place in a more consolidated context, leaving less margin of error for developers and testers. User stories could be distributed to the team with enough sequence of steps to understand the software flows. Additionally, there is the aggravating factor that, in some teams, there is no professional specialized in gathering and specifying requirements, generating shallow documentation in the projects, according to the interviewees. Positions like the Scrum Master deal with the requirements rather than having a position focused on writing user stories;

- **Inefficient team communication:** A problem reported by 6 respondents that impacts requirements indirectly is the lack of communication between team members. Changes are made to non-shared specifications. The documentation lacks detail and still undergoes constant changes that are not shared with the quality team or even the developer assigned to the coding task. Interviewee #7 informs the difficulty in its context: “[...] what happens a lot is that the feature is already being developed, its prototype is ready in Figma, and the user story is already ready, but after one or two weeks of development there are changes in the software interface, with that it is necessary to change the test scenarios that are already ready, the documentation and everything else [...]”. The requirements of the software being developed and subsequently used in the test planning process must be communicated by the team stakeholders to align expectations of how the software should work in its final version. Thus, if there are changes in the specifications, the team must be notified to avoid problems related to incorrect versioning of information.

4.4.2 Difficulties and solutions in running manual GUI testing

Regarding the execution of the manual GUI test, three types of limitations were found: (i) Process Limitations; (ii)

Technical Limitations and (iii) Organizational Limitations (Figure 12).

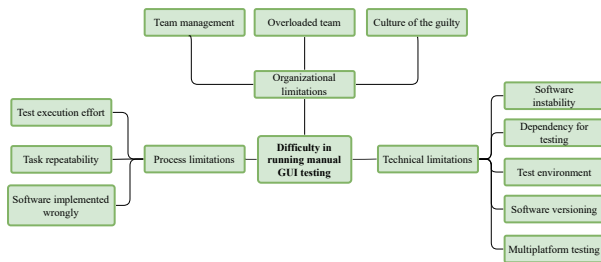


Figure 12. Issues related to running GUI tests

1. Process Limitations

As with the limitations for creating tests, running GUI tests also has difficulties generated by the organization of the process used in the project. The following are the process limitations:

- **Software implemented wrongly:** Based on intuition or “guessing”, the developer codes the software not respecting the specified requirements, which align with the customer expectations. As the software was produced erroneously, design time is lost, as the tester can “waste” a test run on software that is not in its final version, returning the task to the developer to redo the entire process;
- **Test execution effort:** The need to manually evaluate various aspects during test execution, such as business rules and visual elements, represents a problem for software development projects. According to the 3 interviewees, due to the lack of objectivity in the test, time is wasted analyzing details that, in their opinion, are irrelevant to the software. In addition, more critical points are left out because of this excessive test responsibility. Because many developers do not respect the requirements at the time of coding, the tester may not trust his teammate work. Thus, to prevent any flaws from passing into the final version of the software, the tester may find it necessary to test all of the software again, even functions not directly related to the one the developer had changed;
- **Task repeatability:** Testing the same function more than once, whether or not in a regression test, to try to identify new errors, can impact the quality of test execution. Running the same tests multiple times does not guarantee that new failures will be identified. Furthermore, repeatedly performing the same activities can cause testers to develop a “trained eye” and lose critical perspective on what they are testing, according to the 3 testers.

2. Organizational Limitations

The Organizational limitations category corresponds to

the difficulties generated by the organization culture in which the interviewees work, together with the project management choices or lack of action. The identified limitations were:

- **Team management:** According to two interviewees, teams made up of professionals with no experience can impact the quality of test execution. Important positions, such as project managers and Scrum Master, when filled by people with no experience, can lead to delays, changes in development methodology during project execution, and, consequently, the need to work overtime;
- **Staff overloaded:** Due to deficiencies in planning activities and consequent pressure on professionals, many said they were unmotivated and had low productivity. In this context, they usually receive the software very late, with little time to run the tests. Some said they needed working late at night to be able to assess test scenarios before the deadline;
- **Culture of the guilt:** When a problem occurs in the project, instead of finding an adequate solution in a collaborative way among the interested parties, there is an attempt to blame only one person for the existing crisis. For the 2 interviewees, this leaves the work environment toxic and demotivating for everyone.

3. Technical Limitations

The Technical Limitations category corresponds to issues generated from the framework in which the GUI tests are run. The following limitations were found:

- **Software instability:** The performance of the software version to be evaluated can slow down the execution of GUI tests. This problem can confuse the tester if the low response of the software is a bug, which must be registered for the developers to evaluate, or a behavior mapped by the team, hindering the finalization of the tests. The developer must ensure that the test environment and software are working as expected before asking the quality team to evaluate their task;
- **Dependencies for the test:** The lack of mass of data and the external integrations performed with third-party software for the tests is the biggest complaint of the interviewees (9 interviewee’s) about the manual execution of the GUI. The necessary preconditions for the execution of the tests are disregarded when planning the project, and the impact ranges from the lack of accredited users to login to the software, the impartial use of functions due to the lack of permissions to the excessive time to configure the software, which many times is longer than the test time itself. One way to mitigate this problem

is to give the tester access to create the necessary dependencies for the test. If he does not have technical competence, the necessary skills must be passed on so that he can produce the data for the tests;

