Journal of Software Engineering Research and Development, 2026, 14:14, doi: 10.5753/jserd.2026.6604

Inside the Mobile Community: GUI Testing Discussions and Challenges on Stack Exchange

Matheus Costa Pinto Marçal [**Federal University of Bahia** | ***matheuscpm@ufba.br***]
Joselito Mota Júnior [**Federal University of Bahia** | ***joselito.mota@ufba.br***]
Lidia Perside Gomes Nascimento [**Federal University of Bahia** | ***lidianascimento@ufba.br***]
Claudio Sant'anna [**Federal University of Bahia** | ***claudionsa@ufba.br***]
Ivan Machado [**Federal University of Bahia** | ***ivan.machado@ufba.br***]

Abstract

The Graphical User Interface (GUI) is increasingly becoming the focus of developers and testers. A well-designed, defect-free interface can significantly affect whether the application functions properly and provides a satisfying user experience. In particular, in the era of smartphones with varying screen sizes and operating systems, mobile developers and testers are increasingly developing and using interface tests. This article aims to characterize the practical landscape of Mobile GUI testing by analyzing discussions among practitioners across six Stack Exchange communities, identifying the main challenges, tools, and platform-specific differences reported by developers and testers. To achieve this goal, we collected and filtered data from Stack Exchange Data Dumps covering the years 2012 to 2023, combining automated keyword-based filtering with manual validation by three independent analysts, resulting in 714 posts and 203 comments for analysis. The yielded findings show that the main challenges are device compatibility, unexpected UI interference, and limitations in automation tools. Appium emerges as the most prevalent tool, mentioned in over 40% of analyzed posts, while platform-specific frameworks such as Espresso and UI Automator (Android) and XCTest and XCode UI Test (iOS) are also widely discussed. The results also show that Android offers greater testing flexibility, whereas iOS imposes stricter restrictions that require additional configuration. This study provides a practitioner-driven synthesis of real-world Mobile GUI testing practices, shedding light on the main trends and practices developed by the community.

Keywords: *Developer expertise, software testing, gui testing, stack exchange mining, testing community*

1 Introduction

The practice of GUI testing is gaining popularity and becoming a focus for developers and testers, as interacting with the interface is essential to determine whether an application works properly (Bryce et al., 2010; Chaudhary and Sangwan, 2016; Genc-Nayebi and Abran, 2017; Herbold and Harms, 2013; Junior et al., 2021; Lientz et al., 1978). GUI testing encompasses multiple techniques, either manual or automated, focused on performance, usability, and other aspects (Mao et al., 2016; Mazuera-Rozo et al., 2020; Memon et al., 2001; Palomba et al., 2019; Yuan and Memon, 2007, 2009). In the era of smartphones with different screen sizes and operating systems, mobile developers and testers increasingly rely on interface tests to ensure software quality. Despite these efforts, bugs are still misclassified (Anvik et al., 2006; Júnior et al., 2021; Kanwal and Maqbool, 2012; Nascimento et al., 2024; Xuan et al., 2012; Zubrow, 2009), including user interface bugs, still appear and can negatively affect software performance (Aranda and Venolia, 2009; Bernardi et al., 2012; Cavalcanti et al., 2014; Herzig et al., 2013). An inaccurate or unfeasible reproduction of reported UI defects can delay software delivery (Aranda and Venolia, 2009; Bettenburg et al., 2007, 2008; Kanwal and Maqbool, 2012; Kim and Whitehead, 2006; Rajlich, 2014), and the possibility of introducing new bugs should be carefully avoided (Bernardi et al., 2012; Kim and Whitehead, 2006; Mockus, 2010).

Stack Exchange communities are online spaces widely used by developers to seek help or assist others. They represent a valuable source for identifying trends and real-world practices in software development (Gomes et al., 2022; Martins et al., 2023; Posnett et al., 2012; Rosen and Shihab, 2016; Tahir et al., 2020). Online forums serve as a bridge between academic research and daily practice, as researchers are often separated from the practical application of technologies, and gray literature helps fill that gap.

Our analysis of the literature reveals a significant lack of systematically organized knowledge regarding the practices, tools, and challenges discussed by testing practitioners in their daily work (Arnatovich et al., 2016; Mahmood et al., 2014; Wu et al., 2016; Zein et al., 2016). This overall lack of synthesis in the literature may contribute to a disconnect between academic research and industrial practice in the context of Mobile GUI testing. Furthermore, a lack of visibility into peer practices for addressing testing challenges may reduce the efficiency of mobile GUI testing. As a result, testing activities may take longer and yield less effective outcomes, potentially impacting the overall quality and user experience of mobile applications.

To address this gap, this work aims to illustrate the practical landscape of Mobile GUI testing by analyzing discussions among practitioners across six Stack Exchange communities and identifying the main challenges, tools, and platform-specific differences reported by developers and testers. Specifically, the study investigates: (i) the main challenges and problems faced in mobile GUI testing (RQ1), (ii) the testing methods and tools discussed and adopted by practitioners (RQ2), and (iii) whether GUI testing methods differ

between the Android and iOS platforms (RQ3).

To achieve this objective, we performed an empirical study using the official Stack Exchange Data Dumps, covering data from 2012 to 2023. The methodology comprised four phases: selection of six relevant Stack Exchange repositories; automated content filtering using keyword-based Python scripts; manual analysis of the filtered posts and comments by three independent analysts; and qualitative data analysis of the validated content. In total, over one million posts and 1.7 million comments were processed, resulting in 714 posts and 203 comments that met our inclusion criteria after manual validation.

Our findings reveal that the main challenges in mobile GUI testing stem from device compatibility, unexpected UI interference, and limitations in test automation tools. Appium emerges as the dominant tool across both platforms, mentioned in over 40% of all analyzed posts and comments. At the same time, native frameworks such as Espresso and UI Automator are preferred for Android, and XCTest and XCode UI Test for iOS. The analysis also shows that testing on Android is generally more flexible, whereas iOS imposes stricter restrictions that require additional configuration efforts. This article contributes to the literature by providing a practitioner-driven synthesis of the community discussion on Mobile GUI testing, focusing on real-world challenges, tools, and platform-specific differences that have not yet been systematically documented.

The remainder of this article is organized as follows. Section 2 presents the background on Mobile GUI testing and the Stack Exchange platform. Section 3 details the methodology, including the research questions, data collection, filtering, and analysis procedures. Section 4 presents the results organized by research question. Section 5 discusses the findings and their implications for practitioners and researchers. Section 6 discusses related work and positions our study within the existing literature. Section 7 addresses threats to validity, and Section 8 concludes the article with final remarks and directions for future work.

2 Background

This section outlines the core concepts relevant to this study. Section 2.1 presents an overview of Mobile GUI testing, covering its role in the software development lifecycle, the main testing approaches and tools adopted in mobile environments, and the growing relevance of this practice given the diversity of devices and platforms on the market. Section 2.2 introduces the Stack Exchange platform, describing its structure, community dynamics, and its suitability as a data source for investigating how practitioners discuss and share knowledge about Mobile GUI testing in practice.

2.1 Mobile GUI Testing

Software testing is an essential part of the software development cycle and is also one of its most challenging aspects (Alomar et al., 2021; Farley, 2021). Over the years, numerous approaches and tools have been developed to enhance the effectiveness, efficiency, and practicality of the testing process across various aspects of software (Choi et al., 2013; Li et al., 2014; McMinn, 2011). Most of an app's code is built with the User Interface in mind, making testing a necessary phase in software development (Mahajan and Shneiderman, 1997; Memon, 2002; Myers, 1995). This activity can significantly enhance the overall quality of the software (Harrold, 2000; McConnell, 1996).

One of these aspects is the Graphical User Interface (GUI), which serves as the primary point of interaction between users and software. As the main medium through which users interact with an application, the GUI plays a critical role in shaping the overall user experience (Bryce et al., 2010; Chaudhary and Sangwan, 2016; Herbold and Harms, 2013).

GUI testing encompasses a variety of approaches, including functional, usability, and automated testing (Nie et al., 2023). GUI testing tools and techniques have evolved considerably and are now widely used (Amalfitano et al., 2014; Azim and Neamtiu, 2013; Deng et al., 2017; Sadeghi et al., 2017). A wide range of tools, such as Espresso, UIAutomator, and Appium, support the automation of testing activities in mobile applications. In mobile software development, GUI testing has become increasingly relevant as mobile devices continue to dominate the market. As mobile devices continue to dominate the market, ensuring the quality and consistency of graphical user interfaces across heterogeneous environments has become a critical concern (Adamo et al., 2018; Song et al., 2017; Su et al., 2017).

As the Mobile GUI Testing landscape continues to evolve with the introduction of new tools and approaches, the availability of accessible and consolidated knowledge becomes increasingly important.

2.2 Stack Exchange

Stack Exchange, a collection of online forums often focused on technology, provides a valuable space for knowledge sharing (Gomes et al., 2022; Posnett et al., 2012; Tahir et al., 2020). It was created in 2008 when other technology forums suffered from poor moderation and undesired paywalls. It started as Stack Overflow, a general forum for all things programming-related, but it soon expanded into many different and more focused forums. Nowadays, it boasts over 11 million registered users and over 7 million posts made, including questions and answers. Users frequently seek assistance concerning software development, and more experienced users offer guidance through answers and comments. Although Mobile GUI testing does not have a dedicated Stack Exchange forum, information on the subject can be found across several development-focused sub-forums. Therefore, synthesizing and analyzing the dispersed knowledge on Mobile GUI testing from these forums could provide valuable insights for beginners and professionals in Quality Assurance.

2.3 Mobile GUI Testing

Mobile GUI testing lies at the intersection of several foundational concepts in software engineering. To contextualize the scope and challenges of this practice, this subsection progressively introduces the key topics that underpin it. It be-

gins with software testing as a broad discipline, then narrows the focus to the role of graphical user interfaces in modern software systems, followed by the challenges associated with testing these interfaces. Finally, it discusses the particularities that emerge when GUI testing is applied to mobile platforms.

2.3.1 Software Testing

Software testing is an indispensable part of the software development lifecycle and one of its most challenging aspects (Alomar et al., 2021; Farley, 2021). It encompasses a set of activities aimed at evaluating and improving the quality of software products by verifying that they function correctly, meet specified requirements, and are free from critical defects (ISO/IEC/IEEE, 2022).

Over the years, numerous approaches and tools have been developed to enhance the effectiveness, efficiency, and practicality of testing, ranging from unit and integration testing to system-level and acceptance testing (Li et al., 2014; McMinn, 2011). The importance of testing is well-established in the literature: defects identified late in the development cycle are significantly more costly to fix than those caught early (Boehm, 1981), and inadequate testing can lead to system failures with severe financial and operational consequences (Tassey, 2002).

Testing can be broadly classified into functional and non-functional categories. Functional testing verifies that the software behaves according to its specifications, while non-functional testing evaluates attributes such as performance, usability, security, and reliability (Ammann and Offutt, 2016). Within these categories, testing can be performed manually or through automated techniques, the latter becoming increasingly prevalent with the adoption of Agile and DevOps practices that demand continuous integration and continuous testing (Humble and Farley, 2010).

2.3.2 Graphical User Interface (GUI)

A Graphical User Interface (GUI) is the visual layer through which users interact with a software system, replacing command-line interactions with graphical elements such as windows, buttons, text fields, menus, and icons (Myers, 1995). GUIs have become the dominant paradigm for human-computer interaction, as they allow users to perform complex operations through intuitive visual representations rather than memorizing textual commands (Mahajan and Shneiderman, 1997).

