2023 38th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW) | 979-8-3503-3032-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ASEW60602.2023.00030

An Empirical Study on the Adoption of Scripted GUI Testing for Android Apps

Ruizhen Gu and José Miguel Rojas
Department of Computer Science, The University of Sheffield, Sheffield, United Kingdom
rgu10@sheffield.ac.uk, j.rojas@sheffield.ac.uk

***Abstract*—In recent years, Android applications have become larger with the introduction of new features and integration with external APIs and services. As a consequence, testing these applications is more challenging and time-consuming for developers. To reduce manual testing efforts, several frameworks for automated GUI testing of Android applications have been developed, with Espresso and UI Automator being arguably the most popular ones. However, the adoption of these scripted GUI testing frameworks in Android projects remains surprisingly low.**

We investigate the use of two scripted GUI testing frameworks, Espresso and UI Automator among 475 non-trivial open-source Android applications with GUI tests. We also design and conduct an experiment involving human developers to observe and understand their approach to Android automated testing tasks using these two frameworks, analysing the resulting test code and their perception of the tasks.

Our preliminary results show that 58% of Android apps with GUI tests utilise Espresso APIs while only 4% used UI Automator, 71% of the apps that use UI Automator effectively combine it with Espresso, and novice developers tend to prefer Espresso for GUI testing mainly because of its simpler API while agreeing that the combination of the two frameworks can be beneficial in covering diverse test cases.

***Index Terms*—Testing Mobile Apps, Android, Empirical Study**

I. Introduction

Android, and mobile apps in general, naturally tend to increase in size as they evolve [1]. The quality of mobile applications (or "apps") is a determining factor for their success in mobile app stores (e.g., Google Play, App Store). Consequently, testing these apps before their release is an essential part of the software development lifecycle. Unlike traditional non-graphical programs, Android apps have rich Graphic User Interfaces (GUIs) that require a series of user interactions to achieve specific tasks. Besides the testing of the source code of the apps (e.g., unit testing), the GUIs should also be tested thoroughly to achieve the desired high quality.

Although various automated Android GUI testing frameworks exist which aim to reduce manual effort, evidence suggests that Android developers still rely on expensive and error-prone manual testing [2, 3]. Some efforts have been made for scripted GUI testing for Android apps, e.g., generating Espresso tests from UI interactions [4] and synthesizing Espresso tests from interaction sequences [5]. However, this research topic seems less explored compared to others such as input generation or crash reproduction [6].

Previous studies found that Espresso and UI Automator (https://developer.android.com/training/testing) are the two most popular scripted GUI testing frameworks for Android apps [7, 8]. They are both officially supported, which makes them the most intuitive GUI testing frameworks to choose for app developers. Espresso is recommended for small projects because its API is simple and it has synchronization capabilities developers can rely on to deal with transitions between Android windows (i.e., activities, menus and dialogs), which should lead to fewer lines of test code and good readability. However, Espresso can only perform interactions within the current application under test (AUT). Although UI Automator has cross-app functionality to interact with the system or third-party apps, the test code it yields can be very long and requires developers to manually add `wait` actions to deal with window transitions [7]. The features offered by Espresso and UI Automator make them complementary and their combination is encouraged to cover most use cases [9].

There exists little evidence on the extent to which these UI testing frameworks have achieved their expected levels of adoption by the Android app development community. In this work, we investigate the adoption of scripted GUI tests in existing Android projects. To this aim, we designed and conducted an empirical study consisting of two parts:

- We analysed 475 Android projects containing GUI tests to understand the adoption of Espresso, UI Automator and the combination of the two. The projects are selected from an existing dataset of over 12,000 non-trivial open-source Android projects on GitHub [8].
- We asked six novice Android developers to write UI test cases with Espresso and UI Automator. Our research questions aim to understand how developers approach the testing task, the challenges they encounter and their perception towards the frameworks. Study participants attended an introductory lecture on Android testing and a guided hands-on practice session to familiarise themselves with the task. Participants submitted their test code, a log detailing their activities during the task, and responses to an exit survey capturing their perceptions and perspectives towards the testing task and the frameworks.

