Experiences and Practices in GUI Functional Testing: A Software Practitioners' View

Nailton Junior
nailtonalmeidajr@gmail.com
State University of Feira de Santana
Feira de Santana, BA, Brazil

Heitor Costa
heitor@ufla.br
Federal University of Lavras
Lavras, MG, Brazil

Leila Karita
leila.karita@ufba.br
Federal University of Bahia
Salvador, BA, Brazil

Ivan Machado
ivan.machado@ufba.br
Federal University of Bahia
Salvador, BA, Brazil

Larissa Rocha Soares
lrsoares@uefs.br
State University of Feira de Santana
Feira de Santana, BA, Brazil

ABSTRACT

Software testing is an important activity to ensure software quality. A widely performed type of testing is GUI (Graphical User Interface) functional testing, where the aim is to evaluate the software functionality through their graphic interface. We can find in the literature techniques and solutions for handling GUI functional testing in the last years. However, studies that deeply analyze how professionals in the industry perform GUI functional testing are still lacking. In this study, we investigated how testing professionals perform GUI functional testing. We surveyed 222 professionals from different positions and roles who perform those tests. The results indicate that many professionals perform GUI functional testing without proper tool support. They also pointed out that GUI test automation tools have limitations. This study presents preliminary results to understand the Brazilian companies' scenario on the difficulties of GUI functional testing.

KEYWORDS

Software testing, GUI funcional testing, surveys, software testers

ACM Reference Format:
Nailton Junior, Heitor Costa, Leila Karita, Ivan Machado, and Larissa Rocha Soares. 2021. Experiences and Practices in GUI Functional Testing: A Software Practitioners' View. In *Brazilian Symposium on Software Engineering (SBES '21), September 27-October 1, 2021, Joinville, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3474624.3474640

1 INTRODUCTION

Software quality is related to the degree to which the system satisfies its various stakeholders' stated and implied needs and, therefore, provides value [37]. The lack of quality can cause, for example, financial losses, friction between suppliers, and lawsuits for non-compliance. According to the International Software Testing Qualifications Board, one way to avoid those problems is to carry out the software testing process, which aims to ensure the software quality and also reduce the risk of failures in the software in operation [23].

Testers must verify whether the software functionality meets the requirements specification and, more importantly, the users' needs during the software testing process. There are two main types of analysis: (i) static analysis checks the source code or any other project artifact (e.g., requirements specification, user history, and flowcharts) looking for logical errors that could be the cause of future system failures; and (ii) dynamic analysis perform tests while the software is running, searching for functional failures.

GUI (Graphical User Interface) functional testing is a dynamic test that evaluates systems with graphical interfaces [22]. It is concerned about verifying the events performed on the software screen, such as the mouse click on a button, filling in a text field, or using a drop-down menu. GUI testing is a widely used strategy and suitable for applications from several domains and platforms, such as mobile devices and personal computers [6].

GUI testing counts on manual and automated approaches [8]. Manual testing is an essential part of the software testing process. A tester can perform manual testing to test every aspect of the software – be it functional or non-functional [32]. Any new application should be manually tested before its testing can be automated [34]. However, manual testing could be ineffective when employed exclusively due to human error-proneness. Besides, it can be both highly costly and time-consuming [3]. Manual testing completeness is hard to reach in software with complex graphical interfaces [15]. For example, the tester can non-execute the necessary tests for the different ways of a software function; manual testing can non-reach all those ways, resulting in failures in the production environment.

On the other hand, automated testing analyzes the software without the direct supervision of a tester, and its execution might be faster and more reliable than manual testing because there is minimal human intervention in the process [7, 31]. However, testers do not get to automate all tests [13, 21]; therefore, manual testing is still necessary. In particular, automated GUI testing is not a trivial activity as it requires knowledge of programming languages and a clear definition of what will be automated [36, 38].

Recent studies have proposed tools and techniques to perform manual and automated GUI testing. For example, Banerjee et al. [11] performed a systematic mapping study to understand the GUI functional testing state-of-the-art, both for desktop and mobile

SBES '21, September 27-October 1, 2021, Joinville, Brazil

ACM ISBN 978-1-4503-9061-3/21/09…$15.00
https://doi.org/10.1145/3474624.3474640

applications. Mohammad et al. [28] evaluated tools to generate automated testing for the mobile apps through nine quality attributes. Gunasekaran and Bargavi [19] compared the performance of tools to automate GUI testing for Android and IOS. The authors still recommended the use of specific tools (MonkeyTalk and UIAutomator). Ali et al. [5] proposed a new testing GUI process, automating the creation and execution of GUI test cases.

Although most studies propose techniques to test or evaluate the performance of existing tools and frameworks, few studies comprehensively understand the practices and challenges of performing GUI testing in the software industry. Given this context, this paper aims to identify how industry professionals are performing GUI testing. For this, in this study, we surveyed 222 software professionals to understand how they handle the verification of the GUI software. We asked them about their experience in software testing, such as manual GUI testing techniques, and what they use to evaluate new functionalities and releases. Besides, we looked to understand which solutions they adopt to the automated GUI testing and the limitations of the available solutions.

The study yielded fascinating insights and perspectives. We identified that manual GUI testing is still a common practice. When automating GUI testing, Java is the preferred programming language. However, we could observe that the number of software professionals using dynamically typed languages, such as Python, Ruby, and Javascript, is growing. Also, even stating that the tools used to generate automated GUI testing have limitations, they agree that those tools meet their needs. In summary, our contributions are:

(1) We conducted a quantitative study that showed the characteristics of the software professionals who perform GUI tests. We could draw the software professionals' profiles based on academic training, professional experience, and expertise in the industry;
(2) We validated that the execution of GUI testing is still strongly carried out manually. This result can encourage the development of solutions to create automated testing;
(3) We highlighted the software professionals' main computational tools to create automated GUI testing and their respective limitations.

The remainder of this work is organized in the following structure. Section 2 presents concepts regarding manual and automated GUI functional tests. Section 3 details the research methodology of the study. Section 4 describes the results obtained for the research questions. Section 5 analyzes and discusses gathered data. Section 6 addresses threats to validity. Section 7 presents the related work. Section 8 draws concluding remarks.

2 BACKGROUND

The graphical interface of a system is one of the main objects of the tests. GUI tests simulate how the system will be used in its production version, considering that a large part of the end-user interactions is done through the graphical interface [30]. Similar to other tests, practitioners must insert inputs (data) to execute the tests. Next, they need to analyze the conformity between the test outputs and the results previously defined in the requirements specification [27].