Most modern software applications are built with the user interface as a central component: it's estimated that a substantial portion of an application's code is dedicated to the presentation layer (Mahajan and Shneiderman, 1997; Memon, 2002). The quality of the GUI directly affects the overall user experience, making it a critical factor in the success or failure of a software product (Bryce et al., 2010; Chaudhary and Sangwan, 2016; Herbold and Harms, 2013).

A GUI receives user events, such as mouse clicks, keyboard input, touch gestures, selections, and translates them into operations that modify the application's state (Banerjee et al., 2013). This event-driven nature introduces unique challenges for software quality assurance, as the number of possible event sequences grows combinatorially, making exhaustive testing infeasible (Memon et al., 2001).

2.3.3 Mobile GUI Testing

In the mobile domain, GUI testing has become particularly relevant and is gaining in both importance and complexity. Global smartphone shipments reached approximately 1.26 billion units in 2025 (International Data Corporation (IDC), 2025), and mobile devices continue to dominate internet traffic. The proliferation of devices with different screen sizes, resolutions, hardware capabilities, and operating system versions makes GUI testing in mobile platforms significantly more challenging than in traditional desktop environments (Adamo et al., 2018; Song et al., 2017; Su et al., 2017).

Mobile GUI testing inherits the general challenges of GUI testing but adds several platform-specific dimensions. The coexistence of thousands of device models with varying screen densities, aspect ratios, and OS versions is one of the most frequently reported difficulties (Nie et al., 2023; Zein et al., 2016). Mobile applications must also handle interruptions such as incoming calls, notifications, and permission dialogs, which can unexpectedly alter the application's state during test execution (Arnatovich and Wang, 2018). Moreover, mobile-specific interactions such as multi-touch gestures, device rotation, and sensor-based input (GPS, accelerometer) require specialized testing strategies that go beyond traditional GUI testing approaches (Mahmood et al., 2014; Wu et al., 2016).

The tooling landscape for mobile GUI testing is diverse and platform-dependent. For Android, the most widely adopted tools include Espresso, Google's native UI testing framework that synchronizes tests with the application's main thread for reliable execution, and UI Automator, which enables testing across multiple applications and system UI components (Nie et al., 2023). For iOS, XCTest and XCUITest, integrated within Apple's Xcode environment, serve as the primary frameworks for UI test automation. Appium, an open-source cross-platform tool that extends the Selenium WebDriver protocol to mobile platforms, is widely used for teams that need to maintain test suites across both Android and iOS (Amalfitano et al., 2017; Arnatovich and Wang, 2018).

Recent systematic mapping studies have confirmed that model-based testing is the most common research approach in mobile GUI testing. That test case generation and automated exploration remain the most active research topics (Kousar and Javed, 2025; Nie et al., 2023). With the growing number of testers entering the field of Mobile GUI Testing and the continuous development of new testing tools and techniques, accessible, organized information on real-world practices is crucial for both practitioners and researchers.

3 Methodology

Despite the acknowledged importance of Mobile GUI testing, a significant knowledge gap regarding it remains. To address this gap in the literature, our work reports on the results

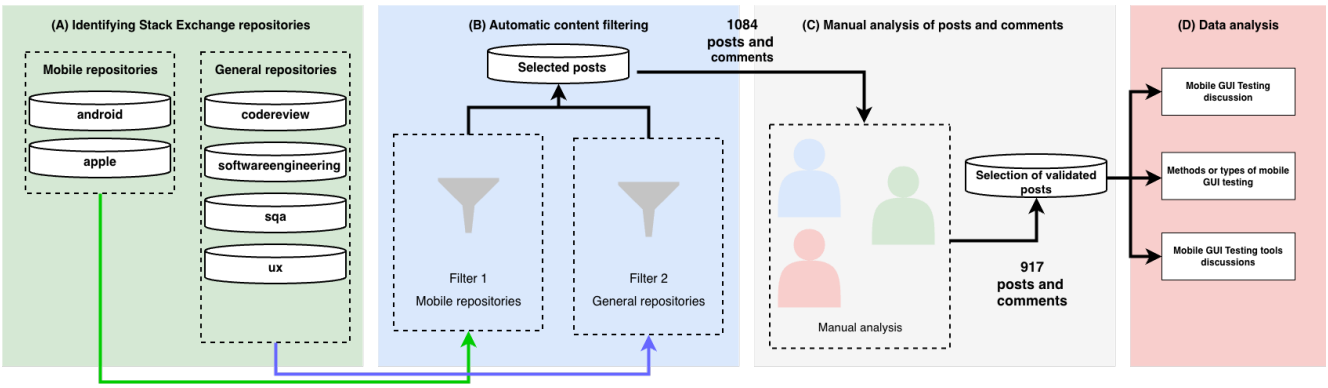


Figure 1. Overview of the study design workflow.

of an empirical study aimed at characterizing the practical landscape of Mobile GUI and UI testing by analyzing discussions among practitioners in Stack Exchange communities.

3.1 Research questions

Our research tackles the following research questions (RQs):

RQ1. What are the main challenges and problems when using GUI Testing in mobile-based systems reported by testers and developers in Stack Exchange?

This RQ aims to identify the challenges and issues in mobile GUI testing discussed in posts and comments written by testers and developers. The challenges and issues were grouped by subject field so that the trends they display, which are common to different repositories and applications of mobile GUI testing, would seem clear.

RQ2. What mobile GUI testing methods do user testers report on the Stack Exchange forums?

This RQ investigates many of the approaches discussed and highlights the main tools and techniques used in the mobile GUI testing community, reported by testers and developers in the Stack Exchange discussions. The objective of the question is to determine which UI testing methods professionals use daily and the reasoning behind the use.

RQ3. Do GUI testing methods differ for the most widely used mobile platforms on the market?

This RQ aims to investigate GUI testing methods and processes on the leading mobile platforms, iOS and Android. The focus of this study is to classify the most commonly used methods on each platform and to highlight the differences reported by users in POSTS and comments on Stack Exchange. Mobile development for Android and iOS is heterogeneous by nature, and avoiding this distinction in this study risks yielding data that is not representative of reality.

Figure 1 shows the methodology employed in the study, containing the following main steps of experiment: (A) Selection of Stack Exchange repositories, (B) Automatic content filtering, (C) Manual analysis of posts and comments, and (D) Data analysis.

3.2 Study design

The study was conducted in four sequential phases (**Figure 1**), including *selection of Stack Exchange repositories*, *automatic content filtering*, *manual analysis of posts and comments*, and *data analysis*.

A) Selection of Stack Exchange repositories:

This study uses an official Stack Exchange Data Dump[1] provided by Stack Exchange, which stores data from 2012 to 2023, to retrieve posts and comments on Mobile GUI Testing reported by testers and developers. The Stack Exchange Data Dump provides extensive data on discussions covering various topics across the Stack Exchange network over time.

We categorize them into mobile repositories and general repositories, as follows:

Mobile repositories

- **Android**[2]: A Stack Exchange forum for Android operating system users.
- **Apple**[3]: A Stack Exchange forum for questions about Apple hardware and software.
- **Windows Phone**[4]: A Stack Exchange forum for Windows Mobile enthusiasts and power users.

The selection of Stack Exchange repositories followed a purposive sampling strategy guided by two complementary criteria: (1) topical relevance to mobile development and testing, and (2) coverage of distinct practitioner perspectives that could collectively provide a comprehensive view of mobile GUI testing discussions. Rather than limiting the study to a single forum, we deliberately selected repositories that capture different facets of the mobile GUI testing ecosystem, from platform-specific user communities to specialized quality assurance forums. The repositories were divided into two groups based on their primary scope. The first group comprises mobile-specific repositories. The **Android** forum was selected because Android holds approximately 72% of the global mobile operating system market share, making it the most widely used mobile platform and, consequently, the one with the largest base of developers and testers facing GUI-related challenges. The **Apple** forum was included to capture the iOS perspective, as iOS accounts for roughly 27% of the global market and presents distinct testing challenges due to Apple's stricter development policies and more controlled ecosystem. The **Windows Phone** forum was included

[1] Available at https://archive.org/details/stackexchange
[2] Available at https://android.stackexchange.com/
[3] Available at https://apple.stackexchange.com/
[4] Available at https://windowsphone.stackexchange.com/tour

to ensure coverage of a third mobile platform that, although discontinued, was active during the period covered by the data dump (2012–2023) and could reveal historical testing practices from a different ecosystem. The second group comprises general repositories selected for their thematic connection to software testing and user interface quality. The **SQA** (Software Quality Assurance and Testing) forum is the only Stack Exchange community explicitly dedicated to software testing professionals, automation engineers, and quality assurance experts, making it the most directly relevant general repository for our study. This relevance is reflected in our data: SQA contributed the vast majority of filtered posts (701 out of 842, 83.3%) and comments (189 out of 242, 78.1%), confirming its central role as a hub for testing-related discussions. The **Software Engineering** forum was included because testing is an integral part of the software engineering lifecycle, and broader architectural or methodological discussions in this forum may touch upon mobile UI testing concerns that do not surface in tool-specific communities. The **Code Review** forum was selected because its peer-review format encourages users to share and discuss actual test code, thereby revealing practical patterns in how mobile GUI tests are written and structured. Finally, the **UX** (User Experience) forum was considered due to the strong conceptual relationship between user interface design and GUI testing. However, UX practitioners may not write test scripts; their discussions of interface usability, interaction patterns, and visual consistency are closely aligned with what GUI tests aim to verify. It is worth noting that UX ultimately yielded no posts after manual filtering, a relevant finding indicating that GUI testing discussions are not prominent in the UX design community despite the conceptual overlap.

General repositories

- **Code Review**[5]: A Stack Exchange forum for peer review code. It's part of Q&A Stack Exchange sites and helps developers improve programming skills.
- **Software Engineering**[6]: A Stack Exchange forum for professionals, academics, and students to discuss software engineering subjects. It's part of Q&A Stack Exchange sites.
- **SQA**[7]: A Stack Exchange forum for software quality control experts, automation engineers, and software testers. It's part of Q&A Stack Exchange sites.
- **UX**[8]: A Stack Exchange forum for User Experience Designers and Human-Computer Interaction research. It's part of Q&A Stack Exchange sites.

The Android, Apple, and Windows Phone repositories were chosen so instances of the use of different tools and methods available on each platform would be found, their data assisting in answering RQ3. Testing is a big part of the Software Engineering process, so the Software Engineering forum could have valuable insights when applying the mobile UI filters. The same applies to the SQA repository. In Code review, users' answers are often helpful, and code related to mobile GUI testing is likely to be submitted. UI and GUI are essential parts of the UX; not only that, but many UX developers work in the mobile field, so the UX repository was also considered for the probability of having insights into mobile UI testing.

The Stack Exchange data dumps are in XML format, so the files were saved on computers for later automatic processing, which filtered relevant or potential posts and comments for the study. In total, 1.061.107 posts and 1.747.350 comments went through the filtering process.

B) Automatic content filtering:

This study phase involved automatically and algorithmically analyzing posts from all previously listed Stack Exchange repositories. Since the data dumps are XML files, we created integrated Python scripts to handle tasks of reading, pre-processing, filtering, and saving potential posts and comments. We used the XML file reading library for Python, xml.dom, which has the minidom class for reading the XML files. The keywords used to filter the posts were selected based on prominent terms in the GUI testing literature and a pilot analysis of relevant posts (Júnior and Machado, 2020).