- **Test Environment:** In many projects, there are no test environments for testers to run their assessments. In addition, when the test environment is created, the hardware structure can respond deterministically, with constant drops and slow access. As noted in some responses, the tester has no barriers to understanding the software or the testing task, but the approval environment is out of the air. It is expected by the interviewees that the homologation environment has a structure as close as possible to the production environment, with sufficient stability for the tests to be executed. Another solution is to use *Docker*⁹, so that environment maintenance is done more dynamically and with more control. In addition, it was pointed out that these problems could be solved with the participation of a DevOps professional in the team, who would be responsible for the project infrastructure and access;
- **Software versioning:** The tester receives the software to evaluate without the latest changes in the production environment. Thus, the professional tests the application in an old and outdated version concerning what end users use. It is the developer responsibility to know how to use the code version control software, as well as manage the branch version he is developing;
- **Multiplatform testing:** The need to test software on multiple screens exists because so many devices on the market have different screen sizes. Most of the time, there are two solutions to this situation (i) buy different devices, such as smartphones from different brands, which involves a significant investment, or (ii) use a tool for cross-browser testing (use a platform that simulates different devices). However, this simulation has some caveats, for example, connection and latency problems, as these services are usually hosted in other countries.

4.5 RQ4 - How is the automated GUI testing process?

In order to understand the planning of automated GUI tests, four categories related to the automation process were evaluated: (i) Tools for test automation; (ii) Inputs for automated tests; (iii) Maintenance of automated tests; and (iv) Best practices for test automation. Notably, at the interview stage, 70% of respondents reported performing automated GUI tests.

4.5.1 Tools for GUI automation

It was investigated about the tools used in the test automation process. It was identified that respondents use solutions mainly from the second generation of automated testing tools, except the *Eye Automate* tool, which is part of the third generation (Table 4). Supporting frameworks such as *Cucumber* and *Behave* were also reported by participants; however, they assist with the use of *BDD* (Behavior-Driven Development) but, they are not tools for GUI test automation. In addition, respondents reported that they use the mentioned tools with the following programming languages: 33% (7 respondents) Java, 19% (4 respondents) Python, 19% (4 respondents) Javascript, 19% (4 respondents) C#, 5% (1 respondent) Cobol and 5% (1 respondent) Ruby.

Table 4. Automation Tools Cited by the Interviewees

Tools	Type of Automation	#Interviewees
Selenium	Web	11
Robot Framework	Web	10
Cypress	Web	9
Selenium IDE	Web	3
Capybara	Web	2
Protractor	Web	2
Appium	Mobile	2
Playwright	Web	1
Katalon	Web/Mobile	1
UFT	Web	1
Eye Automate	Web	1
Espresso Café	Mobile	1
Test Cafe	Web	1
Test Complete	Web	1

4.5.2 Inputs for automated GUI testing

Figure 13 presents the set of project inputs used to support the GUI test automation process. Some interviewees reported that they perform automation without using any formal documentation previously created. Among those who use it, using manual GUI test scenarios was the most cited by them (8 interviewees). In addition to scenarios, user stories, and other business rules were reported as supporting documents for GUI automation. Also, the use of Product Owner planning was mentioned.

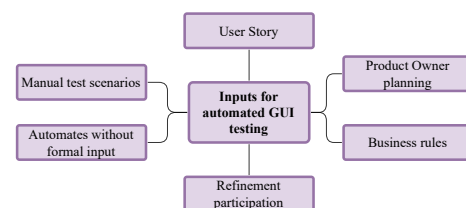


Figure 13. Inputs used in the GUI test automation process

4.5.3 Maintenance of automated GUI tests

Considering that one of the most significant efforts after the creation of automated tests is the maintenance of the created

⁹<https://www.docker.com/>

source code, Figure 14 shows the moment in which the planning for the revision of the automated GUI tests takes place in the development projects of software in which testers are allocated. The biggest motivator for test code maintenance is the false negatives returned after running the test, mentioned by eight out of fourteen respondents. Thus, after analyzing the execution results, a fault is identified in the test code and not in the software being evaluated, showing a more corrective than reactive character for maintenance.

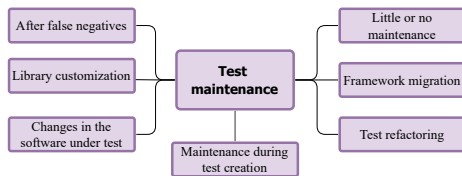


Figure 14. Reasons for performing maintenance on automated tests

Also, alterations related to the test frameworks were identified: three interviewees reported performing customizations in the test frameworks and libraries, which generated changes in the test code. One of these interviewees reported that his team also had to change automated tests when migrating from a test framework from *Selenium* to *Playwright*.

Software changes by the developer also generate maintenance, as reported by three respondents who perform corrections in the automated test code. Another three interviewees mentioned more reactive maintenance when they also review existing codes when creating a new automated test. One respondent reported that maintenance occurs with refactoring bias, recreating all automated test scenarios following good practices. Finally, three interviewees stated that, in their projects, there is little or no code maintenance, the team focus is on creating new tests.