A software tester commonly performs manual GUI testing by interacting, analyzing, and reporting the results to the project team. According to Shruti Malve [36], manual testing has a low implementation cost since the tester only needs the system to be evaluated and a device (mobile, desktop, or tablet) accessing it. Nevertheless, manual testing has disadvantages; for example, (i) it consumes a lot of Information Technology (IT) professional's time if there are several test cases to run, impacting the testing activity performance [2]; (ii) it is likely to be an error-prone strategy, as it counts on humans to carry out [18]; and (iii) it could become tedious due to the repetitive executions [25].

Due to manual testing weaknesses, some studies developed different automated solutions for GUI testing [36]. Automated GUI testing allows faster system validation, reducing the required time to evaluate the system under test [21, 29]; enables test reuse and cross-platform executions [14]; and increases test coverage [33, 35]. Functional automated GUI testing has undergone several changes, and the literature organizes it into three generations [1], as follows:

In the *first* generation, the interaction is with the screen elements from their specific position on the interface (x- and y-axis) [17]. It was a big step to complement manual testing and reduce the effort of testing teams. However, the software screen resizing harmed that interaction as it prevented the test scripts from identifying the element to be used.

The *second* generation proposed another solution, where the interaction with the software parts occurs through locators associated with each screen element, such as the field name in the HTML (HyperText Markup Language) code or the CSS (Cascading Style Sheets) selectors [4]. Although that interaction can solve the problem mentioned in the first generation, it might bring complexities. For example, when programming the test scripts, the tester must keep in mind the different browsers and devices and singular configurations that each test needs to execute.

The *third* generation, also known as the Visual GUI Testing (VGT), is associated with image recognition [3]. The automation performed by the tester interacts with the software elements through the identification of the screen elements, such as the shortcut icon and "back" button. Problems like the non-recognition of visual elements and the test execution performance are issues that the tester must consider.

The number of proposed tools to automate GUI testing has led the research community to carry out empirical evaluations to investigate both commercial and open-source tools, as well as to propose new frameworks to automate systems that pursue graphical interfaces [38]. A study worth mentioning is the one of Alegroth et al. [4], which ranked 54 tools by their popularity – considering metrics like the number of results returned by Google and the number of page views on Wikipedia about a given tool. The authors concluded that the existing tools do not encompass all the functionalities a tester needs, such as the Record/Replay function and support for CI/CD (continuous integration/continuous delivery) plugins.

In addition, Garousi et al. [16] investigated the best tools for automating third-generation tests (VGT). The authors compared the *JAutomate* and *Sikuli* tools in two industry-specific cutouts. As a result, the *JAutomate* obtained better marks in the tests of robustness, repeatability, and effort for the development of the tests. Arnatovich and Wang [10] conducted a systematic review aiming

to provide a broad overview of the tools for automating mobile GUI tests presented in academic papers. The research identified 23 different frameworks. Using efficiency and effectiveness metrics, the authors concluded that some current automated testing tools could not be extensively used in practice, considering that they did not meet the quality criteria defined in the review. Another finding identified is that the identified solutions usually require at least 30 minutes for implementation.

3 METHODOLOGY

This study aims to understand the current scenario of GUI functional testing from an industry perspective. We defined the following main research question: **How do practitioners perform GUI functional testing?** In order to systematically answer this question, the main research question was split into the following sub-questions:

RQ1- How do professionals perform manual GUI testing? This question aims to unveil commonly employed practices on how software professionals plan and perform manual GUI testing.

RQ2- How do professionals perform automated GUI testing? This question seeks to identify the programming languages and support tools used to automate GUI testing. In addition, to determine when the software professionals perform automated GUI testing and the share of automated test cases.

RQ3- Which tools do participants commonly use to report project issues and bugs? This question seeks to identify how the software professionals report the bugs found by the team.

We next report the process to create the survey, including the procedures to define the target audience, the pilot test execution, the survey distribution, and the strategies we employed to analyze gathered data.

3.1 Target Audience

We only considered software professionals who perform software testing activities. At the beginning of the survey, we included a control question to know whether the respondent performs or has already performed functional GUI testing.

3.2 Survey Design

The survey comprised 32 questions, 27 mandatory (84%) and 5 optional (16%) questions, as Table 1 shows. Additionally, the survey had 25 closed-ended and 7 open-ended questions (Q6, Q8, Q23, Q26, Q27, Q31, Q32) grouped into four sections:

- **Respondents characterization.** This section has one open-ended and four closed-ended questions on participants' demographics. We also raised their educational level and the programming languages they know;
- **Companies characterization.** This section has one open-ended and six closed-ended questions to identify the software professionals' labor scenario. We intend to collect information on the workplace, software development model, individuals responsible for tests, and software development platforms compatible with the testing execution;
- **Experience with manual GUI functional testing.** This section has four closed-ended questions to investigate the software professionals' experience and how they run the tests and report the identified bugs;
- **Experience with automated GUI functional testing.** This section has three open-ended and nine closed-ended questions. We aimed to gather software professionals' experiences with automated GUI functional testing, tools used in the testing automation, individuals responsible for determining the use of such tools, programming languages with which they develop tests, and testing framework limitations.

3.3 Pilot survey

We conducted a pilot test round to evaluate the survey quality. We sent the survey to software testing professionals. On March 8 and 9, 2021, four participants answered the survey and suggested some changes regarding the wording of the questions. We followed their advice and fixed the writing to make the survey questions clearer.

We also asked the participants to inform the amount of time the survey took them. We identified the average time required to complete the survey (ten minutes) from those responses. We reported that time to the participants when delivering the form.

3.4 Survey Distribution

We used the Google Forms platform[1] to create the survey. Next, we made it available on different social networking platforms:

- *LinkedIn*: direct message to 675 software professionals and publication to 8 groups focused on TI;
- *Email*: message to 100 companies;
- *Facebook*: publication of messages to 8 groups focused on IT and education.

We presented a brief contextualization regarding the research topic for all these communication channels, the study objective, and its importance. The survey was available for one month (March 10 to April 9, 2021).