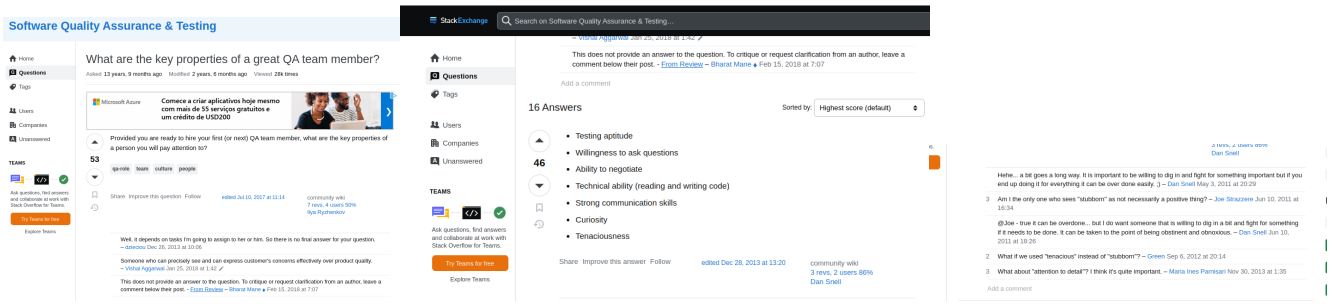
Inside the algorithms, we created two separate scripts for reading posts and comments since the XML tags for posts and comments have different attributes for reading, as shown in Figure 2 (a), Figure 2 (b), and Figure 2 (c). Question-type posts have an attribute called *PostTypeId* with a value of 1, a title, and a description. Answer-type posts have a *PostTypeId* with a value of 2 and only a description, with no title. Comments do not have a title and only have the *Text* tag as text output.

The script followed the filters in Table 4. Separate filters for general repositories were used, as filtering with generic keywords from the subject field, such as "GUI Testing", "UI Testing", and others, returned posts and comments related to GUI and UI Testing platforms, such as Web and Desktop. The study focuses on Mobile UI Testing, so we limited the scope by filtering for general Stack Exchange sites using Filter 02. The more general filter was used for Stack Exchange sites focused on mobile (Filter 01).

The algorithm functions by reading the Posts.xml file following the analyzing_posts.py script and the Comments.xml files with the analyzing_comments.py script. The script uses threads to process the posts. One thread is created to analyze the posts, and another to analyze the comments from each previously specified Stack Exchange site. Thus, using the XML.dom library, the *PostTypeId*, Title, and Body tags are extracted first when the posts file is read. The *PostTypeId* will determine the type of Post and whether it will have a title. Thus, once the post type has been defined, the Body and Title, when present, will be sent to the filtering script. A score is added at the first occurrence of any term related to Mobile GUI Testing in the list. We selected all posts with score > 0; that is, whenever any Mobile GUI Testing terms appeared, the post was selected and saved to a CSV in processed_data folder, named PostsProcessed.csv.

For comments, the logic followed the analysis through the content of the XML Text tag. Selected comments were also saved to a .csv file in the processed_data folder, named CommentsProcessed.csv. The selection of posts with a score

[5]Available at https://codereview.stackexchange.com/
[6]Available at https://softwareengineering.stackexchange.com/
[7]Available at https://sqa.stackexchange.com/
[8]Available at https://ux.stackexchange.com/



(a) (b) (c)

Figure 2. Visual representation of (a) POST Question, (b) POST Answer, and (c) Comment on the SQA Stack Exchange web.

above 0 was chosen considering that a manual analysis would be required at some point. Since our objective aimed to generalize data included on those forums, it was important that the automatic filter could find the largest number of posts related to UI testing as possible. The choice for selecting any posts or comments with a score above 0 could lead to some false positives, like matches with the word ”espresso” being related to coffee and not to the testing tool with that name, but we prioritized completeness and decided to deal with the false positives in the manual phase of filtering.

The results and amount of filtering can be found in Tables 1, 2, and also in the supplementary material[9]. The results are in the processed_data folder with the .csv files, and all posts and comments were filtered in the first phase using the automated script. Around 0.08% of all posts were filtered in this phase and 0.014% of all comments. After automated filtering, the study moved on to manually analyzing the posts and comments filtered by the algorithm. Below, we detail how this manual filtering of the results was carried out, as well as the discussions and the inclusion and exclusion criteria.

Table 1. Number of **posts** after applying automatic and manual filtering phases.

	Posts (#)	Posts filtered automatically (#)	Posts filtered manually (#)
Android	129,815	24	12
Apple	322,218	41	9
Code Review	202,105	38	7
Software Engineering	244,066	30	7
SQA	37,379	701	679
UX	118,115	8	0

Table 2. Number of **comments** after applying automatic and manual filtering phases.

	Comments (#)	Comments filtered automatically (#)	Comments filtered manually (#)
Android	176,703	11	9
Apple	465,947	15	5
Code Review	344,670	3	1
Software Engineering	543,176	19	3
SQA	39,458	189	185
UX	167,925	5	0

We acknowledge that the “score > 0” threshold introduced a considerable number of false positives into the automatically filtered dataset. For instance, the keyword “Espresso” matched posts discussing the coffee beverage rather than the Android testing framework of the same name. Similarly, generic terms such as “UI test” occasionally matched posts about desktop or web UI testing rather than mobile-specific discussions. The extent of this issue is reflected in the manual filtering rates: approximately 84.8% of the automatically filtered posts and 83.9% of the automatically filtered comments were excluded during the manual analysis phase (Tables 1 and 2), confirming that the automated filter produced a high volume of false positives that required human judgment to resolve. This two-phase filtering design, combining a high-recall automated filter with a high-precision manual review, was intentional. A more restrictive automated filter (e.g., requiring a score > 1 or using compound keyword expressions) would have reduced the number of false positives, but at the risk of missing relevant discussions that mention only a single keyword. Given that our study aims to provide a comprehensive characterization of mobile GUI testing discussions, we decided to invest additional effort in manual filtering rather than risk excluding relevant content at the automated stage.

The analysis of excluded items confirmed that the keyword espresso” was the primary source of false positives, triggering 45 out of 56 exclusions (80.4%). In these cases, the term referred to the HTC Espresso phone model, coffee-related discussions, or the macOS dictionary entry for the word, rather than the Android testing framework. Other false positive triggers included UI test” (7 cases, where the discussion referred to desktop or web UI testing) and “ADB” (6 cases, where the tool was used for device management purposes unrelated to GUI testing).

To further illustrate the nature of the false positives captured by the automated filter, Table 3 presents examples of posts that matched the keyword filters but were excluded during the manual analysis phase.

C) Manual analysis of posts and comments:

After the automated filtering phase, we manually analyzed the Posts and Comments selected by the algorithm. The idea was to validate their content and ensure that they fit our study. Three people performed the analysis of the selected posts and comments. Our analysts would thoroughly analyze the CSV files generated in the previous step, and by reading each line that represented a comment or a post, they would determine whether it fit our inclusion criteria in the study. Each file would be reviewed by two different analysts, and if there was

[9]Available at https://figshare.com/s/9bd2320f75f71b6e6a11

Table 3. Examples of false positives captured by the automated keyword filter.

Keyword	Reason for exclusion
"Espresso"	Post refers to the HTC Espresso phone model, not the Android testing framework.
"Espresso"	Post mentions a "USB espresso machine" in a humorous context, unrelated to testing.
"Espresso"	Post discusses the spelling of the word "espresso" in the macOS dictionary, not the testing tool.
"UI test"	Post discusses flashing a custom ROM on a Galaxy Tab, with no relation to GUI testing.

a disagreement between them about any inclusion, a third analyst who had an overview of the process would come and analyze the post or comment for his decision on whether to include it or not.

Finally, the three analysts would collaboratively determine the verdict. Although no formal inter-rater agreement metric was computed, independent dual review followed by consensus resolution helped ensure the reliability of the classification process. The inclusion and exclusion criteria were guided by the keywords and the context presented in the message. The inclusion and exclusion criteria for Posts and Comments used by the reviewers are described below, INCL representing the Inclusion criteria and EXCL representing the Exclusion Criteria:

Inclusion Criteria

- *(INCL A)* Posts (Questions and answers) and Comments that have discussions related to the keywords presented in Table 4;
- *(INCL B)* Posts (Questions and answers) and Comments related to challenges and problems reported in GUI or UI Testing in mobile systems
- *(INCL C)* Posts (Questions and answers) and Comments related to GUI or UI testing methods reported by testers and developers
- *(INCL D)* Posts (Questions and answers) and Comments related to tools and methods included in our filters on Table 4 or new tools and methods used by testers and developers of the most widely used mobile platforms on the market.

Exclusion Criteria

- *(EXCL A)* Posts (Questions and answers) and Comments not written in English
- *(EXCL B)* Posts (Questions and answers) and Comments that do not have discussions semantically related to the topics and keywords presented in Table 4;
- *(EXCL C)* Posts (questions and answers) and comments that do not relate to challenges and issues reported in GUI or UI testing on mobile systems
- *(EXCL D)* Posts (questions and answers) and comments that do not feature GUI or UI testing methods reported by testers and developers

Example of Included Post:

Table 4. Filters are used to automatically process and select posts and comments on Stack Exchange mobile and general sites.

	Mobile Stack Exchange sites (Filter 01)	General Stack Exchange sites (Filter 02)
Subject	"GUI testing", "UI testing", "User Interface testing", "Automated UI testing", "GUI test", "UI test", "user interface test", "automated ui test"	"Mobile UI testing", "Mobile app testing" "End-to-End testing", "End-to-End test", "Mobile UI test", "Mobile app test"
Main types of GUI testing	"End-to-End test", "Mobile UI test", "Mobile app test", "Android UI testing", "iOS UI testing", "golden test"	"Android UI testing", "iOS UI testing", "golden test"
Main tools	"Appium", "Espresso", "XCUITest"	"Appium", "Espresso", "XCUITest"

"Allow USB debugging" keeps on popping up

We are doing automation testing on mobile devices with Appium, but the testing fails because of the popup "Allow USB debugging?" even though the device is already connected through `adb connect` and `adb devices`.

`https://i.stack.imgur.com/Aio8X.png`

I already checked the "Always allow from this computer", but still, sometimes it pops up and causes the test to fail, and other times it works just fine; not popping up at all.

How to address this issue?

Reason for inclusion:

- Active discussion on the Appium tool for mobile UI testing.
- "Appium" triggered the automatic filter and brought the post to manual review.

Example of Excluded Post:

How come I get "This is a 'espresso10wifi"' when I try to flash recovery?

I'm trying to `adb-sideload` a new recovery (Philz Touch Recovery n5110) and I get:

```
This package is for a
'konowifi.5110.GT-N5110.konawifixx'
devices: this is a 'espresso10wifi'.
E: Error in /tmp/update.zip
(Status 7)
Installation aborted.
```

Reason for exclusion:

- No discussion related to mobile UI testing.
- The post was captured by the filter only because it contains the word "espresso", which is also the name of a testing tool.

84.8% of posts that went through the automatic filtering were filtered during the manual filtering phase, for comments

it was 83.9%. Those values show that although the automatic filters used were effective, the manual filtering was essential.