Our preliminary results suggest a limited adoption of scripted GUI testing frameworks in open-source projects. Our study with Android developers requires further replications before we can make stronger claims, but we outline emerging conjectures on challenges and opportunities associated with developing test cases with Espresso and UI Automator.

2151-0849/23/$31.00 ©2023 IEEE
DOI 10.1109/ASEW60602.2023.00030

2023 38th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW) | 979-8-3503-3032-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ASEW60602.2023.00030

II. Methodology

The ultimate goal of this study is to gather evidence on the adoption of scripted GUI testing in Android projects, understand the state of practice, identify current challenges and explore opportunities for improvement.

A. Research Questions

RQ1: *What are the adoption rates of Espresso and UI Automator in open-source Android projects?* With this research question, we investigate the adoption of the two most popular Android GUI testing frameworks (Espresso and UI Automator) in open-source projects, their frequency of usage, and the testing scenarios where developers combine them.

RQ2: *How effective are Android developers at writing test cases using Espresso and UI Automator?* We examine the test code to explore potential correlations with the limited adoption of GUI testing frameworks, such as the quality, effort being made and the maintainability of the test code. We also assess the test code to evaluate if the developers have a clear understanding of the practices of the testing frameworks.

RQ3: *How do Android developers perceive the task of writing test cases using Espresso and UI Automator?* We gather insights from the developers' aspects to understand the challenges and limitations they encounter when using scripted GUI testing in their developments. We also observe if the developers tend to prefer one framework over another and the reasons behind their choices.

RQ4: *What are the challenges and opportunities in Android scripted GUI testing?* Based on the observation from the adoption of the two scripted GUI testing frameworks and the study participants' perspectives, we identify existing challenges for Android developers and discuss potential improvements.

B. Use of GUI Testing Frameworks in Open-Source Projects

To address *RQ1*, we identified the two most popular Android GUI testing frameworks, Espresso and UI Automator, as the study subjects. We used a dataset [8] containing 12,562 non-trivial, real-world Android apps to identify the apps that use Espresso, UI Automator or a combination of both.

C. User Study

The goal of the user study is to gain insights into how developers tackle automated testing tasks using the provided testing frameworks (*RQ2-4*). The study consists of two in-person sessions: a guided training session for participants to come to grips with Android GUI testing and the selected frameworks, and a main session where participants are asked to complete a GUI scripted testing task independently.

1) Participants: The potential participants of the study are software developers with experience in Android app development who may or may not have experience with scripted GUI testing frameworks. We aim to recruit participants with a range of experience using the frameworks, from university students without any prior experience to seasoned professional developers with years of Android testing experience.

For the initial study, we recruited six Computer Science MSc students from The University of Sheffield who had passed the *Software Development for Mobile Devices* module, are familiar with Android app development, but have little prior GUI testing experience. As an incentive [10], participants of the study were compensated with a £30 Amazon vouchers for their time attending both sessions. We aim to recruit 20 to 30 participants altogether by means of replications.

2) Target Apps: Expresso and UI Automator have complementary feature sets, e.g., UI Automator is particularly appropriate for cross-app interactions and custom view component interactions. To address our research questions, it was essential to identify apps where these types of interactions exist. Among the candidate features (e.g., maps or social media interactions), we identified the use of the system calendar as a natural, intuitive choice. Therefore, we searched the F-Droid repository (https://f-droid.org/) for apps with the permission of *Read calendar events and details* enabled, which indicates these apps are likely to implement interactions with the system *Calendar* app. We selected two suitable apps for the study, *ShiftCal* (https://gitlab.com/Nulide/ShiftCal) and *DroidShows* (https://github.com/ltGuillaume/DroidShows). The suitability criteria are: apps with multiple testable features which do not require hardware (e.g., wearables), registration or login and are not home-screen widgets. Both applications were upgraded to use Android 8 and Gradle as build tool. *ShiftCal* was extended with easy-to-test CRUD features for training purposes.

