Source-Codeless Testing for Android Apps

Camilo Escobar-Velásquez
Advised by: Mario Linares-Vásquez
Universidad de los Andes, Bogotá, Colombia
ca.escobar2434@uniandes.edu.co

***Abstract*—Most of the automated approaches to support software engineering tasks for mobile applications, use source code as entry, which due to privacy factors imposes hard constraints on the implementation of those approaches by third-party services. However, the continuous pressure from market is leading practitioners to crowdsource/outsource software engineering tasks to third-parties that provide on-the-cloud infrastructures. Therefore, approaches that work at closed-source level (i.e., do not require source code) are desirable to enable automated outsourced software engineering tasks by third-party services. Specifically, in this paper we focus on testing since is a task that has broadly accepted approaches at source-code level, but new efforts are required to enable source-codeless approaches. We analyzed the current challenges and shortfalls of the currently proposed approaches that works at source code and APK level, and present our research plan oriented to reduce the existing gap for enabling testing techniques at APK level.**

***Index Terms*—Android, APK, Source-codeless, Testing**

I. Problem and Motivation

The power and usefulness of a large number of state-of-the-art approaches for automated software engineering of Android apps rely on the existence of source code for extracting intermediate representations or models that drive the analysis execution or the artifacts generation. However, practitioners often outsource software engineering tasks without releasing the source code, *i.e.,* the third parties work directly on Android Application Packages (APKs), due to different legal and organizational reasons (*e.g.,* source code contains a company's exclusive implementation of an algorithm, source code contains keys/secrets for services, etc.). Furthermore, software engineering approaches for mobile apps that first rely on source code modifications/instrumentation, and then require to build/compile the app, are time consuming in particular because building executable files is time consuming; one representative example is mutation testing, which requires building an executable file for each mutation applied to the original code [1]–[4]. Additionally, there are several programming languages and frameworks that can be used to develop an Android app (*e.g.,* Java, Kotlin, Dart), therefore, testing approaches that rely on source-code are dependent of the language used for creating the app. This increases the research effort because specific source code-based approaches are required for each language.

Previous works have been done in order to identify the current efforts and shortfalls for different testing techniques for Android apps. For instance, Linares-Vásquez *et al.* [5], Kong *et al.* [6], and Li *et al.* [7] present a general view of already proposed testing tools and approaches for both source-code and APK level. Table I summarizes the testing techniques and objectives used/addressed previously, based on [5], [6], and [7]. Note that there are several techniques and objectives that lack a comprehensive approach for either source code or APK, for example, previous works for mutant generation [2], [8]–[10] do not provide a comprehensive set of mutation operators or do not execute the mutants. Additionally, for cross-device testing the only known approach proposed, called Octopus [11], is not publicly available and is tailored for a specific app requirements.

Table I: Enabled testing techniques and objectives based on input type. A ✓ means the technique/objective has been addressed, while an ✗ means it has not.

Testing Technique	**SC**	**APK**	**Testing Objective**	**SC**	**APK**
Random Testing	✓	✓	Bug/Detection	✓	✓
Record and Replay	✓*	✓$^+$	Accesibility	✗	✗
Model-based Testing	✓	✓	Energy	✓	✓
Systematic Exploration	N/A	✓	Performance	✓	✓
Cross-device Testing	✗	✗	Security	✓	✓
Mutation testing	✗	✗	Internationalization	✗	✗

***: requires APK +: enhanced with Source-code**

To overcome the aforementioned issues (in particular for Android apps), our main goal is to enhance source-codeless automated testing (*i.e.,* automated testing techniques that uses APKs as input), to avoid the overload imposed by compiling APK files from source code and avoid dependencies on specific languages used to implement Android apps.

II. Research Program

The objective of this paper is to present a research program to address the missing techniques/objectives previously presented. In particular, a detailed plan is presented involving three main phases: (i) Tools and Infrastructure (1st year), (ii) Testing Approaches (2nd and 3rd year), and (iii) Comprehensive validation with industry (4th year). For each phase, we present a summary of the expected results:

Tools and Infrastructure: The main objective for the first phase is to understand the current state of the art an implement an initial set of tools in order to enable further research around techniques and objectives from Table I. It is worth to notice that part of the expected results are merely infrastructure results (*i.e.,* do not extend the state of the art from a research point of view) and some would generate and contribute to improving the state-of-the-art. The main expected tools are: First, a tool for mutant generation/testing that works at APK level; while there is previous work on mutant generation at APK level [2], [12], there is no comprehensive tool for

Acknowledgment. This work is founded by a Google Latin America Research Award 2018-2020.

mutation testing of Android apps. Second, a tool for improving systematic exploration, since current approaches [13]–[15] present issues around false/positives state identification, do not work for hybrid applications and do not provide a "replay" feature to enable regression testing. Third, a cross-device testing tool, that does not rely in human-interaction, is publicly available and enables interaction-based execution.