D) Data analysis:

The data analysis combined qualitative and quantitative approaches, adapted to the nature of each research question. For all three RQs, the qualitative component followed the principles of Thematic Analysis as described by Braun and Clarke (Braun and Clarke, 2006), which consists of six phases: familiarization with the data, generation of initial codes, searching for themes, reviewing themes, defining and naming themes, and producing the report. This method was chosen for its flexibility and suitability for identifying patterns in textual data without requiring a pre-existing theoretical framework, which aligned with the exploratory nature of our study. For **RQ1**, the analysis was primarily qualitative. During the familiarization phase, the three analysts independently read all validated posts and comments. In the coding phase, each analyst assigned descriptive codes to the challenges and problems reported by practitioners (e.g., ”pop-up interrupting automation”, ”emulator configuration failure”, ”OS update breaking tests”). These initial codes were then compared across analysts and iteratively grouped into subcategories through discussion. The subcategories were further consolidated into four overarching thematic categories through consensus: (1) Tooling and Automation Limitations, (2) Configuration and Environment Issues, (3) UI Interaction and Test Execution Issues, and (4) Functional and Integration Issues. To complement the qualitative findings, we also quantified the frequency of posts and comments associated with each category.

For **RQ2**, the analysis adopted a mixed-methods approach combining qualitative coding with quantitative frequency analysis. Qualitatively, the analysts identified and coded mentions of specific testing tools, techniques, and strategies in each validated post and comment. Quantitatively, we counted the number of items mentioning each tool or technology to determine adoption frequency. This dual approach allowed us to not only identify which tools practitioners discuss, but also to quantify their relative prominence in the community.

For **RQ3**, the analysis followed the same thematic analysis process used in RQ1, applied separately to posts and comments related to the Android and iOS ecosystems. Posts were first classified by platform based on their content (Android-specific, iOS-specific, or cross-platform). Then the challenges, tools, and strategies identified in RQ1 and RQ2 were compared across platforms to identify patterns of similarity and difference.

D) Classification Validation:

To assess the reliability and consistency of the manual classification process, we conducted an inter-rater agreement evaluation following established practices in empirical software engineering research (Cruzes and Dybå, 2011; Seaman, 1999). A random subset of 59 posts from the automatically filtered dataset was selected for independent classification by all three analysts. Each analyst received the same set of posts and independently applied the inclusion and exclusion criteria defined in this study, recording a binary decision (include or exclude) for each post without consulting the other analysts or the final classification used in the study.

The inter-rater agreement was measured using two complementary metrics. First, Cohen’s Kappa coefficient (Cohen, 1960) was computed for each pair of analysts to quantify the level of agreement beyond chance alone. Second, Fleiss’ Kappa (Fleiss, 1971) was computed to capture the overall agreement among all three analysts simultaneously. Both metrics were interpreted according to the scale proposed by Landis and Koch (Landis and Koch, 1977), in which values below 0.00 indicate poor agreement, 0.00–0.20 slight, 0.21–0.40 fair, 0.41–0.60 moderate, 0.61–0.80 substantial, and 0.81–1.00 almost perfect agreement. After the independent classification, all disagreements were discussed among the three analysts in a consensus meeting. During this meeting, each disputed post was revisited, the analysts presented their reasoning, and a final collective decision was reached. This consensus-based resolution process ensured that all items included in the final dataset reflected a shared understanding of the inclusion criteria among the research team. The agreement results are summarized in Table 5.

Table 5. Inter-rater agreement results for the manual classification of posts.

Pair	Agr.(%)	κ	Level
A1 vs A2	84.7	0.673	Substantial
A1 vs A3	86.4	0.651	Substantial
A2 vs A3	78.0	0.501	Moderate
Avg. pairwise	83.1	0.608	Substantial
Fleiss’ (all 3)	–	0.600	Mod./Subst.

The pairwise Cohen’s Kappa values ranged from 0.501 to 0.673, with an average of 0.608, indicating substantial agreement according to the Landis and Koch (Landis and Koch, 1977) interpretation scale. The Fleiss’ Kappa for all three analysts was 0.600, which sits at the boundary between moderate and substantial agreement. In terms of raw observed agreement, the three analysts fully agreed on 44 out of 59 posts (74.6%).

The analysis of the 15 cases where disagreement occurred revealed a consistent pattern related to differences in the strictness of criteria application rather than fundamental divergences in the understanding of what constitutes mobile GUI testing content. Analyst 2 adopted a more inclusive interpretation of the criteria, classifying 40.7% of the posts as relevant, while Analyst 1 followed an intermediate approach with a 32.2% inclusion rate, and Analyst 3 applied stricter criteria, including only 18.6% of the posts. This variation in strictness is reflected in the Kappa values: the lowest agreement (0.501) was observed between Analyst 2 and Analyst 3, who represent the most inclusive and most restrictive ends of the spectrum, respectively. The highest agreement (0.673) was between Analyst 1 and Analyst 2, who shared a more similar threshold for inclusion. Among the 15 disagreements, 9 followed a pattern where two analysts excluded the post, and one included it, while 6 followed the opposite pattern. In all cases, the disagreements involved posts that were tangentially related to mobile GUI testing, such as posts discussing device configuration issues that could affect test execution but did not explicitly address testing practices. These borderline cases were resolved through the consensus meeting, where the analysts collectively evaluated each disputed

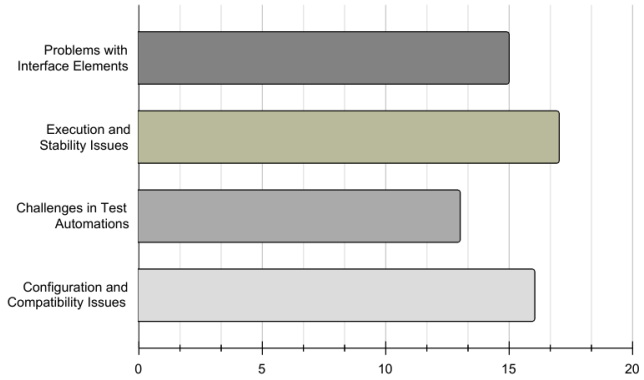


Figure 3. Four major categories of discussions on Stack Exchange forums

item against the inclusion criteria and reached a final decision. The substantial average agreement indicates that the analysts applied the criteria consistently across the dataset, and the consensus resolution procedure ensured that borderline cases were handled through collective deliberation rather than individual judgment.

4 Results

After filtering and analyzing data from 714 Posts and 203 Comments from 6 Stack Exchange sites, reported by testers and developers, we present the collected and analyzed data as answers to the research questions (RQs) proposed by this study.

RQ1. What are the main challenges and problems when using GUI Testing in mobile-based systems reported by testers and developers in Stack Exchange?

Motivation:

Mobile GUI test automation presents frequent and complex challenges. The motivation for this research question (RQ1) was to identify and categorize the primary issues and challenges discussed by testers and developers on Stack Exchange forums, thereby gaining a deeper understanding of the current landscape of the field.

Approach:

To answer this question, we analyzed posts and comments from different Stack Exchange sites related to mobile GUI testing. The identified challenges were grouped by theme to clarify the trends, and then summarized into four main categories, as shown in **Figure 3** of the article.

Findings:

According to the analysis of posts and comments from the different Stack Exchange sites, nine main challenges reported by testers and developers were identified and grouped into four thematic categories, as shown in Figure **??**. Below, we describe each category and illustrate it with representative excerpts from the analyzed posts.

1. **Tooling and Automation Limitations**

This category encompasses challenges related to the capabilities and usability of testing tools. Two main subcategories emerged from the data.

(a) **Limitations in the automation of some tests:** Several posts reported frustration with the limited capabilities of testing tools when dealing with specific UI elements. For example, one practitioner described a performance bottleneck caused by a dynamic UI element that the testing tool could not handle efficiently:

> *"I have automated a production app (don't have control over its source code) using Appium but in an attempt to speed up this automation I would like to remove a dynamic animated element [...] from UI which is slowing down the UI query response time."* (Post 234140, SQA)

This example illustrates a recurring pattern in which testers encounter limitations when automating tests for applications with complex or dynamic interface components, particularly when they do not have access to the application's source code.

(b) **Questions about how to run automated testing for UI:** Both beginners and experienced practitioners posted questions seeking guidance on how to set up and execute automated UI tests:

> *"I'm wanna use Xcode and Android Studio with Appium to do Mobile Automation Testing. Then for websites, I'm gonna use Selenium stuff. [...] For the above, which Mac (Spec) is good to buy?"* (Post 361728, Apple)

The presence of such questions suggests that the entry barrier for mobile GUI test automation remains high, even for practitioners who have already chosen their tools.

2. **Configuration and Environment Issues**

This category groups challenges related to setting up the testing environment and ensuring compatibility across devices and operating system versions.

(a) **Configuration and restart problems:** Several posts described difficulties in configuring testing tools and environments, including resource constraints and conflicts between tools:

> *"Along with Appium Server, I need to run the Emulator of Xcode's iOS Emulator and the Android Studio's Emulator. Does this run smoothly in the 8GB RAM of MacBook Air?" (Post 360884, Apple)*

This post illustrates a common configuration concern: running multiple testing environments simultaneously demands significant system resources, and practitioners often struggle to determine the minimum hardware requirements for their testing setup.

(b) **Compatibility between devices and OS versions:** The issues and questions regarding the

availability and usability of testing tools on different systems;

> *"I am trying to run an application which I am developing currently in Android Studio 3.0 using Genymotion emulator with Android 5.0 - API 21. [...] I am not able to run the application. There is no compilation error in application code as it builds successfully." (Post 186576, Android)*

Posts about iOS-specific restrictions were also frequent:

> *P"I am using Appium as a testing tool. Is it required for me to have a developer account – paid – just to be able to test apps on a real device?" (Post 316709, Apple)*

3. **UI Interaction and Test Execution Issues**
This category captures challenges that arise during test execution due to the behavior of the operating system's user interface.

   (a) **UI interference and messages in tests:** Practitioners reported various error messages and UI-level interference that disrupted their testing workflows. These ranged from permission dialogs to system notifications that appeared unexpectedly during test execution, forcing testers to implement workarounds or add explicit handling logic in their test scripts.

   (b) **Unexpected pop-ups interrupting test automation:** The most frequently reported issue in this category involved system-level pop-ups that interrupt automated test flows. Two posts from different forums described the same problem:

   > *"We are doing automation testing on mobile devices with Appium, but the testing fails because of the popup 'Allow USB debugging?' even though the device is already connected through adb connect and adb devices. [...] sometimes it pops up and causes the test to fail, and other times it works just fine."* (Post 207798, Android)

   > *"The 'Allow USB debugging?' popup keeps recurring during test runs [with Espresso]."* (Post 221408, SQA)

   The recurrence of this specific issue across different tools (Appium and Espresso) and forums (Android and SQA) suggests that unexpected system pop-ups represent a systemic challenge that transcends individual tools.

   (c) **Changes in the operating system's user interface that affect test execution: Practitioners also reported that OS-level UI changes can affect test execution:**

   > *"For manual UI testing purposes, on my Android 6.0 and 7.0 devices, I use a shortcut to access language preferences via ADB. [...] Is it possible to automate the task of changing the system locale (without root) via ADB?"* (Post 182676, Android)

   This example shows how internationalization and localization testing require interaction with system-level settings that are outside the scope of the application under test.

4. **Functional and Integration Issues**

   (a) **Problems in ensuring the correct functionality of the interface through tests:** Some posts described challenges in testing advanced interface features that depend on hardware sensors:

   > *"I'm looking for a way to set the x, y, and z values for the accelerometer sensor on an Android emulator for tests using Espresso. My test requires that the device be positioned in a specific way."* (Post 233530, SQA)

   (b) **Errors detected in the underlying functionality that affect the user interface:** When other parts of the system end up interfering with the UI.