3) Training: We designed a training session to set a baseline understanding of the testing frameworks under study among participants. The session starts with an introductory lecture on automated Android GUI testing using Espresso and UI Automator. A walkthrough and reference materials of the essential APIs of both frameworks are also covered. The lecture is followed by a practical session, in which participants familiarised themselves with both testing frameworks through a guided task (similar to Ardito et al. [11]). Participants were provided with an open-source Android app *ShiftCal* and were asked to develop one test case following step-by-step text instructions. Each instruction step corresponds to one or multiple actions and requires a small number of lines of test code to implement. In this training session, the framework to be used to accomplish each step was prescribed to participants, in order to provide participants with hands-on experience with both frameworks. The lecture lasts 30 minutes and the practical session takes one hour. The research team provided assistance and guidance during the practical session.

4) Main task: In the main session, participants are tasked to develop three test cases for the open-source *DroidShows* app. Participants are asked to work independently and without help from the research team. The testing frameworks to be used for each step are not prescribed and participants are encouraged to use their best judgement to accomplish the task. Brief textual descriptions and video instructions are given in this session instead of step-by-step instructions. The video instructions consist of the recording of the actions to be performed by the test cases and the text descriptions of the actions as

subtitles. The overall time for the main session, including test development and survey is 1.5 hours. It is important to mention that the test cases designed for the apps for both sessions involve actions that can only be performed by UI Automator.

5) Data Collection: Three artefacts are elicited from participants: test code, activity logs, and answers to an exit survey.

a) Test Code: Participants are asked to export their Android projects and place them in a given shared drive. The test code will be analysed in terms of coverage and other quality and maintainability metrics. A qualitative analysis of the use of the GUI testing frameworks will be conducted and mutation analysis will be used to assess test effectiveness [12].

b) Developer Activities: To observe developers' behaviour when writing GUI test cases, we designed an activity log for participants to record their activities every 10 minutes in the sessions using a problem-solution cycle approach [13]. The activities are classified as *Understanding Tasks*, *Searching Solutions*, *Applying/Refining Solutions* and *Others*. The *Applying/Refining Solutions* category is split into two options Espresso and UI Automator to reflect the differences between the efforts made using the two frameworks during the task. Participants were asked to select the activity that dominates their behaviour and indicate the level of confidence about their actions in the last 10 minutes. While the activity log was used in both training and main session, only the activity log from the main study session is used for analysis. The template of the activity log can be found in our supplementary materials[1].

c) Survey: After finishing their testing task, participants are required to answer an exit survey. It covers demographic questions related to the development and testing experience and the participants' perspectives towards tackling the tasks with Espresso and UI Automator. The full survey can be found in our supplementary materials[1].

6) Pilot: To make sure the objectives, materials and instructions of the study are clear, we conducted two pilot sessions with computer science PhD students. Their feedback helped us refine the lecture slides, target apps and task instructions.

III. Preliminary Results and Discussion

A. Scripted GUI Testing Frameworks

RQ1*: Adoption in open-source projects:* We collected 502 apps with UI tests from [8] to examine the adoption rates of Espresso and UI Automator. We used the *Selenium* web crawler (https://selenium.dev) to identify repositories that imported the libraries from the frameworks and called the APIs in the test code under the *androidTest* folders in the projects. Table I summarises the results. We filtered out 27 projects which have been taken down after the construction of the dataset. 277 out of 475 apps use Espresso, UI Automator, or both[2]. Espresso is clearly more prominent than UI Automator, with 274 apps using it (vs 21). UI Automator is rarely used exclusively (only in three projects). Out of 18 apps which use both frameworks, 15 of them include calls to both frameworks' APIs in the same test (actual combination). Of these, (i) 9 apps have UI Automator exclusive system-level interactions (e.g., pressing the home button, recent app button, opening the notification bar, etc), (ii) 4 apps have `wait` actions, (iii) 3 apps encompass both system-level interactions and `wait` actions.