3.5 Data Analysis and Reporting

We carried out one quantitative-qualitative study. We adopted some criteria and procedures for collecting data to identify and understand the preferences and behaviors of the participants. We adopted four assumptions in the survey to achieve the defined objectives:

(1) We applied single closed-ended questions with a range of values to characterize the target audience. When discussing the results, the sum of percentages is equal to a hundred percent;
(2) We applied open-ended questions. The sum of some questions percentages was superior to 100% when we discussed

[1] https://www.google.com/forms

Table 1: Survey Questions

#	RQ†	Question Description
Q1	G	Do you agree with the consent form?
Q2	G	Do you do or have you ever done GUI functional testing?
Q3	D	In which state of the country do you live?
Q4	D	What is your academic background?
Q5	D	If you have a graduate degree, which one have you attended?
Q6	D	Do you have any certifications in the area of software testing?
Q7	D	Besides testing software, do you develop? In which programming language?
Q8	D	What position or function do you currently hold in the company you work for?
Q9	D	Which area your company's software serves
Q10	D	Which software development model is currently used in your company?
Q11	D	In your current professional experience, who is responsible for GUI functional testing?
Q12	RQ1	In your current professional experience, what is checked when performing functional GUI testing?
Q13	D	On which platform do the GUI functional tests run?
Q14	D	How do you currently perform functional interface (GUI) testing?
Q15	D	How many years of experience do you have in GUI functional testing?
Q16	RQ1	What action do you perform in GUI functional testing?
Q17	RQ1	How do you manually run the GUI tests?
Q18	RQ3	How are the bugs identified when performing GUI functional tests reported?
Q19	D	How many years of experience do you have in automated GUI functional testing?
Q20	RQ2	In which programming language do you automate GUI functional tests?
Q21	RQ2	In your current professional experience, which tools are used in GUI functional test automation?
Q22	RQ2	Besides the tools mentioned above, what other frameworks are used for GUI functional test automation?
Q23	RQ2	Who proposed the tools used for GUI functional test automation in your company?
Q24	RQ2	How often do you run automated GUI functional tests in your company?
Q25	RQ2	Of all the functional GUI tests performed, what proportion do you believe is done in an automated way?
Q26	RQ2	What are the biggest limitations of the tools you use for GUI functional test automation? Comment on:
Q27	RQ2	Do you think the tools used for GUI functional test automation meet your need? Comment on:
Q28	RQ3	How are the bugs identified in the automated execution of the GUI functional tests reported?
Q29	RQ2	Besides GUI functional tests, what other tests do you perform?
Q30	D	Besides functional GUI test, what other software testing is performed in your company?
Q31	G	Do you have any comments about the survey topic or the questions on the form?
Q32	G	If you are interested in receiving the results of this survey or are available to answer future questionnaires, please inform your e-mail.

† D = Demography Question, G = General Question, RQ = Research Question

the results because the participants could answer in more than one option (alternative);

(3) We conducted one open coding rigorous process for the open-ended questions. Two authors independently extracted the general concepts from the answers and discussed the divergences to validate the coding;

(4) We selected excerpts from the open-ended questions answers to support the discussion of the results. We used a unique identifier for respondents' each of the quotes. For example, #R1 indicates the first respondent's answers.

Table 2: Participant Testing Certifications

Test Certification	#Certified Testers
CTFL	58
CTFL-AT	17
CTAL-TAE	7
CTAL-TM	5
CTAL-TA	3
CBTS	3
CTFL-TM	1
CTFL-PT	1
CTFL-MBT	1
CTAL-TTA	1
CTAL-SEC	1
CTAL-AT	1
BSTQB	1

4 RESULTS

4.1 Respondents' profile

We shared the survey with Brazilian IT companies and software professionals and received responses from almost every Brazilian state. Among them, we could highlight São Paulo (71 respondents - 32%), Rio Grande do Sul (40 respondents - 18%), and Paraná (27 respondents - 12%). Out of the respondents, 11 have high school education (5%), 2 have technical education (1%), and 209 hold a Bachelor's degree (94%). Of those with Bachelor's degrees, 57 hold a graduate degree (28%), where 4 have a Master's degree (2%). Moreover, 53 respondents have got certifications in one course after concluding their Bachelor's degree[2] (26%).

Regarding the respondents' education, 73 hold a degree in Systems Analysis and Development (36%), 56 respondents in Information Systems (27%), and 27 respondents have a degree in Computer Science (13%). Moreover, 28 respondents (14%) hold degrees in other technology-related courses (e.g., IT Management and Information Security), and 21 respondents in courses not associated with IT (10%) (e.g., Chemical Engineering and Physical Education).

In addition, 61 respondents (27%) are certified testers, as Table 2 shows. Among them, 20 respondents (9%) have more than one software tester certification. The most mentioned certifications were CTFL and CTFL-AT, 58 (95 %) and 17 (28%) respondents, respectively. It is worth remarking that CTFL certification is a prerequisite for obtaining other more advanced ISTQB certifications[3].

Concerning the role respondents play in their companies, we identified that 134 work as Test Analysts (60.6%), 24 work as Quality Engineers (10.9%), 19 work as Managers (8.6%), 16 work as Developers (7.2%), and 17 respondents have other roles (12.7%), such as Support Analyst, Infrastructure Analyst, and Data Analyst. We also asked whether they use some programming language to develop software besides their testing-related duties. As a result, 161 respondents (72.5%) indicated they use to develop in at least one

[2] In Brazil, people that hold a Bachelor's degree can take courses at the intermediate level, between the undergraduate degree and graduate degree (Master and Doctorate), called specialization courses.

[3] https://www.istqb.org/certification-path-root/why-istqb-certification.html/

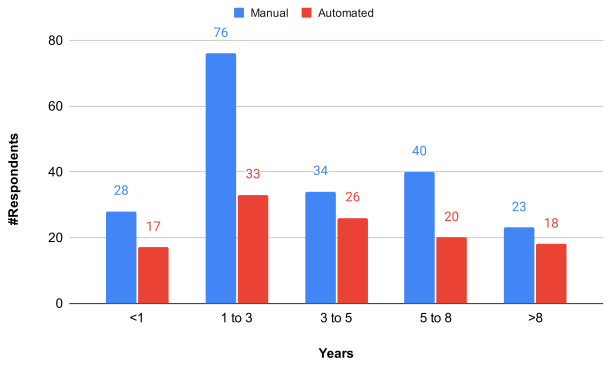

Figure 1: GUI testing type and participants' experience

programming language. The respondents cited 28 different programming languages, where the most cited ones were: Java (46 respondents - 20.8%), JavaScript (40 respondents - 18.3%), Python (28 respondents - 12.6%), and Ruby (27 respondents - 12.1%).