   > *"I can recommend to use instead of Appium's app the FakeTraveler. It has not the disadvantage to run a lot of adb permission commands, which not really work out of the box."* (Post 225393, Android)

   This example reveals a practical pattern in which testers actively seek alternatives to established tools when the default approach proves unreliable. Table 6 provides a consolidated view of the coded excerpts mapped to each category and subcategory.

Problems and challenges in automating GUI and UI tests for mobile devices are frequent, as most frequently asked questions and answers are reported to address doubts about these issues. Some discussions are very long, and sometimes the problems are not completely resolved, as **Table 8** shows.

To illustrate how posts were coded into the four main categories, Table 6 presents representative excerpts from the analyzed posts, each mapped to its corresponding category and subcategory. The coding process followed an inductive approach: the three analysts independently read each validated post and assigned descriptive codes based on the post's main topic. These codes were then grouped into subcategories through iterative comparison, and the subcategories were further consolidated into the four overarching categories shown in Figure **??**. For example, posts describing pop-ups interrupting automated tests and posts reporting locale changes breaking test scripts were initially coded separately but later grouped under the common category of "UI Interaction and Test Execution Issues, as both reflect situations where the operating system's interface behavior interferes with test execution.

RQ2. What mobile GUI testing methods and tools do user testers report on the Stack Exchange forums?

Table 6. Examples of coded post excerpts mapped to the four challenge categories identified in RQ1.

Category	Subcategory	Post excerpt	ID	Source
Tooling and Automation Limitations	Limitations in automation	"I have automated a production app using Appium but in an attempt to speed up this automation I would like to remove a dynamic animated element from UI which is slowing down the UI query response time."	234140	SQA
	Questions about automated testing	"I'm wanna use Xcode and Android Studio with Appium to do a Mobile Automation Testing. […] which Mac is good to buy?"	361728	Apple
Configuration and Environment Issues	Configuration problems	"Along with Appium Server, I need to run the Emulator of Xcode's iOS Emulator and the Android Studio's Emulator. Does this run smoothly in 8 GB RAM?"	360884	Apple
	iOS-specific restrictions	"I am using Appium as a testing tool. Is it required for me to have a [paid] developer account just to be able to test apps on a real device?"	316709	Apple
UI Interaction and Test Execution Issues	Unexpected pop-ups	"We are doing automation testing on mobile devices with Appium, but the testing fails because of the popup 'Allow USB debugging?' even though the device is already connected."	207798	Android
	OS UI changes affecting tests	"For manual UI testing purposes, on my Android devices, I use a shortcut to access language preferences via ADB. Is it possible to automate changing the system locale?"	182676	Android
Functional and Integration Issues	Ensuring correct functionality	"I'm looking for a way to set the accelerometer sensor values on an Android emulator for tests using Espresso. My test requires that the device be positioned in a specific way."	233530	SQA
	Underlying functionality affecting UI	"I can recommend to use instead of Appium's app the FakeTraveler. It has not the disadvantage to run a lot of adb permission commands."	225393	Android

Motivation:

The motivation of RQ2 was to investigate which mobile GUI testing methods, tools, and techniques are discussed and used by the developer and tester community. The goal was to fill the gap in organized information about real-world practices, beyond theoretical comparative studies, and to identify which approaches practitioners find most useful.

Approach:

The approach to answering RQ2 involved analyzing posts and comments from Stack Exchange forums to identify and map the most frequently cited testing technologies, strategies, and techniques. The process involved centrally identifying key tools and methods to understand the adoption trends and discussion patterns in the community.

Findings:

Testers and developers using the forums reported different questions and answers about the methods and tools that were their focus. Table 7 presents the frequency of tool and technology mentions across the validated dataset, complementing the qualitative findings with quantitative data on tool adoption.

Many posts and comments reported using Appium for test automation and testing specific UI elements present in mobile systems. Appium was the most frequently discussed tool, appearing in 65.7% of all included items, which confirms its dominant position in the mobile GUI testing community. The analysis of the posts reveals three main reasons for this popularity: its cross-platform capability, the natural migration path for practitioners with prior Selenium experience, and its integration with CI pipelines as an open-source solution. As one practitioner noted:

> *"I don't really have that much experience aside from using it on Android testing. It is so much easier to set it up on Windows and Android. But since I have more experience in Selenium, this is the reason [I chose Appium]."* (Comment 402818, Apple)

Table 7. Frequency of tool mentions in validated posts and comments (N=35).

Tool	Items	%
Appium	23	65.7
ADB	13	37.1
Xcode	5	14.3
Espresso	5	14.3
Selenium	4	11.4
XCTest/XCUITest	3	8.6
Android Studio	3	8.6
Calabash	1	2.9
FakeTraveler	1	2.9
UI Automator	1	2.9

In addition, users frequently reported doubts about the tool's features and problems configuring the environment to start their execution, as shown in Table 8. ADB (Android Debug Bridge) was the second most mentioned technology (37.1%), reflecting its central role in Android test automation work-

flows, from device management to test execution and log collection. However, ADB was typically mentioned as an enabling technology rather than a testing tool per se, used in conjunction with frameworks like Appium or Espresso. As for native Android frameworks, users reported mainly using the Espresso (14.3%) and UI Automator (2.9%) tools for UI testing. Many were asking for and giving comparisons of mobile testing tools (Appium and Espresso), TestingBot for mobile testing, and Protractor (end-to-end test), which emulates Appium in mobile browsers. For iOS, Xcode (14.3%) and XCTest/XCUITest (8.6%) were the primary native frameworks discussed. Also reported were posts and comments mentioning "crowd testing" for mobile app testing, using Sikuli with Appium to test GUI, SMS testing, and page inspection for automation with Appium as testing strategies. Issues on specific UI Testing Techniques inside the tools were found in the analysis of Posts and Comments, such as the use of sendKeyEvent in Appium, API automation, code to automatically start sessions, questions about hybrid automation frameworks, automation of hybrid applications, strategies for automating text input and side-scroll tests in mobile UI, interface testing based on sensors and physical interactions, and automation of sensor changes in Android emulators.

RQ3. Do GUI testing methods differ for the most widely used mobile platforms on the market?

Motivation:

The motivation for RQ3 came from the recognition that mobile device development requires different approaches. The objective was to study and classify the GUI testing methods and tools specific to each of the main mobile platforms, highlighting the differences reported by the community to avoid obtaining data that is not representative of the reality of each ecosystem.

Approach:

The approach consisted of analyzing and classifying separately posts and comments from Stack Exchange forums that referred to the Android or iOS ecosystem. The focus was to identify tool usage patterns and catalog the challenges and procedural differences reported by developers on each platform, to recognize patterns.

Findings:

The quantitative analysis of platform focus in the validated dataset revealed that Android-related discussions predominated, with 51.4% of included items focusing specifically on the Android ecosystem. iOS-specific discussions accounted for 11.4%, while cross-platform discussions (mentioning both Android and iOS) represented another 11.4%. The remaining 25.7% of items were platform-neutral, discussing general mobile testing concepts without referencing a specific operating system. This distribution is consistent with Android's larger market share and the wider availability of open-source testing tools for the Android platform.

Android testing ecosystem. According to testers and developers, the primary tools for automated testing of mobile UI on Android are Appium, Espresso, UI Automator, and Selenium. The Android ecosystem offers greater flexibility for testing due to its open-source nature and the availability of ADB for device interaction without restrictive licensing requirements. However, this flexibility comes with challenges related to device fragmentation, as one practitioner reported:

> *"I am trying to run an application which I am currently developing in Android Studio 3.0 using Genymotion emulator with Android 5.0 - API 21. [...] I am not able to run the application. There is no compilation error in the application code as it builds successfully."* (Post 186576, Android)

The diversity of Android emulators (Genymotion, Android Studio's built-in emulator, and others) and the range of API levels that must be supported create a fragmented testing environment that is unique to the Android platform. iOS testing ecosystem. The main tools used for automated testing of mobile UI on iOS are XCode UI Test, XCTest, and Appium (for iOS). Another tool that helps in conducting tests is FakeTraveler (fake location test), for testing location and widgets that use location (Maps, pins, and others). A distinctive characteristic of the iOS testing ecosystem, as reported by practitioners, is the additional configuration overhead imposed by Apple's policies. The following exchange illustrates this challenge:

> *"I am very new to iOS – I am an Android person, and I have been given an app to test which is already installed on our iPhone. I am using Appium as a testing tool. Is it required for me to have a developer account – paid – just to be able to test apps on a real device?" (Post 316709, Apple)*

> *"You don't need a paid developer account to test on a real device. You can develop apps and deploy them to your own real device without paying Apple. You only need a paid account if you want the app in the App Store." (Post 316711, Apple)*

This exchange reveals a common source of confusion for practitioners transitioning from Android to iOS testing: Apple's developer account requirements introduce a configuration barrier that does not exist in the Android ecosystem. Practitioners also reported that iOS testing often requires managing device state through backups and configuration profiles:

> *"We have hundreds of devices running UI and unit tests. The tests are written using XCTest framework. [...] As part of the testing framework, we need to bring devices to a certain state before running each test. This includes clearing caches, changing setup items, and installing applications." (Post 343388, Apple)*

Key differences between platforms. The differences between mobile UI testing across operating systems were discussed in several posts, and the analysis identified three main differences between Android and iOS testing practices. First, the tooling is largely platform-specific: while Appium serves as a cross-platform bridge, native frameworks (Espresso/UI Automator for Android, XCTest/XCUITest for iOS) are preferred for platform-specific testing, and users reported that in some instances Android and iOS use entirely different tools for UI testing. Second, iOS requires additional configuration for test automation due to Apple's policies regarding developer accounts, device provisioning, and code sign-

Table 8. Summary of challenges, testing strategies, and platform-specific issues reported on Stack Exchange forums.

RQ	Category	Subcategory
RQ1	Challenges/Problems	Compatibility between devices and OS versions
		UI interference and messages in tests
		Limitations in the automation of some tests
		Configuration and restart problems
		Ensuring correct functionality of the interface through tests
		Errors in underlying functionality affecting the UI
		Unexpected pop-ups interrupting test automation
		Changes in the OS UI affecting test execution
		Questions about how to do automated UI testing
RQ2	Testing Methods and Strategies	Appium to test specific UI elements
		Native Android frameworks (Espresso and UI Automator)
		Comparison between Appium and Espresso
		TestingBot for mobile testing
		Protractor for mobile browsers
		Crowd testing for mobile apps
		Sikuli with Appium for GUI testing
		SMS testing
		Page inspection for automation
		Hybrid automation frameworks
		API automation
		Text input and side-scroll automation strategies
RQ3	Tools for Automated Testing in Android	Appium
		Espresso
		UI Automator
		Selenium
	Tools for Automated Testing in iOS	XCode UI Test
		XCTest
		Appium (for iOS)
	Auxiliary Tools	FakeTraveler (fake location test)
	Differences in Tools and Frameworks	Android and iOS use different tools for UI testing
		iOS requires additional configuration for test automation
		Difficulties running tests on iOS devices due to Apple policies

ing, which were not reported as challenges in Android discussions. Third, auxiliary tools differ between platforms: Android testers frequently use ADB for device interaction and FakeTraveler for location testing, while iOS testers rely on libimobiledevice and Apple Configurator for device management.

To provide a consolidated and easily referenced view of all these findings, Table 8 summarizes the information collected. It organizes the main challenges reported (RQ1), the testing strategies and tools discussed (RQ2), and the specific issues for each platform, Android and iOS (RQ3), providing a complete overview of the data analyzed in this study.