4.5.4 Best practices for GUI test automation

It was investigated whether the interviewees had in their projects any document of good practices, manuals or any type of material to provide support to the tester in the test automation task. The categories of documentation identified for this topic were (Figure 15):

- **Wiki about automation:** 5 interviewees reported using documentation produced internally to guide how to use test frameworks, rules for creating variables for automated tests, and how to manage the repository containing the automated test codes. According to them, this documentation objective is to guide the testers newly admitted to the teams mainly. In addition, one respondent reported that there are weekly presentations on test automation given by more experienced

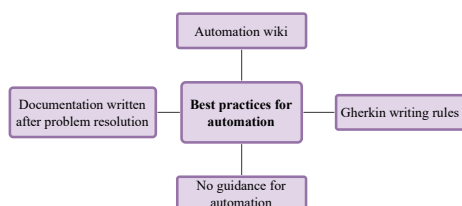


Figure 15. Best practices for test automation

professionals in order to share knowledge with other co-workers;

- **Writing rule for *Gherkin*:** according to 1 interviewee, the only good practice passed on to the test team was about the number of links in a test scenario in *Gherkin*, the which should not exceed more than five commands per test. However, there were no other supporting materials or rules for the tester;
- **Documentation written after problem resolution:** Given the lack of formal documentation available to testers, members of the testing area decided that, after resolving an automation related issue, the corresponding solution should be recorded on a platform with unrestricted access for all team members. In this way, a knowledge base would gradually be built within the company.;
- **No guidance for automation:** 6 respondents reported no good practices documented in their projects. If any additional guidance was needed, the tester should verbally contact another teammate or wait until the code review process of the test code, and the reviewer would indicate what should be changed in the created automation.

4.6 RQ5 - What are the main limitations in the automated GUI testing process?

As discussed in relation to manual GUI testing, this section presents the limitations that hinder respondents from creating automated GUI tests. These limitations were grouped into three main categories: (i) Limitations related to the software under test; (ii) Educational limitations; and (iii) Organizational limitations (Figure 16). These are detailed as follows:

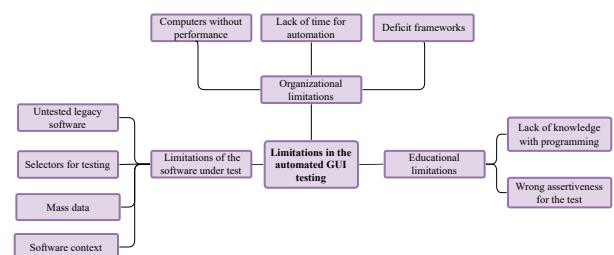


Figure 16. Difficulties that impact the GUI test automation process

1. Organizational Limitations

The Organizational limitations category, as well as for manual GUI tests, refers to problems generated from actions taken or lack of decision by the companies in which the testers work.

- **Computers without performance:** The productivity of automating GUI tests can be negatively impacted by the computers made available to employees. According to 4 interviewees, machines with few computational resources, such as memory and processing, do not support

automated test frameworks, making automation take longer. For the interviewees, it would be interesting to have newer computers available that meet the professional needs of testers;

- **Lack of time for automation:** The time available for creating automated scripts is insufficient. This makes the constant evolution of automated validations in the project challenging. This problem occurs due to the interruption of automation activities to assign professionals to tasks related to manual testing. As a solution, they indicate that the company should have better planning to have enough time for GUI test automation;
- **Deficit frameworks:** Two companies with a more conservative testing sector use old tools without the necessary tester functions to automate GUI tests. Thus, the tester can take extra time with library customizations or using other third-party solutions. One solution is using more modern tools that facilitate test automation, to the detriment of legacy frameworks that do not meet the demands of the area.

2. Educational Limitations

The Educational Limitations category mentions the respondents lack of technical knowledge to carry out automated GUI testing activities.

- **Lack of knowledge with programming:** 5 interviewees reported that, although their teammates know the test automation tools, some do not know enough of the programming languages or logic necessary to perform the tasks related to automation. This was mostly reported concerning customizing commands or setting up continuous integrations. The automated test professional needs at least a basic understanding of programming;
- **Wrong assertiveness for the test:** Due to the lack of experience of some testers, the test automation they produced may contain problems related to the objective of the test and the chosen assertiveness criteria. These criteria are used to determine whether a software assessment passes or fails. According to the 4 interviewees, one way to minimize this problem is to instruct less experienced professionals to transmit knowledge to the team.

3. Limitations of software under test

Difficulties involving the tested software and the environment in which it will be executed make up the categories of limitations of the software under test.

- **Untested legacy software:** Starting automation in untested software is challenging for respondents. In this context, it is necessary to

have time to prepare the test environment and its respective dependencies in order to execute the automated test scripts;

- **Selectors for testing:** In many software, there are no selectors for testing frameworks to interact with elements displayed in the interface. This can prevent automated tests from running on the project. This problem occurs when the developer does not add relevant selectors to the screen elements, such as the Front-End development library *React*¹⁰, which exports HTML pages without adequate structure for automated tests. One solution is for the developer to add information in the HTML software, such as *testing-id*, so that the tester can more easily automate software evaluations. Also, the tester should be able to create consistent *xpaths* if no *testing-id* is available;
- **Mass data:** As reported by six respondents who perform manual tests, the lack of mass data impacts the test automation process. According to them, simulating third-party data when creating tests can be challenging;
- **Software context:** Understanding a software domain can take time for a new tester, mainly if the project does not have documented requirements and business rules. It is not an easy task to automate software flows that nobody knows how it should work. One recommendation given by respondents is for the company to have software documentation available for testers. Also, if possible, the tester should perform manual software verification, so he has in mind what should be automated later.

5 Discussion

5.1 Overview of the interviewees

Analyzing the interview results, it was identified that most professionals who perform exclusively manual tests have between 2 and 4 years of experience, whereas testers who conduct both manual and automated GUI tests typically have between 5 and 10 years of experience.

Therefore, it is plausible that the length of professional experience influences how testers perform GUI tests. Automated GUI tests tend to be developed by professionals with longer tenure and greater seniority in the testing field, who have likely acquired sufficient knowledge throughout their careers to extend their testing activities beyond manual approaches.