Testing Approaches: For this second phase, our plan is to address some testing techniques and objectives that to the time of the writing of this paper have not been studied yet. Firstly, mutant selection techniques applied at APK level; it is important to note that this have been addressed previously at source code level, however, the current approaches as generating mutants only over the last modified/added code or prioritized code is not suitable at APK since a more complex and error prone translation process must be done in order to get the SMALI representation of the changes or selected code. A second objective is to enable internationalization testing for android apps at APK level, by extending previous works on web apps [16]–[18]. We also expect to create an automated approach for accessibility testing; Vendome *et al.* [19], show that no current approach has been proposed, and elucidate its need for smartphone users. Finally, our last goal is to improve model-based testing by relying on the usage of machine learning techniques to extract event sequences from videos along with contextual information in order to reproduce bug reports.

Validation with industry: The third and last phase, consists of a further validation of our approaches with industry practitioners, in order to obtain enhanced feedback and to evaluate whether the proposed approaches can help to improve real testing processes. It is worth notice that the approaches created in the first two phases will be validated with empirical studies with students, and preliminary studies with practitioners. However, this last phase will include larger longitudinal studies with different software companies.

III. Work in progress

Phase 1 - Tools and Infraestructure: Our current efforts have been dedicated to the design and implementation of two of the proposed tools: (i) *MutAPK* a publicly available command-line tool for mutant generation at APK level, based on a set of 38 mutation operators [20] inherited from MDroid+ [2], [8]. Specifically, we used the SMALI intermediate representation as the base to define the mutation rules, since studies were showing the benefits of SMALI representation when compared to other representations closer to the APK and the source code [21], [22]. It is important to note that because MutAPK works with APKs, it is straightforward to generate mutated APKs without compiling source code. Therefore, all the APKs generated by MutAPK can be directly installed to a device/emulator to be used with the test suite. MutAPK implementation and usage details are depicted in a tool demo paper [20] presented at ASE'19. Additionally, a comprehensive study has been done comparing different mutant generation tools at both source code and APK level. The current results have shown that mutant generation at APK level is faster than source code approaches, while generates more non-functional mutant when compared to source code approaches. Also, from the aforementioned comparison, we identify shortfalls in mutant generation at APK level as mutation understandability due to its closeness to machine code. This comparison was presented in a paper that is currently under review st the TSE journal. Future work for mutation will be oriented to implement mutant selection techniques and to improve the quality of generated mutants. Finally, due to the extensibility of MutAPK, a mutant execution mode should be added, to completely enable mutation testing at APK level.

(ii) *Kraken-Mobile* was implemented to enable testing of apps whose expected usage requires the interaction between several devices (*e.g.,* uber rider and uber driver apps). Because of this, Kraken was conceived as a cross-device interaction-based testing tool that allows practitioners to define test suites using scenarios written with the Gherkin [23] syntax. This allows Kraken's users to use natural language to define tests and then generate the corresponding matching with step definition by using regular expressions. Kraken provides instructions that enable the waiting and sending of a signal to orchestrate the execution of the test cases on different devices. Kraken is publicly available [24] and it was presented in a tool demo paper [25] at ICSME'19. From the aforementioned article's results, we identified some restrictions for the easiness of test suite definition. Since, due to selected syntax, signals flow might be confusing and a new test definition format could be proposed to resemble a sequence diagram, either after processing the file or as the default test definition process. Moreover, Kraken could be extended to enable other interaction scenarios that include web apps.

Phase 2 - Testing Approaches: Our current progress in the second phase is on *Internationalization testing*. We have implemented the first version of a tool to recognize and locate internationalization changes/bugs on android apps. Our approach automatically generate different internationalized versions of an original APK and detect presentation issues in the internationalized versions, by relying on automated techniques such as static analysis, automated APK manipulation, systematic exploration and replay. As of today, our approach identifies layouts changes/bugs; future work will be dedicated to (i) recognize the changes that break the app's layout, and (ii) automatically repair the issues.

IV. Expected contributions

Our research plan is expected to close the current gap for source-codeless testing. Specifically, we expect to provide practitioners with approaches and tools that enhance its testing experience. The envisioned approaches will help practitioners to enable automated testing at the level of techniques that are not widely used (*e.g.,* mutation testing) and quality attributes that currently do not have support (*e.g.,* internationalization and accessibility). In addition, we expect that having approaches and tools that do not rely on source code will impact positively the times involved in testing tasks.