5 Discussions

This study provides actionable insights for both practitioners and researchers. For practitioners, the categorized challenges and reported tool limitations serve as a reference to anticipate common pitfalls during mobile GUI testing and to choose more suitable tools based on project needs. When analyzing Stack Exchange posts and comments through our methodology presented in figure 1, we found the main challenges that testers encounter when creating and executing tests, as we noticed that:

Table 9 shows the temporal distribution of validated items. The majority of relevant discussions (71.4%) were concentrated between 2016 and 2019, with a peak in 2018 (10 items, 28.6%). This period coincides with the maturation of mobile testing frameworks such as Appium 1.x and the release of Espresso as part of Android Testing Support Library. Discussions before 2016 were scarce (5.7%), suggesting that mobile GUI testing was not yet a prominent topic in the selected Stack Exchange communities during that period. A decline is observed from 2020 onward (22.9%), which may reflect the general decrease in Stack Exchange activity reported in recent years or a shift toward other platforms for testing discussions.

Table 9. Temporal distribution of validated items by year.

Year	Posts	Comments	Total
2011	0	1	1
2013	0	1	1
2016	3	1	4
2017	3	2	5
2018	4	6	10
2019	4	2	6
2020	3	0	3
2021	2	1	3
2022	1	0	1
2023	1	0	1
Total	**21**	**14**	**35**

Main challenges in mobile GUI testing are device compatibility and unexpected interference:

Analysis of the reports indicated that the main challenges testers and developers face when testing mobile GUIs are compatibility between devices and unexpected UI interference. The many reports of pop-ups interrupting automation and inconsistent behavior between Android and iOS versions show that ensuring test stability remains an important concern. Issues such as unexpected reboots and difficulty automating tests without manual interaction were mentioned, supporting the complexity of GUI testing.

This finding strongly corroborates the existing literature. Nie et al. (2023), in their systematic mapping study of GUI testing on mobile apps, identified that the dynamics of mobile platforms, including new development concepts and interaction gestures, make GUI testing a particularly difficult task. Similarly, Yu et al. (2025), in a recent survey on vision-based mobile app GUI testing, explicitly refers to the "fragmentation problem" of mobile platforms as a fundamental obstacle, noting that GUI testing must be adapted to completely different environments with different implementations and system support.

The device fragmentation challenge has also been recognized by Zein et al. (2016), who identified fragmentation, compatibility, and testing as the most frequently reported challenges for native mobile applications. Our findings provide empirical practitioner-side evidence that confirms what these academic studies have described from a research perspective: device compatibility is not merely a theoretical concern but a daily, recurring obstacle that testers actively seek help to overcome in community forums. However, while the academic literature tends to frame fragmentation as a broad, systemic issue, our analysis reveals the specific, granular ways in which it manifests in practice, such as pop-ups triggered by USB debugging permissions, emulator incompatibilities with specific API levels, and OS-level locale changes that break test scripts.

Appium leads the way in mobile UI testing, but the use of different native tools for Android and iOS reflects different testing methods:

The Appium tool is the leading choice for automated UI testing on mobile devices. In addition, native Android tools, such as Espresso and UI Automator, were frequently mentioned, while on iOS, XCTest, and XCode UI Test are the most used frameworks. The variety of tools reinforces the

differences in testing methods between Android and iOS. Sensor-based automation, such as touch simulation, device rotation, and sensor interaction, was also pointed out as a relevant strategy for advanced testing. But why is Appium the leading tool? Appium is a fast and simple-to-use testing tool that users of the Selenium tool on the desktop were instantly drawn to. It also features cross-platform capabilities along with integration with testing frameworks and Continuous Integration (CI) tools, all in an open source application. The dominance of Appium in our dataset is consistent with findings from multiple academic studies. Wang and Wu (2019) conducted an empirical evaluation of Appium and concluded that it is more efficient in the realization of mobile automation testing compared to alternative tools. Silva and Santos (2023), in a comparative analysis of mobile testing tools, found that Appium offered the most comprehensive support for multiple platforms among the analyzed tools, confirming its position as the preferred cross-platform solution. Berihun et al. (2023), in a systematic literature review on automated testing frameworks, also identified Appium as the most widely discussed tool in the academic literature from 2016 onward. Our study complements these research-oriented assessments by revealing the practitioner perspective: the reasons testers actually choose Appium in their daily work, including familiarity with Selenium, ease of setup on specific platforms, and integration with existing CI pipelines, are practical motivations that are rarely captured in tool comparison studies. This contrast between academic evaluations (which tend to focus on technical features and performance metrics) and community discussions (which emphasize usability, learning curve, and ecosystem integration) highlights the value of mining practitioner forums as a complement to traditional comparative research.

Challenges in GUI testing directly impact the user experience (UI/UX), not just the technical functionality of the application:

The work also emphasizes that the challenges faced in GUI testing directly impact the user experience (UI/UX). The high number of reports of failures caused by unexpected pop-ups, permissions that interfere with testing, and issues with internationalization and localization demonstrates that, in addition to functionality, tests need to consider the actual usability of applications. Testers and developers are increasingly striving to ensure that test automation verifies that the interface works correctly and that the end user will have a fluid and uninterrupted experience. Indika et al. (2024) recognized many challenges of accessibility, while the ones we found relate to testing for testers, which could then lead to the issues they found.

This connection between GUI testing challenges and user experience has been increasingly recognized in the literature. Banerjee et al. (2013) noted that GUI testing literature has historically emphasized testing a system's functionality through its GUI rather than testing the visual or experiential aspects of the GUI itself. Our findings suggest that practitioners are ahead of the academic curve in this regard: the community discussions we analyzed frequently frame testing challenges in terms of their impact on the end user's experience, not merely on functional correctness. This observation aligns with the work of Indika et al. (2024), who found that accessibility challenges in mobile apps, while distinct from GUI testing issues, share the common thread of affecting the end user's ability to interact with the interface. The convergence of these findings suggests that future research on mobile GUI testing should adopt a broader perspective that encompasses not only functional verification but also usability, accessibility, and experiential quality.

The flexibility of testing on Android contrasts with the restrictions of iOS, presenting complexity and adaptation of strategies for each system:

The data collected about Android and iOS in testing methods revealed that, while Android offers more flexibility for testing, iOS has stricter restrictions. The different challenges reinforce the complexity of automation for cross-platform applications, requiring teams to adapt their strategies according to the limitations and tools available for each system. The analysis in this work provides a complete overview of the challenges and solutions adopted by the GUI testing community on mobile devices, highlighting trends and difficulties that still need to be overcome. The data we found regarding Android could complement the work of Villanes et al. (2017), which covers overall testing in the Android platform, and used StackOverflow, a repository not included in this work.

This platform asymmetry is well documented in the literature, though from a different angle. Arnatovich and Wang (2018), in their systematic literature review of automated GUI testing techniques for mobile applications, noted that the majority of research focuses on Android due to its open-source nature and the availability of instrumentation tools, while iOS testing receives comparatively less attention. Our findings from the practitioner community mirror this imbalance: 51% of the discussions in our dataset focused on Android, compared to only 11.4% on iOS. However, our analysis adds a dimension that is absent from the academic literature: the specific nature of the barriers that iOS imposes on testers, such as the confusion around paid developer accounts, the need for device state management through configuration profiles, and the stricter code signing requirements. These practitioner-reported barriers help explain why iOS testing is underrepresented not only in academic research but also in community discussions, as the higher entry barrier may discourage testers from engaging with the platform altogether.

Final considerations relevant to highlight:

The analyses carried out in this article also revealed that frequent pop-ups or OS-related interference help prioritize robustness checks in test scripts. A summary of the findings can be seen in Table 8. For researchers, the findings provide empirical evidence of recurring problems and underexplored testing strategies, including sensor-based interaction and hybrid automation. These insights open avenues for improving tool development, proposing new frameworks, or refining existing techniques to address identified gaps in cross-platform GUI testing.

Overall, our study contributes to the field by bridging the gap between academic research and practitioner experience

in mobile GUI testing. While systematic mapping studies and tool comparison papers provide valuable technical assessments, they often lack the practical, ground-level perspective of the testers who use these tools daily. Conversely, community forums contain rich practical knowledge but lack the synthesis and organization that academic research provides. By systematically mining and analyzing Stack Exchange discussions, this work offers a structured, evidence-based synthesis of practitioner knowledge that complements and, in several cases, corroborates the findings of prior academic studies, while also revealing practical challenges and emerging strategies that have not yet been documented in the research literature.

6 Related Work

Villanes et al. (2017) focus specifically on Stack Overflow and use the LDA algorithm to analyze questions about testing Android applications, covering topics such as testing tools, functional testing, and unit testing. While their study provides an overview of general testing practices in Android development, it is limited to Stack Overflow and employs automated topic modeling. Our study, in contrast, takes a broader, more focused approach: we analyze six Stack Exchange platforms and focus on mobile GUI testing. Also, our methodology is qualitative, enabling a deeper analysis of discussions and challenges that automated techniques often overlook. This approach enables us to identify patterns in how developers and testers interact with GUI testing tools and techniques across platforms. Beyond these methodological differences, while their findings remain at the level of topic clusters identified by the algorithm, our qualitative analysis reveals specific, recurring challenges that practitioners face in daily mobile GUI testing, such as unexpected popups that interrupt test automation, configuration failures that require tool restarts, and OS-level UI changes that break existing test scripts. These granular, practitioner-reported difficulties are not captured by automated topic modeling and represent actionable insights for both tool developers seeking to improve their products and researchers looking to identify underexplored problems in the field. Furthermore, our cross-platform comparison of Android and iOS testing practices shows that iOS requires additional configuration overhead and faces stricter Apple policy restrictions, extending the Android-only scope of prior studies and providing a more complete picture of the mobile GUI testing landscape.

Indika et al. (2024) studied the challenges of mobile app accessibility for users with disabilities, especially the difficulties developers face in implementing accessible features. This study also used Stack Overflow as its primary data source and focuses on accessibility issues in Android and iOS development, such as integrating assistive technologies, accessible UI design, multilingual text-to-speech support, gesture handling, and accessibility testing. While Indika et al. (2024) center their investigation on accessibility as a usability concern, our work focuses on GUI testing as quality assurance. Even though both studies are concerned with interface-related challenges in mobile systems, the nature of the challenges and the objectives differ. Our research seeks to understand how GUI testing is perceived and implemented, providing insights into tools, limitations, and strategies employed by developers and testers across mobile platforms.

Linares-Vásquez et al. (2013) used topic modeling techniques to identify the main topics discussed in mobile development questions on Stack Overflow. The study covers a broader range of topics, such as general development questions, compatibility issues, crash reports, and database connections. In contrast, our work has a more technical focus on GUI testing. While their use of automated topic modeling provides a general view of mobile development issues, our qualitative analysis enables a precise investigation of how mobile developers engage with GUI-specific challenges. Our dataset also includes discussions from multiple Stack Exchange communities, which provides a more diverse and representative set of perspectives on mobile GUI testing. In terms of new knowledge, our study goes beyond confirming known patterns such as Appium's popularity. We document the specific reasons why practitioners favor Appium over native alternatives, including its cross-platform capability, its familiarity for teams already experienced with Selenium, and its integration with CI pipelines, which are practical motivations that do not emerge from topic modeling approaches. More importantly, our analysis uncovers testing strategies that have received limited attention in the academic literature, such as crowd testing for mobile apps, combining Sikuli with Appium for visual GUI verification, and sensor-based automation techniques involving touch simulation, device rotation, and accelerometer interaction. These findings point to emerging practices within the practitioner community that have not yet been systematically studied in research, offering concrete directions for future investigation.