Most testers involved in GUI testing hold higher education degrees. It was observed during the interviews that only one out of twenty respondents did not have a higher education degree. Among the degrees most commonly pursued by the

¹⁰<https://pt-br.reactjs.org/>

interviewees, Information Systems was the most frequently reported.

In addition, few respondents hold testing certifications, such as the Certified Tester Foundation Level (CTFL), despite the presence of a local ISTQB board in Brazil — the Brazilian Software Testing Qualifications Board (BSTQB). This board has translated study materials and exams into Brazilian Portuguese and allows the exam fee to be paid in the local currency (Real - R\$).

Based on the reports of the interviewed professionals, testers generally develop their skills to work in the testing field through gray literature, such as blogs and courses created by other testing professionals. Additionally, guidance from colleagues in the work environment and hands-on execution of testing tasks within software projects are the main learning methods. These findings can be largely explained by the way higher education technology courses address the subject of testing. As noted by the interviewees, in System Analysis and Development programs, students take several programming-related courses; however, there are no dedicated courses specifically focused on software testing, which compels those interested to seek supplementary testing material outside the formal educational institution.

The most commonly used process model in the software development projects of the interviewees is the Agile Model. Created in 2001 to respond more efficiently to the demands of software development, Agile has led many organizations to adopt agile methodologies in their projects.

As shown in the results chapter, many testers who perform GUI tests are assigned to projects exhibiting organizational and process deficiencies. These issues raise the following question: are the companies where these professionals work genuinely agile, or are they operating within a context of *false agile*? The term “**false agile**” refers to the adoption of agile practices without the necessary cultural transformation to achieve meaningful results through Agile methods.

The first principle of the Agile Manifesto¹¹, which contrasts with the limitations reported by testers, states: *Business people and developers should work together daily throughout the project*. Given that software requirements often change and such changes are not effectively communicated to all team members, it becomes evident that the team lacks adequate communication and that professionals from different areas do not collaborate synergistically in task execution.

5.2 Manual GUI testing process

Analyzing the objectives defined for manual GUI tests, it was observed that testers in their projects evaluate multiple aspects of the software during the same test run, such as verification of screen elements, responsiveness, and usability, in addition to the standard software functions. Given this context, it is possible to state that testers assume a polyvalent role within their teams regarding test execution tasks.

After defining the objectives of the GUI tests, testers use different test artifacts to create the scenarios to be executed later. It was observed that the development model used in the project impacts the type of input used by the tester. For ex-

ample, projects following the Waterfall development model provide traditional documentation such as requirements and business rules, while Agile projects provide user stories and acceptance criteria for defining manual GUI tests. Additionally, sprint planning ceremonies are used by testers to better understand what will be implemented in the software and, consequently, obtain more details about what will need to be tested in the future.

As mentioned, the development model also influences how test scenarios are created within projects. In Agile projects, many testers structure their tests using mind maps, which tend to be written faster than cursive writing depending on the complexity of what will be tested. Flexibility is another benefit of mind maps, as testers can edit and organize scenarios more dynamically by modifying the nodes of the mind map.

Regarding the motivation for choosing manual GUI testing, despite the growing adoption of automation techniques and tools, many testers still rely on manual checks of the software interface. Only a small proportion of testers perform GUI tests exclusively through automation, approximately 5%.

As previously shown, there are several reasons why testers run GUI tests exclusively manually. First, in many cases, the projects testers work on do not support the creation of automated GUI tests. Among the main reasons are: (i) short-term projects that do not justify the investment in automated GUI testing; and (ii) lack of organizational interest in training and allocating employees to automate tests. Management often values manual GUI testing more and overlooks the potential benefits of test automation.

Beyond organizational issues, there are challenges related to testers' educational backgrounds. Some professionals experience difficulties with programming logic activities and, therefore, do not effectively create automated scripts in their projects. It is likely that during their technical training, these testers did not achieve satisfactory proficiency in programming subjects, despite being required to attend courses involving coding, such as algorithm analysis and object-oriented programming. Two main reasons may explain this: (i) lack of effort on the part of the student during their studies, or (ii) shortcomings in teaching quality (or both). Due to these difficulties, two testers reported feeling “repulsed” (term used by the interviewee) by programming and consequently do not specialize in test automation.

5.3 Automated GUI testing process

Regarding the motivations that lead testers to engage in automated GUI testing, those who perform automated GUI tests tend to have more experience in the testing field than testers who work exclusively with manual GUI testing, which contributes to a more mature perspective of the industry. Their motivations for adopting automation include the pursuit of better compensation, organizational demands for automation skills during hiring, and project-related benefits such as reduced execution time and increased productivity. Additionally, interest in programming was frequently cited as a motivating factor, closely associated with testers' self-esteem, as automation enhances the perceived technical value of their

¹¹<https://agilemanifesto.org/>

work.

About the programming language to automation GUI test, Java was the most popular among the interviewed testers. Dynamically typed languages such as *Python* and *JavaScript* were also widely used, occupying the second and third positions, respectively. In addition to its popularity, *Java* is compatible with *Selenium*, one of the most widely implemented automation tools by testers. Furthermore, *Java* runs on any operating system and benefits from a very active community. Languages like *Python*, *JavaScript*, and *Ruby* were also frequently mentioned. These languages typically have a lower learning curve compared to others like *C*. They are compatible with various test automation tools such as *Selenium* and *Robot Framework*, and their adoption continues to grow throughout the industry.