References

[1] D. Appelt, C. D. Nguyen, L. C. Briand, and N. Alshahwan, "Automated testing for sql injection vulnerabilities: an input mutation approach," in *ISSTA-14*, 2014, pp. 259–269.

[2] M. Linares-Vásquez, G. Bavota, M. Tufano, K. Moran, M. Di Penta, C. Vendome, C. Bernal-Cárdenas, and D. Poshyvanyk, "Enabling mutation testing for android apps," in *ESEC/FSE'17*, 2017, pp. 233–244.

[3] U. Praphamontripong, J. Offutt, L. Deng, and J. Gu, "An experimental evaluation of web mutation operators," in *ICSTW´16*, 2016, pp. 102–111.

[4] D. Rodríguez-Baquero and M. Linares-Vásquez, "Mutode: generic javascript and node. js mutation testing tool," in *ISSTA'18*, 2018.

[5] M. Linares-Vásquez, K. Moran, and D. Poshyvanyk, "Continuous, evolutionary and large-scale: A new perspective for automated mobile app testing," in *ICSME'17*, Sep. 2017, pp. 399–410.

[6] P. Kong, L. Li, J. Gao, K. Liu, T. F. Bissyandé, and J. Klein, "Automated testing of android apps: A systematic literature review," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 45–66, 2018.

[7] L. Li, T. F. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, and L. Traon, "Static analysis of android apps: A systematic literature review," *Information and Software Technology*, 2017.

[8] K. Moran, M. Tufano, C. Bernal-Cárdenas, M. Linares-Vásquez, G. Bavota, C. Vendome, M. Di Penta, and D. Poshyvanyk, "Mdroid+: A mutation testing framework for android," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ACM, 2018, pp. 33–36.

[9] L. Deng, N. Mirzaei, P. Ammann, and J. Offutt, "Towards mutation analysis of android apps," in *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*, April 2015, pp. 1–10.

[10] R. Jabbarvand and S. Malek, "mudroid: An energy-aware mutation testing framework for android," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: ACM, 2017, pp. 208–219. [Online]. Available: http://doi.acm.org/10.1145/3106237.3106244

[11] B. J. A. Chow. (2015) Octopus to the rescue: The fascinating world of inter-app communications at uber engineering. [Online]. Available: https://eng.uber.com/rescued-by-octopus/

[12] Yuan-W, "mudroid project at github," 2017, https://goo.gl/sQo6EL.

[13] Y. Li, Z. Yang, Y. Guo, and X. Chen, "Droidbot: A lightweight ui-guided test input generator for android," in *Proceedings of the 39th International Conference on Software Engineering Companion*, ser. ICSE-C '17. IEEE Press, 2017, p. 23–26. [Online]. Available: https://doi.org/10.1109/ICSE-C.2017.8

[14] Google, "Firebase test lab. https://firebase.google.com/docs/test-lab."

[15] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, and D. Poshyvanyk, "Automatically discovering, reporting and reproducing android application crashes," in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, April.

[16] A. Alameer, S. Mahajan, and W. G. Halfond, "Detecting and localizing internationalization presentation failures in web applications," in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2016, pp. 202–212.

[17] A. Alameer and W. G. Halfond, "An empirical study of internationalization failures in the web," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2016, pp. 88–98.

[18] S. Mahajan and W. G. J. Halfond, "Detection and localization of html presentation failures using computer vision-based techniques," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, April 2015, pp. 1–10.

[19] C. Vendome, D. Solano, S. Liñán, and M. Linares-Vásquez, "Can everyone use my app? an empirical study on accessibility in android apps," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019.

[20] O.-R. M. Escobar-Velásquez, Camilo and M. Linares-Vásquez, "Mutapk: Source-codeless mutant generation for android apps," in *2019 IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019.

[21] Y. Arnatovich, H. B. K. Tan, S. Ding, K. Liu, and L. K. Shar, "Empirical Comparison of Intermediate Representations for Android Applications," in *Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering*. Knowledge Systems Institute Graduate School, 2014, pp. 205–210.

[22] Y. L. Arnatovich, L. Wang, N. M. Ngo, and C. Soh, "A comparison of android reverse engineering tools via program behaviors validation based on intermediate languages transformation," *IEEE Access*, vol. 6, pp. 12 382–12 394, 2018.

[23] Cucumber. (2012) Gherkin language. [Online]. Available: https://cucumber.io/docs/gherkin/

[24] "Rip. https://thesoftwaredesignlab.github.io/KrakenMobile/."

[25] E.-V. C. Ravelo-Méndez, William and M. Linares-Vásquez, "Kraken-mobile: Cross-device interaction-based testing of android apps," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019.