In summary, while previous studies have contributed valuable insights into mobile development discussions on Stack Overflow, they have generally relied on automated topic modeling, focused on broader development concerns rather than GUI testing specifically, or limited their scope to a single platform or forum. Our study complements this body of work by offering three distinct contributions that go beyond confirming established patterns. First, we provide a qualitative, practitioner-driven categorization of nine specific challenges in mobile GUI testing, organized into four thematic categories, that reflect the real difficulties testers encounter in their daily work. Second, we document not only which tools are popular but why practitioners choose them, revealing decision factors such as cross-platform needs, CI integration, and migration paths from desktop testing frameworks. Third, we identify emerging testing strategies, including sensor-based automation and visual verification through tool combinations, that represent underexplored areas where academic research could have a considerable practical impact. Together, these contributions offer a grounded, evidence-based complement to the comparative and tool-focused studies that currently dominate the mobile GUI testing literature.

7 Threats to Validity

In this section, we discuss potential threats to the validity of our study. The main limitations identified in our research re-

late to two central aspects of our methodology: the search process used to collect the data and the generalizability of our findings to other contexts. Below, we detail each of these threats.

Search process. We filtered the discussions related to the Mobile GUI Testing and Mobile UI Testing by an automatic filter as a search using occurrences of texts in the title and description of each post. As testers and developers can use types of texts other than the ones we considered in our search, we may have missed some discussions in exchange for reducing the number of non-relevant discussions returned with the search. In addition, we analyzed relevant posts and comments for each discussion to analyze only the Mobile GUI Testing and UI Testing answers that provided real action or explanation for the problems raised on the Stack Exchange network.

Generalization of findings. There are many technology-based question-and-answer websites on the Stack Exchange network. For this study, our scope focused on the top 6 related websites (Android, Apple, Code Review, Software Engineering, SQA, and UX), encompassing other programming-related subjects. In addition, we applied an automatic search in the title and body of each Post and Comment to limit our analysis to UI Testing and GUI Testing performed in mobile systems based on iOS and Android. Also, the total of posts and comments that went through the manual filtering phase was 917, which is too few to generalize the area as a whole, but it is enough to define the mobile GUI testing scene in the more focused communities we chose.

Sample size and generalizability. The final dataset used in this study comprises 714 posts and 203 comments, totaling 917 items analyzed after manual filtering. We acknowledge that this sample size may raise concerns regarding the generalizability of our findings to the broader mobile GUI testing community. However, several considerations help contextualize this number and support the adequacy of the dataset for this study's objectives.

First, it is important to note that the sample size is not the result of arbitrary selection but rather the outcome of a systematic, multi-layered filtering process applied to a large initial corpus. The automated filtering phase processed over 1,061,107 posts and 1,747,350 comments across six Stack Exchange repositories, and the keyword-based filters retained approximately 0.08% of all posts and 0.014% of all comments. The manual analysis, conducted independently by three analysts, further refined the dataset by removing approximately 84% of the automatically filtered posts and 83.9% of the comments that were false positives or did not meet the inclusion criteria. The resulting 917 items therefore represent a highly curated subset of discussions that are genuinely relevant to mobile GUI testing, rather than a convenience sample.

Second, the relatively small final sample reflects the particularities of the research topic rather than a methodological shortcoming. Mobile GUI testing is a niche subject within a broader software development domain, and unlike general programming topics that generate thousands of questions on Stack Overflow alone, discussions specifically about mobile GUI testing are naturally less frequent and dispersed across multiple communities. The fact that the SQA forum, which is the Stack Exchange community most directly dedicated to software testing, contributed the majority of relevant posts (679 out of 714, or 95.1%) further confirms that discussions on this topic are concentrated in specialized communities and are not widespread across the network.

Third, our study employs a qualitative methodology, with the objective not of statistical generalization but of analytical depth. In qualitative research, the adequacy of the sample is typically assessed by the richness and diversity of the data rather than by the number of items alone (Braun and Clarke, 2006; Seaman, 1999).

Our dataset was sufficient to reach thematic saturation for the main research questions: the nine challenges identified in RQ1, the tools and methods cataloged in RQ2, and the platform-specific differences documented in RQ3 began to recur consistently across posts and comments before all items had been analyzed, indicating that additional data would be unlikely to produce substantially new categories.

Nevertheless, we recognize that a larger dataset, particularly one that incorporates data from Stack Overflow and other developer forums such as GitHub Discussions or Reddit, could reveal additional challenges, tools, or strategies that are not captured in our current analysis. We discuss this as a direction for future work in Section 8.

Finally, although the sample size does not support broad statistical claims about the entire population of mobile GUI testers worldwide, it is comparable to or larger than datasets used in similar empirical studies that mine developer forums. For instance, Villanes et al. (2017) analyzed Stack Overflow posts on Android testing using automated topic modeling, and Indika et al. (2024) studied accessibility challenges using a similar forum-mining approach. Our dataset of 917 manually validated items provides a solid foundation for the qualitative and descriptive analyses presented in this work, and the findings should be interpreted as a characterization of the discussions present in the selected Stack Exchange communities rather than as a definitive account of all mobile GUI testing practices in the industry.

8 Conclusion

Information regarding mobile GUI testing present in literature exists in the form of detailed comparative studies of different methods and tools developed for the task (Bryce et al., 2010; Chaudhary and Sangwan, 2016; Herbold and Harms, 2013). However, there is a lack of studies aimed at finding what working professionals are using and discussing. In this work, we extracted data from Stack Exchange communities to answer important questions that paint a picture of how mobile GUI testing is being practiced daily and what knowledge is being shared among testers online.

To fulfill our objective, we used the official data dumps provided by Stack Exchange and ran a multi-layered extraction of data. First, by selecting the relevant repositories for the research, then by running a filtering algorithm based on keywords, and lastly by manually going through the results and selecting the most important ones. The data we found showed that the main challenges testers have with mobile GUI testing are related to configuration and compatibility is-

sues, problems with interface elements, execution and stability, and challenges in test automation. It also showed that the tools related to mobile GUI testing, most discussed on Stack Exchange, are automated testing tools like Appium for all devices, Espresso for Android, and XCode UI Test for iOS. We also concluded from the data that iOS and Android GUI testers face some challenges of a dissimilar nature from one another.

These findings carry practical implications for different stakeholders in the mobile GUI testing ecosystem. For **tool developers**, the recurring reports of unexpected pop-ups interrupting automation, emulator incompatibilities, and configuration difficulties point to specific areas where existing tools can be improved. The fact that practitioners actively seek workarounds for these issues in community forums suggests that the current tooling does not fully address the real-world conditions under which mobile GUI testing is performed, particularly regarding cross-platform stability and seamless integration with CI/CD pipelines. For **educators and trainers** in Software Engineering, the challenges and tool preferences identified in this study can inform curriculum design. The dominance of Appium in practitioner discussions, combined with the frequent questions from beginners about environment setup and tool selection, suggests that introductory courses on mobile testing should prioritize hands-on experience with Appium and native frameworks (Espresso for Android, XCTest for iOS), with particular attention to the configuration challenges that our data shows are the most common entry barriers for new testers. The platform-specific differences between Android and iOS testing should also be explicitly addressed in educational materials, as our findings show that practitioners transitioning between platforms frequently encounter confusion that could be mitigated through better training.

For researchers interested in mining online developer discussions, our study demonstrates that Stack Exchange communities beyond Stack Overflow contain valuable, specialized knowledge that is often overlooked. The SQA forum, in particular, proved to be a rich source of focused testing discussions. Researchers seeking to understand practitioner perspectives on software engineering topics should consider expanding their data sources beyond Stack Overflow to include these specialized communities, as they capture discussions among dedicated professionals that may not surface in more general forums. Additionally, the methodological approach adopted in this study, combining automated keyword filtering with manual validation and thematic analysis, can be replicated for investigating other software engineering practices through community forum mining.

For future work that wishes to expand on the knowledge generated by this work, there are different paths it could follow. Based on the trends found here, they can focus their analysis on the most popular testing tools and techniques to find more insightful information regarding their use. Future work could also combine knowledge on the topic that is present on Stack Exchange, Stack Overflow, GitHub, and any other online forums where Mobile GUI testing is discussed to create the most complete database on the topic. Other work that could come from the insight present here includes an analysis of the evolution of mobile GUI testing and benchmarking of the different tools studied here. The advent of Large Language Models and Deep Learning may change where developers go to solve their problems and discuss testing, so a work like this, at the transition point, could serve as an overlook on the topic before it, serving as a baseline for what is to come.

Furthermore, the temporal decline in discussions observed from 2020 onward raises an important question for future research: whether practitioners are migrating to other platforms (such as GitHub Discussions, Reddit, or Discord communities) or whether AI-powered assistants are reducing the need for forum-based help-seeking in mobile GUI testing. Investigating this shift could provide insights into the evolving landscape of developer knowledge sharing and its implications for the accessibility of practical testing knowledge.

Artifact Availability

The research package, including the results tables and the scripts employed in this study, is accessible via the supplementary material repository at Figshare: **https://figshare.com/s/d93ed2039a0cfe5dd7d2**

9 Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001; FAPESB grant PIE0002/2022; and CNPq grants 315840/2023-4 and 403361/2023-0.

References

Adamo, D., Nurmuradov, D., Piparia, S., and Bryce, R. (2018). Combinatorial-based event sequence testing of Android applications. *Information and Software Technology*, 99:98–117.

Alomar, E. A., Mkaouer, M. W., Newman, C., and Ouni, A. (2021). On preserving the behavior in software refactoring: A systematic mapping study. *Information and Software Technology*, 140:106675.

Amalfitano, D., Amatucci, N., Memon, A. M., Tramontana, P., and Fasolino, A. R. (2017). A general framework for comparing automatic testing techniques of Android mobile apps. *Journal of Systems and Software*, 125:322–343.

Amalfitano, D., Fasolino, A. R., Tramontana, P., Ta, B. D., and Memon, A. M. (2014). MobiGUITAR: Automated model-based testing of mobile apps. *IEEE Software*, 32(5):53–59.

Ammann, P. and Offutt, J. (2016). *Introduction to Software Testing*. Cambridge University Press, 2nd edition.

Anvik, J., Hiew, L., and Murphy, G. C. (2006). Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 361–370, New York, NY, USA. ACM.

Aranda, J. and Venolia, G. (2009). The secret life of bugs: Going past the errors and omissions in software repositories. In *Proceedings of the 31st International Confer-*

ence on Software Engineering, ICSE '09, pages 298–308, Washington, DC, USA. IEEE Computer Society.

Arnatovich, Y. L., Ngo, M. N., Kuan, T. H. B., and Soh, C. (2016). Achieving high code coverage in Android UI testing via automated widget exercising. In *Proceedings of the 23rd Asia-Pacific Software Engineering Conference*, APSEC '16, pages 193–200. IEEE.

Arnatovich, Y. L. and Wang, L. (2018). A systematic literature review of automated techniques for functional GUI testing of mobile applications. *arXiv preprint arXiv:1812.11470*.

Azim, T. and Neamtiu, I. (2013). Targeted and depth-first exploration for systematic testing of Android apps. *ACM SIGPLAN Notices*, 48(10):641–660.

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.

Berihun, N. G., Dongmo, C., and Van der Poll, J. A. (2023). The applicability of automated testing frameworks for mobile application testing: A systematic literature review. *Computers*, 12(5):97.