Half of the testers who reported that the QA team or the testers themselves are responsible for selecting test automation tools work in Agile environments, where autonomy in decision making aligns with the principles of the development model. According to the Agile Manifesto, teams are encouraged to continuously reflect and adapt their processes, which reinforces the need for organizational support so that testers feel confident in choosing how to perform their tasks.

The results indicate that second generation GUI testing is the most widely adopted automation approach, mainly through tools such as *Selenium* and *Robot Framework*. *Selenium* stands out due to its open-source nature, cross-platform compatibility (*Windows*, *Linux*, and *Mac*), and strong community support, while *Robot Framework* is highlighted for enabling easy customization of automation scripts using *Python*. *Cypress* was also frequently mentioned for supporting GUI, component, and API test automation within a single codebase.

Notably, among the tools cited by the interviewees, only *Eye Automate* belongs to the third generation of GUI test automation, with all other mentions predominantly representing the second generation. This preference can be attributed to cultural factors, where software projects are already accustomed to these established frameworks, showing a lack of pursuit for more disruptive solutions. This trend is further intensified by the existence of significant legacy test code, which creates high demand and effort for migration to newer generation tools. Furthermore, second generation tools are often more commercially prominent and widely recognized as the industry standard, ensuring they remain the primary choice during the strategic definition of automation processes.

In addition, testers reported using frameworks such as *Cucumber*, *Capbara*, and *Behave*, which adopt *Gherkin* syntax to support Behavior-Driven Development (*BDD*). By enabling test scenarios written in an accessible language, *BDD* promotes collaboration among stakeholders and emphasizes alignment with client business needs.

It is important to highlight that the study did not focus on the motivations for using one generation of GUI testing tools over another; therefore, the interviewees were not asked directly about the reasons behind this choice.

5.4 Limitations on GUI testing

Many of the issues identified in the creation and execution of GUI tests are related to process and organizational failures. Despite the availability of best-practice guidelines in both formal and gray literature, deficiencies in requirements specification, incomplete documentation, and poor task planning persist, often leading to fragile requirements and poorly implemented software. As a consequence, the testing phase is frequently neglected or inadequately performed throughout the software development lifecycle.

Limitations affecting manual GUI testing are strongly influenced by organizational culture and workforce constraints. Overloaded teams and a shortage of testers often result in the prioritization of manual testing due to its immediacy, leaving little or no time for investment in automated GUI tests. In the long term, this scenario compromises test reproducibility and increases physical and mental fatigue, while also leading to the assignment of less experienced professionals to roles requiring specific expertise. Additionally, low team maturity, inadequate reporting practices, and the delivery of unstable software versions, sometimes without prior unit testing, further hinder the effectiveness of manual GUI testing. Common operational issues, such as unstable test environments and unavailable system integrations, also negatively impact testing activities, highlighting the need for early identification of dependencies and preconditions by project leaders.

Similar organizational limitations also affect automated GUI testing. Testers frequently work with outdated hardware that fails to meet the performance requirements of automation tools, increasing the time required to develop automated scripts. Participants also reported resistance from organizations to updating or changing automation frameworks, which prevents the adoption of more efficient solutions. Furthermore, the lack of testing identifiers in HTML elements complicates tool interaction with the system under test. Finally, the absence of formal documentation on automation best practices—reported by approximately 46% of the respondents—forces testers themselves to initiate knowledge-sharing efforts, rather than benefiting from structured management-driven support.

The limitations identified in this study regarding the creation and execution of GUI tests reflect broader trends observed in the global software industry. Although the primary focus of this research is the Brazilian industry, many of the challenges identified by the participants align with findings from other countries, which suggests that the results have a high degree of transferability. For instance, the difficulties with the maintenance of scripts and the technical limitations of tools reported by our participants echo the findings of previous studies conducted in Italy by Amalfitano et al. (2026) and Coppola et al. (2020), also in Turkey by Alégroth et al. (2015).

This alignment suggests that the core obstacles in GUI testing, such as the high effort required for automation and the gap in programming skills among testers, are persistent issues across different software engineering contexts globally. Therefore, the patterns observed in the Brazilian context regarding environmental instability and the lack of test data are not isolated cases but represent common industrial problems

that transcend geographic boundaries.

These findings imply that the persistent reliance on manual GUI testing and older automation frameworks is not merely a consequence of individual skill gaps, but a systemic issue rooted in organizational maturity. When companies operate in a “false agile” context—prioritizing short-term feature delivery over sustainable engineering practices like testability (e.g., adding testing IDs) and environment stability—they inadvertently force testers into reactive, manual workflows. This suggests that advancing GUI testing practices in the industry requires more than just technical training for testers; it demands a cultural shift in project management to integrate quality assurance as a foundational element of the development lifecycle rather than an afterthought.

5.5 Final findings and recommendations

Based on quotes from research participants, it was possible to identify the contexts in which testers work professionally and the respective difficulties they have to deal with in their day-to-day work. Some suggestions to improve GUI test execution are:

- **Testers’ professional background and knowledge sharing:** Although many testers hold technology degrees, they frequently encounter difficulties in identifying test scenarios and coding automated scripts due to gaps in programming logic. Given that 40% of work-based learning occurs through tacit knowledge transfer between colleagues and that friend recommendations are a primary entry point into the field, organizations should prioritize internal mentorship programs and Pair Testing over generic external training. Since 55% of online learning comes from self-paced courses (e.g., Udemy), fostering Communities of Practice (CoPs), where practitioners share insights from these courses and collectively resolve automation roadblocks, can be a more effective strategy for upskilling. This approach leverages the existing culture of informal knowledge acquisition to bridge technical gaps in a context-sensitive manner.
- **Software testability and technical infrastructure:** The study identified low software testability as a major roadblock, characterized by application instability, resource-depleted environments, and a lack of test data, the latter being the “biggest complaint” among interviewees regarding manual execution. To move beyond generic planning, teams should adopt concrete engineering practices identified in the results: to facilitate tool interaction, developers should incorporate unique testing-IDs or consistent XPath paths into HTML elements. Additionally, mitigating environmental instability requires adopting Docker for dynamic maintenance and integrating DevOps professionals to oversee infrastructure and access. These measures convert abstract testing goals into concrete technical requirements, effectively stabilizing the testing cycle.
- **Process used in software development:** Regardless of the development model used in companies, distinct limitations were identified in how software development projects are managed. Failures such as lack of task planning, inefficient team communication, emphasis on activities with little value, overloaded teams, and conflicting environments were some examples raised by the testers during the research. As Waterfall, Incremental and Agile methods discuss solutions to these problems, it is recommended that those responsible for the teams review at what point the execution of the project distanced itself from the best practices adopted by the methods that support software development activities and then evaluate which measures can be implemented in the short and medium term to improve the health of the project;
- **Management practices and organizational culture:** The research identified significant management failures that impact the health of projects, revealing a context of false agile where practices are adopted without the necessary transformation in culture. Participants reported that inefficient communication regarding changes in specifications, such as updates to prototypes in Figma that are not shared with the quality team, leads to constant rework and wasted effort. Furthermore, the culture of guilt identified in some teams creates a toxic environment that demotivates professionals instead of fostering solutions based on collaboration. To address these issues, managers should move beyond a superficial use of methodologies and prioritize synergistic work between business people and developers to ensure that the execution of the project aligns with the core principles of the chosen development model.
- **Investment in GUI test automation:** The test automation activity requires several elements to be carried out, such as development skills on the part of the tester, use of tools, and additional implementation costs. However, the benefits of automated GUI testing can be more significant. These range from reducing errors generated by humans to the possibility of running more tests with each new version of the software. Even with the mentioned advantages, many companies have no interest in using this approach in their projects since testers stated that their organizations had a greater incentive to perform manual tests, lack of time in projects for automating GUI tests, and provision of old computers for coding test scripts. Keeping only manual GUI tests running can become technical debt for the project. Thus, it is recommended that companies evaluate the possibility of at least automating the most valuable test scenarios for the software to minimize the team’s effort.

6 Threats to Validity

The threats to validity identified during the interviews are:

- **Construct Validity:** The questions formulated for the interviews were designed to enable data collection from various aspects of GUI testing. However, it was observed that some interviewees answers deviated from the topics questioned; this possibly occurred due to a lack of understanding of the questions by the interviewees. Therefore, answers unrelated to the questions were disregarded. Furthermore, we acknowledge a limitation regarding our analytical framework. Because we adopted a broad operational definition of GUI testing based on industry jargon, our open and axial coding did not separate accounts of strictly interface-level validation from system-level functional checks executed through the interface. This conflation during the coding phase reflects a construct validity limitation, as the analysis treats both concepts under the single umbrella of GUI testing;
- **Internal Validity:** Due to difficulties in recruiting testers willing to participate in the interviews, there was an unequal distribution of interviewee profiles; that is, the numbers of participants performing only manual GUI tests, only automated GUI tests, and both types of tests were not balanced. Furthermore, data collection was halted based on the exhaustion of available participants within the research timeframe rather than a theoretical stopping criterion, which limits the breadth of our qualitative sample;
- **External Validity:** The sample of 20 participants does not represent the entire global population of testers. Nevertheless, the results offer significant transferability as they provide a detailed look into challenges that are common in the software industry regardless of geography. The convergence of our findings with existing international literature regarding tool limitations and the maintenance of automated suites suggests that the identified patterns are relevant beyond the Brazilian context. Future studies could perform cross national comparisons to further explore how cultural and economic factors influence these testing practices
- **Reliability:** To ensure the quality of the results, two researchers coded the interview transcripts and compared the extracted codes to resolve discrepancies in the preliminary results. Subsequently, two other researchers reviewed the categories and their respective codifications to minimize biases or misinterpretations of the data obtained from the interviews.

7 Conclusions

This research contributes to a broader understanding of industry practices for GUI testing based on interviews with 20 professionals from the Brazilian software industry. The findings highlight that manual GUI testing is mainly driven by difficulties with programming related tasks and personal preference, as well as by organizational challenges such as insufficient staffing, inadequate hardware resources, and ineffective team communication.

In contrast, testers engaged in automated GUI testing tend to have greater professional experience, motivated by factors such as career advancement, improved compensation, reduced testing time, and interest in programming. The most frequently cited technologies for GUI test automation were the *Java* programming language and the *Selenium* tool.

This research contributes to the field of software testing, particularly GUI testing, by reporting empirical evidence from testers in the Brazilian software industry. The main contributions are:

1. Presentation of the profile of software testers in the industry, including educational background, sources of knowledge, and motivations for working in software testing;
2. Description of how testers perform manual and automated GUI tests;
3. Discussion of aspects related to the manual GUI testing process, including test objectives and support materials used to create test scenarios;
4. Discussion of aspects related to both manual and automated GUI testing processes, such as the programming languages and tools most adopted by testers who create automated GUI tests;
5. Identification of the main limitations impacting the activities of software testers in both manual and automated approaches;
6. Exploration of motivations influencing testers' choices between manual and automated GUI testing approaches.