In terms of how they perform GUI testing, (i) 108 respondents (48%) stated they are used to test manually exclusively, ii) 21 respondents (9%) said they exclusively perform automated tests, and iii) 93 respondents (41%) answered that they perform tests both manually and automatically.

We also asked them about their experience with each type of test. For respondents who perform only manually:

- 14%: have less than 1 year of experience;
- 46%: have between 1 and 3 years of experience;
- 9%: have between 3 and 5 years of experience;
- 22%: have between 5 and 8 years of experience;
- 9%: have over 8 years of experience.

Similarly, for the respondents who perform automated tests only:

- For 19%: less than 1 year of experience;
- For 14%: between 1 and 3 years of experience;
- For 19%: between 3 and 5 years of experience;
- For 19%: between 5 and 8 years of experience;
- For 29%: over 8 years of experience.

The respondents who perform tests manually and automatically answered twice, one per type of test. First, we asked about the participant's experience with manual tests and then with automated tests. Thus, we were able to add the data and present only two groups in Figure 1. In details, for manual tests:

- For 14%: less than 1 year of experience;
- For 28%: between 1 and 3 years of experience;
- For 26%: between 3 and 5 years of experience;
- For 18%: between 5 and 8 years of experience;
- For 14%: over 8 years of experience.

For the experience with automated tests:

- For 14%: less than 1 year of experience;
- For 32%: between 1 and 3 years of experience;
- For 24%: between 3 and 5 years of experience;
- For 17%: between 5 and 8 years of experience;
- For 13%: over 8 years of experience.

Finding #1. Almost half of the respondents perform *manual* GUI testing exclusively.

4.2 Companies Characterization

This section presents the results for the companies in which the respondents work. We categorized a company as "Multi-Segment" when the answers represented more than one sector of activity. The respondents work in companies from different sectors: (i) 79 work in the Financial sector (35%); (ii) 48 respondents in Multi-Segment companies (21%); and (iii) 29 respondents work in e-commerce (13%). Other domains account for 30% of the responses, such as food, human resources, communication, games, healthcare, and logistics.

About the development model used in the companies, 52 respondents referred to use Scrum or Kanban. However, Scrum is one framework for project management, and Kanban is a visual work management system. Hence, knowing that software professionals associate Scrum and Kanban with agile methodologies, we added those responses to the "agile" category. Thus, 95 respondents (43%) work with Agile methods, 76 respondents use the incremental development model (35%), 33 respondents use the waterfall model (15%); and 16 respondents use other alternatives (7%).

We asked the respondents who are responsible for carrying out functional GUI testing. 142 respondents (64%) informed that the responsibility is exclusive to the Quality Assurance team, and 36 respondents (16%) reported that both the Quality Assurance team and the programmer(s) who developed the functionality share the responsibility. 43 respondents (20%) reported other reasons, such as programmers who did not develop the functionality (4.1%) and the user support team (3.4%). As the companies can develop applications for a set of platforms, we found that 194 respondents (46.3%) develop functional GUI testing for the Web platform, 141 respondents (33.7%) for the mobile platform, and 83 respondents (19.8%) for the desktop platform.

Finding #2. Waterfall and Incremental together represent most of the development models used in the companies from our dataset.

Finding #3. The Quality Assurance team and programmers who develop the functionalities are usually responsible for running GUI testing.

4.3 How do professionals perform manual GUI testing? (RQ1)

We asked the respondents about manual testing. After gathering data, we split the respondents into two groups: (i) those who only carry out manual tests; (ii) those who carry out manual and automated tests.

In Group (i), most respondents (86%) reported that they are responsible for *planning* and *executing* GUI tests; 11% reported that only *execute* manual tests; and 3% respondents informed that only

plan them. In Group (ii), we found a similar rate, as 93.6% respondents reported they perform test *planning* and *execution*; 3.2% only *execute* tests; and 3.2 % only *plan* tests.

We also asked what they use to support the manual testing. In Group (i), 44% reported that use *test cases* to support the testing process; 36% test in an *exploratory way*; and 19% often use *checklists*. In Group (ii), we identified a slight increase in the number of respondents who use *test cases* (52%); while 22% perform tests in an *exploratory way*; and 16% use *checklist*.

Additionally, the functionalities tested by the respondents are: (i) new features and some old ones (46%); (ii) all features, new and old, (31%); and 23% answered *others*: "only old feature", "what impacts the developed functionality", "behavior-based scenarios", "User journey", and "using heuristics".

Finding #4. Most software professionals plan and execute manual tests, use test cases to perform tests, and usually test new features and some old ones.

4.4 How do professionals perform automated GUI testing? (RQ2)

We asked the respondents who performed automated GUI testing on the programming language they used. Thus, we split the respondents into two groups: (i) those who only carry out automated tests; (ii) those who carry out both automated and manual tests.

In Group (i), the most cited languages were Java (37.1%), Python (20%), Ruby (17.1%), Javascript (14.3%), and others (12%), i.e., TypeScript, Kotlin, C#, and Robot Framework. In Group (ii), the most cited were Java (28.5%), Javascript (26.1%), Ruby (19.4%), and Python 14.6%). They also cited others (12%), TypeScript, Kotlin, C#, .net, Scala, Swift, and Visual Basic. In this question, the respondent could inform more than one programming language.

We also asked the respondents who choose the frameworks and tools used to support the automated testing. For Group (i), 22% answered the Quality Assurance (QA) team chooses, 22% informed the tester chooses, 17% said the company's leadership determines, 13% answered the project client decides, and 26% informed "Others", such as Project Client, Project Owner, and Development Team. For Group (ii), 36% answered the QA team decides internally, 20% informed the tester chooses, 16% said the company's leadership determines, and 28% reported others.

Table 3 shows the most cited tools and frameworks for automating GUI testing, as reported by the respondents. The most cited were: *Selenium* (36%), *Appium* (21%), *Cypress* (9%), and *TestComplete* (5%). The "Others" category (7%) comprises the tools mentioned by only one participant, such as *Autoit*, *Android Studio* and *Orange Testing*. In addition, they reported that generally use those tools combined with additional frameworks, like *Cucumber* (47%), *Capybara* (23%), and *Behave* (8%). 22% informed others tools, such as *Watir*, *Percy* and *Rest Sharp*. In this question, the participant could choose more than one option.