TABLE I: Adoption in open-source Android apps

Group	Number of Apps
Total apps with UI tests	502
Active apps	475
Apps including Espresso APIs	274
Apps including UIAutomator APIs	21
Apps including both	18
Apps with actual combination	15

RQ1: 58% of projects with UI tests use Espresso APIs while only 4% use UI Automator. Among the apps using UI Automator, 71% combine it with Espresso.

B. User Study

RQ2*: Effectiveness at GUI testing task:* Six participants took part in the study. Three of them attempted all three test cases, achieving 73% of the test steps provided as requirements on average. At this early stage, this confirms the viability of our methodology, but attempting to analyse quantitative effectiveness metrics would be futile. After conducting further replications to collect more data, we plan to evaluate code coverage, maintainability and fault detection capability.

RQ3*: Perspectives towards Espresso and UI Automator:* Overall, participants preferred to use Espresso to implement the required test cases. The activity logs suggest they all followed the pattern *Understanding Tasks* $\rightarrow$ *Searching Solutions* $\rightarrow$ *Applying/Refining Solutions*. In the category of *Applying/Refining Solutions*, all participants spent more time using Espresso than using UI Automator.

In response to the survey, four participants, including the three participants who attempted all three test cases, said they used both Espresso and UI Automator for the task. The main challenge they identified was searching for Android/resource ID and locating views/UI elements. Three participants had trouble locating views without IDs (e.g., a searchable view) using Espresso without knowing it can be handled by UI Automator. Answers to *"What actions do you want to perform but the testing framework(s) you used can not?"* confirm participants' confusion, as they mentioned actions which can indeed be performed by both frameworks, e.g., *"Selecting specific items in a list using item number"*. These observations are understandable, given that the participants in our initial study are novice developers, but may also be symptomatic of a lack of comprehensive documentation online for the frameworks, in particular for UI Automator, as participants were encouraged to use online resources during the study yet still struggled to locate helpful material.

All participants perceived Espresso as easier to use, with statements such as *"It was more intuitive for me"* and *"Syntax*

[1]https://bit.ly/android-testing-study

[2]Investigating the adoption of other Android GUI testing frameworks, e.g., Appium (https://appium.io), is left for future work.

was easier" (survey question #16). They also acknowledged that UI automator could be helpful to perform actions that Espresso cannot: *"It can perform some actions that cannot be performed by Espresso"* and *"I only used UI Automator for the last test case, when I had to interact with the system UI, rather than just the application, since this is fundamentally not supported by Espresso"* (survey question #17).

RQ3: Participants prefer using Espresso due to its simple APIs but acknowledge that combining it with UI Automator could mitigate Espresso's limitations.

***RQ4**: Challenges and Opportunities:* While this is still work in progress, the low adoption of scripted GUI testing frameworks seems evident. We conjecture the contributing factors include insufficient mobile testing education and a lack of comprehensive documentation with usage examples and best practices. Our work needs maturing in order to answer this RQ more assertively, but enhanced tool support and automated generation of scripted tests seem like open challenges.

IV. Threats to Validity[3]

External validity: The dataset used in this work is obtained from previous work [8], and we consider only tests located in the *androidTest* folder as UI tests. It is technically possible, though not very likely as it goes against Android app development guidelines, to place UI tests in other directories. Missing these tests could impact the accuracy of the results of the adoption rates of the frameworks. Furthermore, our initial study only included novice developers as participants, therefore answers to RQ2-4 may change after conducting further replications with more experienced participants.

Internal validity: The data from the activity log is produced by human developers (participants) and is therefore subject to cognitive biases. We mitigated this threat by requesting participants to associate a confidence level with their activities and reassuring them that they were in an observational study and were not assessed on their performance.