Industry professionals can leverage the findings of this study to improve their GUI testing processes, particularly by addressing the limitations reported by testers in both manual and automated approaches. Furthermore, academia may develop solutions to alleviate the challenges faced by testers and facilitate their daily activities in software development projects.

Future work will be developed considering the results of this research, focusing on:

- Encourage formal education institutions to incorporate software testing topics into undergraduate curricula;
- Investigate why software testers lack programming skills despite having attended programming logic courses during their undergraduate studies;
- Develop guidelines to assist testers who currently perform only manual GUI tests in transitioning to automated testing.
- Investigate the reasons why a tester chooses one tool over another, especially in cases where the solutions are from different generations of tools.

- Verify the potential and limitations of third-generation GUI test automation solutions.

In addition to the future ideas mentioned, regarding GUI test automation, we should propose a deeper investigation into the adoption barriers for 3rd Generation tools to understand why the software industry still heavily relies on 2nd Generation solutions.

8 Dataset Availability

To foster transparency and support the open science movement, all research artifacts are organized in a structured repository on Zenodo¹².

The dataset is partitioned into three main files:

- Protocol: containing the semi-structured interview script and the recruitment templates;
- Legal: including the anonymized Informed Consent Form;
- Transcripts: which provides the full textual data in Brazilian Portuguese.

To comply with the Brazilian General Data Protection Law (LGPD), we performed a rigorous de-identification process: all specific mentions of company names, proprietary software, and individual identities. While the raw coding process was conducted on the Delve platform, we have exported a synthesized version of the axial coding as a spreadsheet within the repository to ensure replicability without compromising participants' privacy.

References

- 25000, S. I. (2011). Iso/iec 25010. <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?start=6>. Last checked on Sep 08, 2021.
- Adams, W. C. (2015). *Conducting Semi-Structured Interviews*, chapter 19, pages 492–505. John Wiley Sons, Ltd.
- Alégroth, E., Feldt, R., and Olsson, H. H. (2013). Transitioning manual system test suites to automated testing: An industrial case study. In *Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST'13*, pages 56–65. IEEE Computer Society.
- Alégroth, E., Feldt, R., and Ryrholm, L. (2015). Visual gui testing in practice: challenges, problems and limitations. *Empirical Softw. Engg.*, 20(3):694–744.
- Alégroth, E., Feldt, R., and Ryrholm, L. (2015). Visual GUI testing in practice: challenges, problems and limitations. *Empir. Softw. Eng.*, 20(3):694–744.
- Alegroth, E., Gao, Z., Oliveira, R., and Memon, A. (2015). Conceptualization and evaluation of component-based testing unified with visual gui testing: An empirical study. In *IEEE 8th International Conference on Software Testing, Verification and Validation, ICST'15*.
- Alferidah, S. K. and Ahmed, S. (2020). Automated software testing tools. In *2020 International Conference on Computing and Information Technology (ICCIT-1441)*, pages 1–4.
- AltexSoft (2023). Acceptance criteria for user stories: Purposes, formats, examples, and best practices. Online article.
- Alégroth, E., Feldt, R., and Kolström, P. (2016). Maintenance of automated test suites in industry: An empirical study on visual gui testing. *Information and Software Technology*, 73:66–80.
- Alégroth, E., Petersén, E., and Tinnerholm, J. (2021). A failed attempt at creating guidelines for visual gui testing: An industrial case study. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 340–350.
- Amalfitano, D., Coppola, R., Distant, D., and Ricca, F. (2026). Ai in gui-based testing: A survey of techniques, tools, and perceived advantages and limitations. *Journal of Systems and Software*, 235:112751.
- Amalfitano, D., Fasolino, A. R., Tramontana, P., De Carmine, S., and Memon, A. M. (2012). Using gui ripping for automated testing of android applications. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012*, page 258–261, New York, NY, USA.
- Amalfitano, D., Júnior, M., Fasolino, A. R., and Delamaro, M. (2025). A gui-based metamorphic testing technique for detecting authentication vulnerabilities in android mobile apps. *Journal of Systems and Software*, 224:112364.
- Bach, T. (2022). Testing in very large software projects. Master's thesis, The Faculty of Mathematics and Computer Science, <https://archiv.ub.uni-heidelberg.de/volltextserver/31757/7/dissertation-tbach-color-druck.pdf>.
- Banerjee, I., Nguyen, B., Garousi, V., and Memon, A. (2013). Graphical user interface (gui) testing: Systematic mapping and repository. *Information and Software Technology*, 55(10):1679–1694.
- Canny, A., Palanque, P., and Navarre, D. (2020). Model-Based Testing of GUI Applications Featuring Dynamic Instantiation of Widgets. In *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2020)*, pages 95–104, Porto (virtual), Portugal. IEEE.
- Coppola, R., Ardito, L., Morisio, M., and Torchiano, M. (2020). Mobile testing: New challenges and perceived difficulties from developers of the italian industry. *IT Professional*, 22(5):32–39.
- Corbin, J. and Strauss, A. (2008). Basics of qualitative research (3rd ed.): Techniques and procedures for developing grounded theory.
- Cui, C., Li, T., Wang, J., Chen, C., Towey, D., and Huang, R. (2024). Large language models for mobile gui text input generation: An empirical study. *arXiv preprint arXiv:2404.08948*.
- Dallal, J. A. (2009). Automation of object-oriented framework application testing. In *2009 5th IEEE GCC Conference Exhibition*, pages 1–5. IEEE Computer Society.