We also investigated whether the tools meet the respondents' needs. Figure 2 shows the results for both groups, Group (i) and Group (ii). In Group (ii), 80.6% of the participants reported the tools meet their needs, 11.8% answered they partially meet, 4.3% reported that they do not know the tools meet, and 3.2% informed the tools do not meet. In Group (i), 85.7% reported they meet, 4.8% answered they partially meet, 4.8% informed they do not meet, and 4.8% said they do not know. In general, comparing the responses, we observed: (1) most of them agree that the tools and frameworks meet their needs; and (2) the negative answers were more significant for the Group (ii) – 15% combining *Partially* and *do not meet*). The testers who only execute automated tests, Group (i), can have more knowledge about the existing tools. The percentages for the ones that do not know were quite similar.

Table 3: Most cited Tools/Frameworks

Tools/Frameworks	#Respondents	%
Selenium	91	35,97%
Appium	53	20,95%
Cypress	24	9,49%
Others	19	7,51%
TestComplete	13	5,14%
Sikuli	10	3,95%
SilkTest	8	3,16%
Robot Framework	7	2,77%
Microsoft Coded UI Tests	6	2,37%
Oracle Application Testing Suite	4	1,58%
UFT	4	1,58%
IBM Rational Functional Tester	3	1,19%
Ranorex	3	1,19%
Protractor	2	0,79%
TestCafe	2	0,79%
VS Code	2	0,79%
Webdriver IO	2	0,79%

The use of frameworks/tools is essential in automated GUI testing, but the respondents also pointed to some limitations to an opened-end question. We coded the responses and associated them with one or more product quality (sub)characteristics (ISO/IEC 25010 [37]). We identified twelve different limitations, as **Table 4** shows. We observed that more than 40% of the respondents did not inform any limitation. They reported that either (i) the frameworks meet their needs or (ii) they do not know any limitations (lack of knowledge). On the other side, the most cited quality (sub)characteristics (limitations) refers to Functional Completeness (17.2%), Time Behaviour (7.8%), and Interoperability (5.2%).

Finding #5. Java is the most widely used programming language for automating functional GUI tests. However, testers have also used Javascript, Ruby, and Python.

Finding #6. The Quality Assurance team usually determines the frameworks and tools to support the automated GUI testing.

Finding #7. Selenium is the most used tool for automating GUI tests. Testers also claim that the tools usually meet their needs, even though they have limitations.

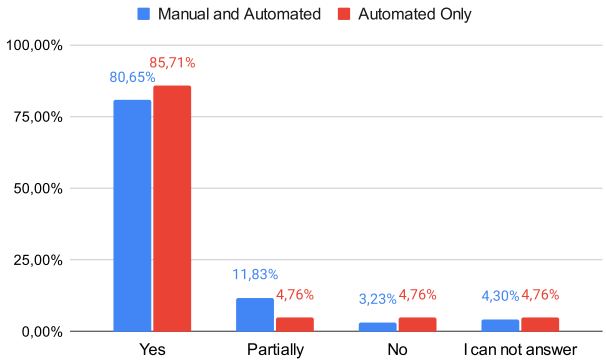

Figure 2: Do frameworks and tools meet the needs of testers?

Table 4: Tools Limitations

Limitations	#Respondents	%
Not informed limitations	49	42,2%
Functional Completeness	20	17,2%
Others	13	11,2%
Time Behaviour	9	7,8%
Interoperability	6	5,2%
Adaptability	5	4,3%
Functional Appropriateness	3	2,6%
Modifiability	3	2,6%
Operability	3	2,6%
Capacity	3	0,9%
Functional Correctness	2	0,9%
Installability	2	0,9%
Learnability	2	0,9%
Resource Utilization	2	0,9%

Regarding the proportion of GUI tests performed automatically, we found differences between Group (i) and Group (ii). On the one hand, respondents from Group (i) stated that:

- For 9% of the respondents: 100% of the tests are automated;
- For 29%: 70% to 90% of tests are automated;
- For 14%: 50% to 70% of tests are automated;
- For 24%: 30% to 50% of tests are automated;
- For 24%: 10% to 30% of tests are automated.

On the other hand, respondents from Group (ii) stated that:

- For 2% of the respondents: 100% of tests are automated.
- For 17%: 70% a 90% of tests are automated;
- For 26%: 50% a 70% of tests are automated;
- For 24%: 30% a 50% of tests are automated;
- For 31%: 10% a 30% of tests are automated;

We observed that almost 10% of Group (i) reported that all tests are automatically performed. 38% reported that from 70% to 100% of all tests are automated for Group (i). Compared to Group (ii), which performs manual and automated tests, this number is much lower, 19%. We also observed that almost half of the respondents reported that only 10% to 50% of all tests are automated.

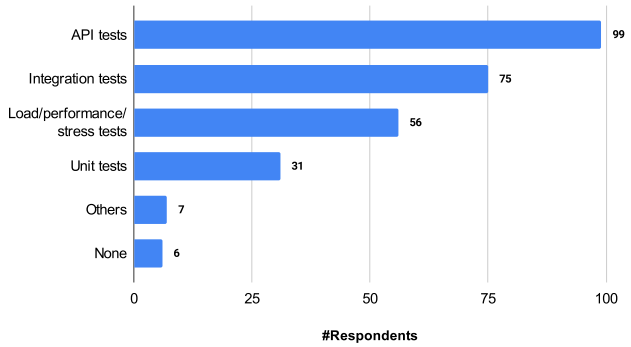

Figure 3: Others types of automated tests performed by respondents

Regarding the frequency at which automated tests are performed, 39% reported that they run the tests after adding a new feature; 25% after bug fixes; 22% run automated tests daily; and 14% informed other frequencies, such as three days a week, 15 days a week, and at the end of the sprints. As respondents could choose more than one option, the results exceed 100%.

In addition to GUI tests, the respondents stated to automate other types of tests, such as API tests (37%), integration tests (28%), performance (21%), and unit tests (12%). Figure 3 shows the most cited test types. Other types received one response each, such as regression tests, chatbot tests, and security.

Finding #8. More than half of software testers reported performing automated tests after adding new features to the system. In many cases, the number of automated tests is still low.

4.5 Which tools do participants commonly use to report project issues and bugs? (RQ3)

For manual tests, the respondents informed many different ways to report tests. Figure 5 shows the most cited ways and the number of respondents to each one. For example, about 41% of the testers report bugs using cards in Kanban, 18% specifically informed the Jira tool, 11% send a message on Slack, and 6% uses Mantis.