V. Conclusions and Future Work

This work-in-progress paper presents the methodology and preliminary results of our study on the use of scripted GUI testing for Android apps. With the analysis of 475 open-source apps with UI tests and a user study with 6 computer science students, we examined the adoption of Espresso and UI Automator in open-source Android applications and the perspectives of developers with no prior experience in Android GUI testing. We found that (i) UI Automator has a lower adoption rate than Espresso in open-source Android apps yet most projects featuring UI Automator combine it with Espresso and (ii) novice developers tend to prefer Espresso because of its simple APIs while recognising that combining it with UI Automator can address its limitations.

This paper reports on an initial study with six novice developers. We aim to conduct further replications of our study with more experienced developers as participants. We also aim to employ other methods, such as interviews or think-aloud observations, to elicit further insights from them. Further replications will also enable us to perform more quantitative evaluations of participants' performance. We believe the insights from this study and future replications could be beneficial for practitioners to improve tool support and to inform further research in automated UI test generation [5].

[3]Due to space constraints, we leave a more comprehensive account of threats to validity for an extended version of this work.

References

[1] J. Gao, L. Li, T. F. Bissyandé, and J. Klein, "On the evolution of mobile app complexity," in *Intl. Conf. on Eng. of Complex Computer Systems*. IEEE, 2019.

[2] M. Linares-Vásquez, C. Bernal-Cardenas, K. Moran, and D. Poshyvanyk, "How do developers test android applications?" in *Intl. Conf. on Software Maintenance and Evolution (ICSME)*. IEEE, 2017.

[3] P. S. Kochhar, F. Thung, N. Nagappan, T. Zimmermann, and D. Lo, "Understanding the test automation culture of app developers," in *Intl. Conf. on Software Testing, Verification and Validation (ICST)*. IEEE, 2015.

[4] S. Negara, N. Esfahani, and R. Buse, "Practical android test recording with espresso test recorder," in *Intl. Conf. on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019.

[5] I. Arcuschin, C. Ciccaroni, J. P. Galeotti, and J. M. Rojas, "On the feasibility and challenges of synthesizing executable Espresso tests," in *Intl. Conf. on Automation of Software Test (AST)*. ACM, 2022.

[6] S. R. Choudhary, A. Gorla, and A. Orso, "Automated test input generation for android: Are we there yet?" in *Intl. Conf. on Automated Software Engineering (ASE)*, 2015.

[7] R. Coppola and M. Torchiano, "Scripted gui testing of android apps: A study on diffusion, evolution and fragility," in *Intl. Conf. on Predictive Models and Data Analytics in Software Engineering*. ACM, 2017.

[8] J.-W. Lin, N. Salehnamadi, and S. Malek, "Test automation in open-source android apps: A large-scale empirical study," in *Intl. Conf. on Automated Software Engineering (ASE)*. IEEE, 2020.

[9] D. Zelenchuk, *Android Espresso Revealed: Writing Automated UI Tests*. Apress Berkeley, CA, 2019.

[10] W. Mason and D. J. Watts, "Financial incentives and the "performance of crowds"," in *SIGKDD Workshop on Human Computation (HCOMP)*. ACM, 2009.

[11] L. Ardito, R. Coppola, M. Morisio, and M. Torchiano, "Espresso vs. EyeAutomate: An experiment for the comparison of two generations of android gui testing," in *Eval. and Assmt. on Soft. Eng. (EASE)*. ACM, 2019.

[12] L. Deng, J. Offutt, P. Ammann, and N. Mirzaei, "Mutation operators for testing android apps," *Inf. Softw. Technol.*, vol. 81, no. C, 2017.

[13] T. Roehm and W. Maalej, "Automatically detecting developer activities and problems in software development work," in *Intl. Conf. on Software Engineering (ICSE)*. IEEE, 2012.