¹²<https://doi.org/10.5281/zenodo.8231172>

- Dias-Neto, A. C., Matalonga, S., Solari, M., Robiolo, G., and Travassos, G. H. (2017). Toward the characterization of software testing practices in south america: looking at brazil and uruguay. *Softw. Qual. J.*, 25(4):1145–1183.
- Garousi, V., Rainer, A., Lauvås, P., and Arcuri, A. (2020). Software-testing education: A systematic literature mapping. *Journal of Systems and Software*, 165:110570.
- Grechanik, M., Xie, Q., and Fu, C. (2009). Creating gui testing tools using accessibility technologies. In *International Conference on Software Testing, Verification, and Validation Workshops*, pages 243–250. IEEE Computer Society.
- Jones, C. (2015). Wastage: The impact of poor quality on software economics. *Software Quality Professional*, 18(1).
- Junior, N., Costa, H., Karita, L., Machado, I., and Soares, L. (2021). Experiences and practices in gui functional testing: A software practitioners' view. In *Proceedings of the XXXV Brazilian Symposium on Software Engineering, SBES '21*, page 195–204, New York, NY, USA. Association for Computing Machinery.
- Karhu, K., Repo, T., Taipale, O., and Smolander, K. (2009). Empirical observations on software testing automation. In *Second International Conference on Software Testing Verification and Validation, ICST 2009, Denver, Colorado, USA, April 1-4, 2009*, pages 201–209. IEEE Computer Society.
- Kertusha, I., Assess, G., Duman, O., and Arcuri, A. (2025). A survey on web testing: On the rise of ai and applications in industry. *arXiv preprint arXiv:2503.05378*.
- Klaus Olsen, M. P. and Ulrich, S. (2019). *Certified Tester Foundation Level Syllabus*. International Software Testing Qualifications Board, https://bstqb.org.br/b9/doc/syllabus_ctfl_2018br.pdf, third edition.
- Leitner, A., Ciupa, I., Meyer, B., and Howard, M. (2007). Reconciling manual and automated testing: The autotest experience. In *40th Hawaii International International Conference on Systems Science (HICSS-40 2007), CD-ROM / Abstracts Proceedings, 3-6 January 2007, Waikoloa, Big Island, HI, USA*, page 261. IEEE Computer Society.
- Melo, S. M., Santos, Í., Souza, P. S. L., and Souza, S. R. S. (2022). A survey on the practices of software testing: a look into brazilian companies. *J. Softw. Eng. Res. Dev.*, 10:11:1–11:15.
- Molyneaux, I. (2015). *The Art of Application Performance Testing: From Strategy to Tools*. O'Reilly Media, second edition.
- Polepalle, C., Kondoju, R. S., and Badampudi, D. (2022). Evidence and perceptions on gui test automation - an exploratory study. In *Proceedings of the 15th Innovations in Software Engineering Conference, ISEC '22*, New York, NY, USA. Association for Computing Machinery.
- Qian, J., Lv, G., Jin, Y., Shang, Z., Yan, S., Wang, Y., and Chen, L. (2025). Robotic visual gui testing for truly non-intrusive test automation of touch screen applications. *IEEE Transactions on Software Engineering*, 51(4):1205–1231.
- Ramler, R. and Wolfmaier, K. (2006). Economic perspectives in test automation: Balancing automated and manual testing with opportunity cost. In *Proceedings of the 2006 International Workshop on Automation of Software Test, AST '06*, page 85–91, New York, NY, USA. Association for Computing Machinery.
- Rothermel, G., Untch, R., Chu, C., and Harrold, M. (2001). Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering*, 27(10):929–948.
- Sánchez-Gordón, M., Rijal, L., and Colomo-Palacios, R. (2020). Beyond technical skills in software testing: Automated versus manual testing. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW'20*, page 161–164, New York, NY, USA. Association for Computing Machinery.
- Sharma, R. M. (2014). Quantitative analysis of automation and manual testing. Technical Report Volume 4, International Journal of Engineering and Innovative Technology (IJEIT).
- Shruti Malve, P. S. (2017). Investigation of manual and automation testing using assorted approaches. *International Journal of Scientific Research in Computer Science and Engineering*, 5:81–87.
- Tanaka, H. (2019). X-BROT: prototyping of compatibility testing tool for web application based on document analysis technology. In *Workshop on Industrial Applications of Document Analysis and Recognition, WIADAR@ICDAR 2019, Sydney, Australia, September 22-25, 2019*, pages 18–21. IEEE.
- Tramontana, P., Amalfitano, D., Amatucci, N., and Fasolino, A. R. (2019). Automated functional testing of mobile applications: a systematic mapping study. *Softw. Qual. J.*, 27(1):149–201.
- Yarifard, A. A., Araban, S., Paydar, S., Garousi, V., Morisio, M., and Coppola, R. (2025). Extraction and empirical evaluation of gui-level invariants as gui oracles in mobile app testing. *Information and Software Technology*, 177:107531.
- Ye, J., Chen, K., Xie, X., Ma, L., Huang, R., Chen, Y., Xue, Y., and Zhao, J. (2021). An empirical study of gui widget detection for industrial mobile games. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2021*, page 1427–1437, New York, NY, USA. Association for Computing Machinery.
- Yu, S., Fang, C., Tuo, Z., Zhang, Q., Chen, C., Chen, Z., and Su, Z. (2023). Vision-based mobile app gui testing: A survey. *arXiv preprint arXiv:2310.13518*.
- Zein, S., Salleh, N., and Grundy, J. (2016). A systematic mapping study of mobile application testing techniques. *Journal of Systems and Software*, 117:334–356.