For the execution of automated GUI tests, they cited far fewer ways to report bugs, as Figure 4 shows. The most cited ways were: cards in Kanban (40%), Slack (15%), Jira (13%), and e-mail (9%). The *others* category (11%) includes the ways reported to one tester each, such as Mantis, Gitlab, Trello, and calls to the development team.

Finding #9. Kanban board was the most cited way to report bugs, for both manual and automated execution of GUI tests.

5 DISCUSSION

Testers profile. Most testers working with GUI testing have higher education, although a small portion has only completed high school or technical courses (6%). Additionally, few respondents have test certifications, even though there is a local ISTQB, the quality seal

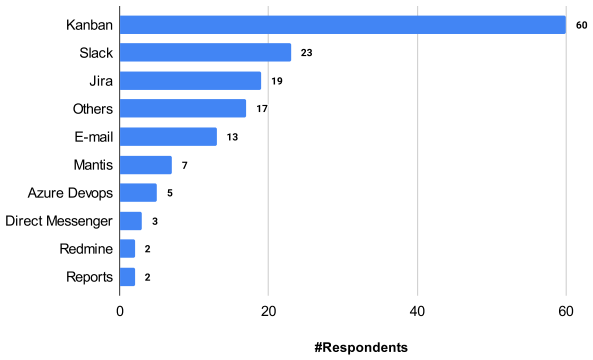

Figure 4: Ways to report bugs in automated tests

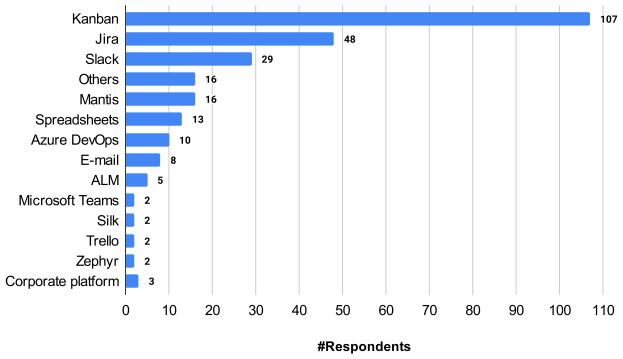

Figure 5: Ways to report bugs in manual tests

for software testers, the council in the country (BSTQB)[4]. Also, a significant number of testers have between 1 to 5 years of experience, showing that the area is sought after by recently graduated software professionals. Further studies are needed to investigate the reasons behind this.

When we compared the testers group that executes manual tests only with the testers group that evaluates software in an automated way, the main difference observed is fluency in programming languages. 50.45% of the first group did not know any programming language for the automated test process. This result could be one of the reasons for the large number of evaluations done manually. On the other hand, only 7.89% of the professionals that execute automated tests no have experience in some programming language. Although the evidence shows a low fluency of professionals who only perform manual testing in programming languages, it is essential to highlight that there are solutions to automate tests, such as Record and Playback techniques, which do not need explicit programming [24].

Manual execution of GUI tests. Even with the increasing adoption of automated techniques and several tools and frameworks, many testers still perform manual checks on the system interface. Since the systems under test can require specific domain knowledge, it can depend on the developers' expertise to be tested, making testing more challenging to automate.

Another factor that may be associated with manual tests is related to the return on investment. Creating and maintaining automated tests might be a time-consuming task for testers. For large projects, the amount spent on automation tends to be worthwhile if there is a well-defined reuse process of tests that need to perform several times. However, if the software project is small, has a low budget, short time to develop, inexperienced team, and without the need for constant reproducibility of tests, automating may not be essential.

Automating tests requires different professional and understanding software skills, such as source code development and familiarity with different tools for the automation process. Thus, the time for completion of the software project may be less than the time required for the learning curve necessary to create automated tests.

Automated execution of GUI tests. From the testers who reported that the QA team or the testers themselves are responsible for choosing the test automation tools, half use the Agile methodology. The autonomy to decide how to do the work could be directly associated with the principles of the applied development model. The Agile manifesto states that the team must reflect on how to become more effective and adjust their solutions to comply with this[5]. The project must provide the environment and the necessary support to feel confident and define how to perform their work.

In addition to the tools used in the tests, the testers reported a set of additional frameworks, such as Cucumber, Capybara, and Behave, which use BDD (Behavior Driven Development) in the quality process [12]. This technique allows the development of automated tests through scenarios that are easy to read and understand by any project member, such as requirement analyst, developer, and tester. Therefore, projects that use BDD seek a collaborative approach to define and share the tests. Another characteristic of BDD is the special attention to the client's business needs, in which market requirements are necessary information used for test development.

Furthermore, the Java programming language remains to be popular with software professionals in the field. However, we have also identified the acceptance of dynamic typing languages for the testing process, e.g., Javascript, Python, and Ruby. Testers said that developing tests with those languages is usually easy because, for example, it is not necessary to define the type in every variable of the code. Consequently, it takes less time to test creation. Another advantage of dynamic typing languages is the compatibility with Selenium, a popular tool for automating GUI testing.

Among respondents who use Selenium, we observed that the second generation of GUI testing is the most used for automation. Some advantages of this tool are open-source license, compatibility with several operating systems (Windows, Linux, and Mac), an active community for technical support, and improvements.

According to the limitations of the tools, among the complaints of respondents regarding Functional Completeness, 30% criticized the lack of support from the Cypress framework for running GUI tests in several tabs at the same time. Consequently, testers are probably looking to perform more complex tests and interact with

[4]https://bstqb.org.br/

[5]https://agilemanifesto.org/

the browser instead of just using one system under test (SUT) web page. Although some participants reported having problems with the automation solutions, most stated that they meet their design needs.

Issues management. Kanban board was the most popular among the software professionals to report the bugs found during GUI testing. Unlike traditional openings in service desk platforms, Kanban allows an integrated communication between the people on the team, where everyone can contribute relevant information on the board's cards. In addition, Kanban is easy to interpret and use, as it has a graphic proposal of the activities to perform.

6 THREATS TO VALIDITY

Construct Validity.

We disregard software professionals who have no experience with GUI tests, blank answers, and invalid answers to minimize the difficulties in extracting the study data. Our study only considered Brazilian companies and testers. When sending invitations, we observed that most selected companies came from the South and Southeast regions, which may have a particular culture and impact the results. However, software testers with different profiles, education, and experience in GUI testing answered the survey. The number of questions in the survey also might be a threat. To organize the study, we grouped the sections by subjects, which may have helped and supported the respondents during the filling out.