Bernardi, M. L., Canfora, G., Di Lucca, G. A., Di Penta, M., and Distante, D. (2012). Do developers introduce bugs when they do not communicate? the case of Eclipse and Mozilla. In *Proceedings of the 16th European Conference on Software Maintenance and Reengineering*, CSMR '12, pages 139–148, Washington, DC, USA. IEEE Computer Society.

Bettenburg, N., Just, S., Schröter, A., Weiß, C., Premraj, R., and Zimmermann, T. (2007). Quality of bug reports in Eclipse. In *Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology EXchange*, ETX '07, pages 21–25, New York, NY, USA. ACM.

Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., and Zimmermann, T. (2008). What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '08, pages 308–318, New York, NY, USA. ACM.

Boehm, B. W. (1981). *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, NJ, USA.

Braun, V. and Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101.

Bryce, R. C., Sampath, S., and Memon, A. M. (2010). Developing a single model and test prioritization strategies for event-driven software. *IEEE Transactions on Software Engineering*, 37(1):48–64.

Cavalcanti, Y. C., da Mota Silveira Neto, P. A., Carmo Machado, I., Vale, T. F., de Almeida, E. S., and de Lemos Meira, S. R. (2014). Challenges and opportunities for software change request repositories: A systematic mapping study. *Journal of Software: Evolution and Process*, 26(7):620–653.

Chaudhary, N. and Sangwan, O. (2016). Metrics for event driven software. *International Journal of Advanced Computer Science and Applications*, 7(1).

Choi, W., Necula, G., and Sen, K. (2013). Guided GUI testing of Android apps with minimal restart and approximate learning. *ACM SIGPLAN Notices*, 48(10):623–640.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46.

Cruzes, D. S. and Dybå, T. (2011). Recommended steps for thematic synthesis in software engineering. In *Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement (ESEM '11)*, pages 275–284. IEEE.

Deng, L., Offutt, J., Ammann, P., and Mirzaei, N. (2017). Mutation operators for testing Android apps. *Information and Software Technology*, 81:154–168.

Farley, D. (2021). *Modern Software Engineering: Doing What Works to Build Better Software Faster*. Addison-Wesley Professional, Boston, MA, USA.

Fleiss, J. L. (1971). Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382.

Genc-Nayebi, N. and Abran, A. (2017). A systematic literature review: Opinion mining studies from mobile app store user reviews. *Journal of Systems and Software*, 125:207–219.

Gomes, F., Santos, E. P. d., Freire, S., Mendonça, M., Mendes, T. S., and Spínola, R. (2022). Investigating the point of view of project management practitioners on technical debt: A preliminary study on Stack Exchange. In *Proceedings of the International Conference on Technical Debt*, TechDebt '22, pages 31–40.

Harrold, M. J. (2000). Testing: A roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, ICSE '00, pages 61–72, New York, NY, USA. ACM.

Herbold, S. and Harms, P. (2013). AutoQUEST: Automated quality engineering of event-driven software. In *Proceedings of the 6th IEEE International Conference on Software Testing, Verification and Validation Workshops*, ICSTW '13, pages 134–139. IEEE.

Herzig, K., Just, S., and Zeller, A. (2013). It's not a bug, it's a feature: How misclassification impacts bug prediction. In *Proceedings of the 35th International Conference on Software Engineering*, ICSE '13, pages 392–401, Washington, DC, USA. IEEE Computer Society.

Humble, J. and Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional.

Indika, A., Lee, C., Wang, H., Lisoway, J., Peruma, A., and Kazman, R. (2024). Exploring accessibility trends and challenges in mobile app development: A study of Stack Overflow questions. *arXiv preprint arXiv:2409.07945*.

International Data Corporation (IDC) (2025). Worldwide quarterly mobile phone tracker — Q4 2025. `https://www.idc.com/promo/smartphone-market-share/`. Accessed: April 2026.

ISO/IEC/IEEE (2022). Software and systems engineering — software testing — part 1: General concepts.

Júnior, J., Boechat, G., and Machado, I. (2021). Label it be! A large-scale study of issue labeling in modern open-source repositories. *arXiv preprint arXiv:2110.01328*.

Júnior, J. M. and Machado, I. (2020). Avaliação empírica de termos técnicos em issues de projetos opensource. *Revista*

Eletrônica de Iniciação Científica em Computação, 18(3).

Junior, N., Costa, H. A. X., Karita, L., Machado, I., and Soares, L. (2021). Experiences and practices in GUI functional testing: A software practitioners' view. In Vasconcellos, C. D., Roggia, K. G., Collere, V., and Bousfield, P., editors, *35th Brazilian Symposium on Software Engineering, SBES 2021, Joinville, Santa Catarina, Brazil, 27 September 2021 - 1 October 2021*, pages 195–204. ACM.

Kanwal, J. and Maqbool, O. (2012). Bug prioritization to facilitate bug report triage. *Journal of Computer Science and Technology*, 27(2):397–412.

Kim, S. and Whitehead, E. J. (2006). How long did it take to fix bugs? In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, MSR '06, pages 173–174, New York, NY, USA. ACM.

Kousar, S. and Javed, A. (2025). A systematic literature review on graphical user interface testing through software patterns. *IET Software*, 2025:9140693.

Landis, J. R. and Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174.

Li, A., Qin, Z., Chen, M., and Liu, J. (2014). ADAutomation: An activity diagram based automated GUI testing framework for smartphone applications. In *Proceedings of the 8th International Conference on Software Security and Reliability*, SERE '14, pages 68–77. IEEE.

Lientz, B. P., Swanson, E. B., and Tompkins, G. E. (1978). Characteristics of application software maintenance. *Communications of the ACM*, 21(6):466–471.

Linares-Vásquez, M., Dit, B., and Poshyvanyk, D. (2013). An exploratory analysis of mobile development issues using Stack Overflow. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 93–96. IEEE Press.

Mahajan, R. and Shneiderman, B. (1997). Visual and textual consistency checking tools for graphical user interfaces. *IEEE Transactions on Software Engineering*, 23(11):722–735.

Mahmood, R., Mirzaei, N., and Malek, S. (2014). EvoDroid: Segmented evolutionary testing of Android apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE '14)*, pages 599–609, New York, NY, USA. ACM.

Mao, K., Harman, M., and Jia, Y. (2016). Sapienz: Multi-objective automated testing for Android applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ISSTA '16, pages 94–105, New York, NY, USA. ACM.

Martins, L., Campos, D., Santana, R., Junior, J. M., Costa, H., and Machado, I. (2023). Hearing the voice of experts: Unveiling Stack Exchange communities' knowledge of test smells. In *Proceedings of the 16th IEEE/ACM International Conference on Cooperative and Human Aspects of Software Engineering*, CHASE '23, pages 80–91. IEEE.

Mazuera-Rozo, A., Trubiani, C., Linares-Vásquez, M., and Bavota, G. (2020). Investigating types and survivability of performance bugs in mobile apps. *Empirical Software Engineering*, 25:1644–1686.

McConnell, S. (1996). Daily build and smoke test. *IEEE Software*, 13(4):144.

McMinn, P. (2011). Search-based software testing: Past, present and future. In *Proceedings of the 4th IEEE International Conference on Software Testing, Verification and Validation Workshops*, ICSTW '11, pages 153–163, Washington, DC, USA. IEEE Computer Society.

Memon, A. M. (2002). GUI testing: Pitfalls and process. *IEEE Computer*, 35(8):87–88.

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

Mockus, A. (2010). Organizational volatility and its effects on software defects. In *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, pages 117–126, New York, NY, USA. ACM.

Myers, B. A. (1995). User interface software tools. *ACM Transactions on Computer-Human Interaction*, 2(1):64–103.

Nascimento, L. P. G., Santos, A., Machado, I., et al. (2024). Issue labeling dynamics in open-source projects: A comprehensive analysis. In *Anais do Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software*, SBCARS '24, pages 51–60. SBC.

Nie, L., Said, K. S., Ma, L., Zheng, Y., and Zhao, Y. (2023). A systematic mapping study for graphical user interface testing on mobile apps. *IET Software*, 17(3):249–267.

Palomba, F., Di Nucci, D., Panichella, A., Zaidman, A., and De Lucia, A. (2019). On the impact of code smells on the energy consumption of mobile applications. *Information and Software Technology*, 105:43–55.

Posnett, D., Warburg, E., Devanbu, P., and Filkov, V. (2012). Mining Stack Exchange: Expertise is evident from initial contributions. In *Proceedings of the 2012 International Conference on Social Informatics*, SocialInformatics '12, pages 199–204. IEEE.

Rajlich, V. (2014). Software evolution and maintenance. In *Future of Software Engineering Proceedings*, FOSE '14, pages 133–144, New York, NY, USA. ACM.

Rosen, C. and Shihab, E. (2016). What are mobile developers asking about? A large scale study using Stack Overflow. *Empirical Software Engineering*, 21:1192–1223.

Sadeghi, A., Jabbarvand, R., and Malek, S. (2017). PATDroid: Permission-aware GUI testing of Android. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE '17, pages 220–232, New York, NY, USA. ACM.

Seaman, C. B. (1999). Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 25(4):557–572.

Silva, G. and Santos, R. S. (2023). Comparing mobile testing tools using documentary analysis. In *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6. IEEE.

Song, W., Qian, X., and Huang, J. (2017). EHBDroid: Beyond GUI testing for Android applications. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, ASE '17, pages 27–37.

IEEE Press.

Su, T., Meng, G., Chen, Y., Wu, K., Yang, W., Yao, Y., Pu, G., Liu, Y., and Su, Z. (2017). Guided, stochastic model-based GUI testing of Android apps. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE '17, pages 245–256, New York, NY, USA. ACM.

Tahir, A., Dietrich, J., Counsell, S., Licorish, S., and Yamashita, A. (2020). A large scale study on how developers discuss code smells and anti-pattern in stack exchange sites. *Information and Software Technology*, 125:106333.

Tassey, G. (2002). The economic impacts of inadequate infrastructure for software testing. Planning Report 02-3, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA. U.S. Department of Commerce.

Villanes, I. K., Ascate, S. M., Gomes, J., and Dias-Neto, A. C. (2017). What are software engineers asking about Android testing on Stack Overflow? In *Proceedings of the 31st Brazilian Symposium on Software Engineering*, SBES '17, pages 104–113, New York, NY, USA. ACM.

Wang, J. and Wu, J. (2019). Research on mobile application automation testing technology based on appium. In *2019 International Conference on Virtual Reality and Intelligent Systems (ICVRIS)*, pages 247–250. IEEE.

Wu, X., Jiang, Y., Xu, C., Cao, C., Ma, X., and Lu, J. (2016). Testing Android apps via guided gesture event generation. In *Proceedings of the 23rd Asia-Pacific Software Engineering Conference (APSEC '16)*, pages 201–208. IEEE.

Xuan, J., Jiang, H., Ren, Z., and Zou, W. (2012). Developer prioritization in bug repositories. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 25–35, Washington, DC, USA. IEEE Computer Society.

Yu, S., Fang, C., Tuo, Z., Zhang, Q., Chen, C., Chen, Z., and Su, Z. (2025). Vision-based mobile app gui testing: A survey. *ACM Computing Surveys*, 58(6):1–46.

Yuan, X. and Memon, A. M. (2007). Using GUI run-time state as feedback to generate test cases. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 396–405, Washington, DC, USA. IEEE Computer Society.

Yuan, X. and Memon, A. M. (2009). Generating event sequence-based test cases using GUI run-time state feedback. *IEEE Transactions on Software Engineering*, 36(1):81–95.

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.

Zubrow, D. (2009). IEEE standard classification for software anomalies. *IEEE Std 1044-2009*.