Internal Validity. To reduce the risk of misinterpretations in the survey questions, we conducted a pilot study with experienced testers before sending the survey to practitioners. They also reported that the average time to complete the study was 10 minutes during the pilot, which we consider not to be such a long time.

External Validity. Even though the survey produced many valid entries (222 participants), they may not adequately represent the entire population of software testers in the Brazilian industry. However, we believe that the collected data can assist in understanding the current context of the execution of the GUI tests.

Reliability. It is a threat related to the use of research data. Two researchers performed both the coding of the open responses and the calculation of the percentages of the quantitative data to prevent wrong interpretations from being made to the responses collected—any disagreements in the processing of responses we discussed until we reached consensus and correction of existing errors. Subsequently, a third author reviewed the information transcribed in the text to guarantee the quality of the data.

7 RELATED WORK

We identified few survey studies in the recent literature that investigates functional GUI testing.

Qureshi and Nadeem [30] surveyed the literature to identify the techniques for generating GUI test cases. The authors grouped 12 techniques and classified them concerning the failure model to help determine which solution is most suitable for the testing process. They used four failure models: SMFM (State Machine Fault Model), CFFM (Control Flow Fault Model), TFM (Textual Fault Model), and IFM (Interface Fault Model).

Janicki et al. [20] also studied automated GUI test generation. The authors surveyed 59 participants seeking to know the obstacles and opportunities for the industry to implement model-based testing. As a result, the authors verified the software professionals' interest in model-based testing and identified the need for further research to facilitate that model creation and maintenance.

Seeking to understand more about the generations of automated GUI testing for mobile applications, Ardito et al. [9] conducted an empirical analysis to compare the EyeAutomate (third generation) and Espresso (second generation) tools. The authors surveyed 78 undergraduate students who create tests for open-source Android applications in those tools. The survey participants answered one questionnaire to share their perception of the difficulty in developing the tests. As a result, the authors observed that the participants' productivity was similar in both tools. However, the tests created in EyeAutomate have higher quality regarding the subsequent correct execution of the test scripts. Another finding is for the students to show more preference for EyeAutomate since the components through the DOM (Document Object Model) must be correctly identified to use Espresso.

Regarding GUI testing for mobile applications, Linares-Vásquez et al. [26] reported that developers often perform evaluations manually because they have no confidence in the automated testing techniques. Hence, the authors aimed to help researchers and practitioners understand the needed requirements to develop tests for Android apps. For this purpose, they surveyed 102 developers who contributed to open-source projects on the Android platform. As a result, they concluded that developers use user stories and use cases to create tests. They also noted a preference for automatically generated high-level tests. Furthermore, the participants reported that the code coverage is not a necessary quality metric for test cases.

Although those studies focus on different aspects of GUI functional testing, they did not deeply investigate how practitioners perform manual and automated tests. We believe that our study complements the previous ones by analyzing the responses of 222 participants and bringing more evidence on how GUI tests are performed. For example, on how they perform automated and manual GUI testing and report bugs. Besides, the participants were informed which tools or frameworks use at work and their difficulties using them.

8 CONCLUSION

This study provided a general view of how the professionals from the Brazilian software industry perform functional GUI testing. The first step towards one common understanding of the practices industry should follow for performing GUI testing. We surveyed 222 software professionals from Brazilian software companies to identify their experiences and practices concerning GUI functional testing.

On the one hand, many software professionals perform manual GUI testing using test cases or exploratory testing. On the other hand, the software professionals who perform automated GUI testing cited Java and Selenium as the programming language e framework most used, respectively. Besides, they informed using BDD as one additional tool to create the tests.

Additionally, about half of the software professionals who perform automated GUI testing reported that the solutions do not have

limitations. Those who reported limitations complained about the long time to complete the tests and the need to test the system in more than one browser tab.

As future work, we intend to understand how automated tools for testing can support and improve GUI testing in large-scale scenarios. Additionally, analyzing and understanding how software professionals report bugs is important because their comprehension can improve software quality.

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and FAPESB grant JCB0060/2016.

REFERENCES

[1] Emil Alégroth, Robert Feldt, and Pirjo Kolström. 2016. Maintenance of automated test suites in industry: An empirical study on Visual GUI Testing. *Inf. Softw. Technol.* 73 (2016), 66–80.

[2] Emil Alégroth, Robert Feldt, and Helena Holmström Olsson. 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)*. IEEE Computer Society, 56–65.

[3] Emil Alégroth, Robert Feldt, and Lisa Ryrholm. 2015. Visual GUI testing in practice: challenges, problems and limitations. *Empir. Softw. Eng.* 20, 3 (2015), 694–744.

[4] Emil Alegroth, Zebao Gao, Rafael Oliveira, and Atif Memon. 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)*.

[5] Amira Ali, Huda Amin Maghawry, and Nagwa Badr. 2018. Automated parallel GUI testing as a service for mobile applications. *Journal of Software: Evolution and Process* 30, 10 (2018).

[6] AltexSoft. 2020. *Quality-Assurance-Quality-Control-and-Testing-Whitepaper*. https://behave.readthedocs.io/en/stable/philosophy.html

[7] Domenico Amalfitano, Nicola Amatucci, Atif M. Memon, Porfirio Tramontana, and Anna Rita Fasolino. 2017. A general framework for comparing automatic testing techniques of Android mobile apps. *J. Syst. Softw.* 125 (2017), 322–343.

[8] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M. Memon. 2012. Using GUI Ripping for Automated Testing of Android Applications. In *27th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 258–261.

[9] Luca Ardito, Riccardo Coppola, Maurizio Morisio, and Marco Torchiano. 2019. Espresso vs. EyeAutomate: An Experiment for the Comparison of Two Generations of Android GUI Testing *(EASE '19)*. ACM, 13–22.

[10] Yauhen Leanidavich Arnatovich and Lipo Wang. 2018. A Systematic Literature Review of Automated Techniques for Functional GUI Testing of Mobile Applications. arXiv:cs.SE/1812.11470

[11] Ishan Banerjee, Bao Nguyen, Vahid Garousi, and Atif Memon. 2013. Graphical User Interface (GUI) Testing: Systematic Mapping and Repository. *Information and Software Technology* 55, 10 (Oct. 2013), 1679–1694.

[12] Richard Jones Benno Rice and Jens Engel Revision. 2017. Behavior Driven Development. https://behave.readthedocs.io/en/stable/philosophy.html

[13] Stefan Berner, Roland Weber, and Rudolf K. Keller. 2005. Observations and Lessons Learned from Automated Testing. In *Proceedings of the 27th International Conference on Software Engineering (ICSE '05)*. ACM, 571–579.

[14] Jehad Al Dallal. 2009. Automation of object-oriented framework application testing. In *2009 5th IEEE GCC Conference Exhibition*. IEEE, 1–5.

[15] Omar El Ariss, Dianxiang Xu, Santosh Dandey, Brad Vender, Phil McClean, and Brian Slator. 2010. A Systematic Capture and Replay Strategy for Testing Complex GUI Based Java Applications. In *Proceedings of the 2010 Seventh International Conference on Information Technology: New Generations (ITNG '10)*. IEEE, USA.

[16] Vahid Garousi, Wasif Afzal, Adem Çağlar, Berk Işık, Berker Baydan, Seçkin Çaylak, Ahmet Zeki Boyraz, Burak Yolaçan, and Kadir Herkiloğlu. 2017. Comparing Automated Visual GUI Testing Tools: An Industrial Case Study. In *Proceedings of the 8th ACM SIGSOFT International Workshop on Automated Software Testing* (Paderborn, Germany) *(A-TEST 2017)*. ACM, 21–28.

[17] Vahid Garousi, Wasif Afzal, Adem Çağlar, İhsan Berk Işık, Berker Baydan, Seçkin Çaylak, Ahmet Zeki Boyraz, Burak Yolaçan, and Kadir Herkiloğlu. 2020. Visual GUI testing in practice: An extended industrial case study. arXiv:cs.SE/2005.09303

[18] Mark Grechanik, Qing Xie, and Chen Fu. 2009. Creating GUI Testing Tools Using Accessibility Technologies. In *International Conference on Software Testing, Verification, and Validation Workshops*. IEEE, 243–250.

[19] S Gunasekaran and V Bargavi. 2015. Survey on automation testing tools for mobile applications. *International Journal of Advanced Engineering Research and Science* 2, 11 (2015), 2349–6495.

[20] Marek Janicki, Mika Katara, and Tuula Pääkkönen. 2012. Obstacles and opportunities in deploying model-based GUI testing of mobile software: a survey. *Software Testing, Verification and Reliability* 22, 5 (2012), 313–341.

[21] Katja Karhu, Tiina Repo, Ossi Taipale, and Kari Smolander. 2009. Empirical Observations on Software Testing Automation. In *International Conference on Software Testing Verification and Validation*.

[22] Onur Kilincceker, Alper Silistre, Fevzi Belli, and Moharram Challenger. 2021. Model-Based Ideal Testing of GUI Programs–Approach and Case Studies. *IEEE Access* 9 (2021), 68966–68984.

[23] Meile Posthuma Klaus Olsen and Stephanie Ulrich. 2019. *International Software Testing Qualifications Board* (third ed.). International Software Testing Qualifications Board. https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html

[24] Shin-Jie Lee, Yu-Xian Chen, Shang-Pin Ma, and Wen-Tin Lee. 2018. Test Command Auto-Wait Mechanisms for Record and Playback-Style Web Application Testing. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 2. 75–80.

[25] Andreas Leitner, Ilinca Ciupa, Bertrand Meyer, and Mark Howard. 2007. Reconciling Manual and Automated Testing: The AutoTest Experience. In *40th Annual Hawaii International Conference on System Sciences (HICSS '07)*.

[26] Mario Linares-Vásquez, Cárlos Bernal-Cardenas, Kevin Moran, and Denys Poshyvanyk. 2017. How do Developers Test Android Applications?. In *IEEE International Conference on Software Maintenance and Evolution (ICSME '2017)*. 613–622.

[27] A.M. Memon, M.E. Pollack, and M.L. Soffa. 2001. Hierarchical GUI test case generation using automated planning. *IEEE Transactions on Software Engineering* 27, 2 (2001), 144–155.

[28] Duaa R. Mohammad, Sajedah Al-Momani, Yahya M. Tashtoush, and Mohammad Alsmirat. 2019. A Comparative Analysis of Quality Assurance Automated Testing Tools for Windows Mobile Applications. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*.

[29] Ioannis Parissis and Farid Ouabdesselam. 1996. Specification-based Testing of Synchronous Software. In *SIGSOFT '96, Proceedings of the Fourth ACM SIGSOFT Symposium on Foundations of Software Engineering*. ACM, 127–134.

[30] I. A. Qureshi and A. Nadeem. 2013. GUI Testing Techniques: A Survey. *International Journal of Future Computer and Communication* (2013), 142–146.

[31] Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, Kai Petersen, and Mika V. Mäntylä. 2012. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In *7th International Workshop on Automation of Software Test (AST '2012)*. 36–42.

[32] Rudolf Ramler and Klaus Wolfmaier. 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* (Shanghai, China) *(AST '06)*. ACM, 85–91.

[33] F. Saglietti and F. Pinte. 2010. Automated Unit and Integration Testing for Component-Based Software Systems, In International Workshop on Security and Dependability for Resource Constrained Embedded Systems. *ACM*.

[34] Mary Sánchez-Gordón, Laxmi Rijal, and Ricardo Colomo-Palacios. 2020. Beyond Technical Skills in Software Testing: Automated versus Manual Testing. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops* (Seoul, Republic of Korea) *(ICSEW'20)*. ACM, 161–164.

[35] R. M. Sharma. 2014. *Quantitative Analysis of Automation and Manual Testing*. Technical Report Volume 4. International Journal of Engineering and Innovative Technology (IJEIT).

[36] Pradeep Sharma Shruti Malve. 2017. Investigation of Manual and Automation Testing using Assorted Approaches. *International Journal of Scientific Research in Computer Science and Engineering* 5 (4 2017), 81–87. Issue 2. https://www.isroset.org/journal/IJSRCSE/full_paper_view.php?paper_id=408

[37] ISO 25000 STANDARDS. 2011. ISO/IEC 25010. Retrieved May 03, 2021 from https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?start=6

[38] Porfirio Tramontana, Domenico Amalfitano, Nicola Amatucci, and Anna Rita Fasolino. 2019. Automated functional testing of mobile applications: a systematic mapping study. *Softw. Qual. J.* 27, 1 (2019), 149–201.