Enabling Automated Software Engineering Tasks for Closed-Source Mobile Apps

Mario Linares-Vásquez, Ph.D.
Assistant Professor
Systems and Computing Engineering Department, Universidad de los Andes
Mario Laserna Hall, Bogotá, Colombia
Phone: (57) 320 292 58 66; m.linaresv@uniandes.edu.co

1. Problem statement

Mobile markets have pushed and promoted the raising of an interesting phenomenon that has permeated not only developers culture, but also human beings' daily life activities. Mobile devices, apps, and services are helping companies and organizations to make "digital transformation" possible through services and capabilities that are offered ubiquitously and closer to the users. Nowadays, mobile apps and devices are the most common way for accessing those services and capabilities; in addition, apps and devices are indispensable tools for allowing humans to have in their phones, computational capabilities that make life better and easier. The mobile apps phenomenon has also changed drastically the way how practitioners design, code, and test apps. Mobile developers and testers face critical challenges on its daily life activities such as (i) continuous pressure from the market for frequent releases of high quality apps, (ii) platform fragmentation at device and OS levels, (iii) rapid platform/library evolution and API instability, and (iv) an evolving market with millions of apps available for being downloaded by ends users. Tight release schedules, limited developer and hardware resources, and cross-platform delivery of apps, are common scenarios when developing mobile apps [1]. Therefore, reducing the time and effort devoted to software engineering tasks while producing high quality mobile software is a ''precious'' goal.

Both practitioners and researchers, have contributed to achieve that goal, by proposing approaches, mechanisms, best practices, and tools that make the development process more agile. Cross-platform languages and frameworks (e.g., Flutter, Ionic, Xamarin, React Native) contribute to reducing the development time by providing developers with a mechanism for building Android and iOS versions of apps in a write-one-run-anywhere way [1,2]. Automated testing approaches help testers to increase the apps quality and reduce the detection/reporting time [3,4,5]. Automated categorization of reviews also helps developers to select relevant information, issues, features and sentiments, from large volume of review that are posted by users [6,7,8]. Moreover, approaches for static analysis, are helping developers to early detect different types of bugs and issues that without the automated support could be time consuming for developers --- when doing the analysis manually [9]. Finally, on demand documentation generation is a very active field motivated by the need of having documentation that support the information need of different stakeholder [10].

The aforementioned approaches are representative of the state-of-the-art for supporting mobile software engineering tasks. However, the developer's community is quickly moving towards using cloud-services and crowd-sourced services for software engineering tasks; using those services is becoming a common practice of mobile developers who want to reduce costs and the time devoted for an activity. For example, the Firebase Test Lab platform [11] provides automated testing services, in particular, it automatically executes/explores a given app (provided by the developer as an Android APK file), and reports crashes found on a devices matrix that is selected by the user. However, the power and usefulness of the state-of-the-art approaches rely on the existence of source code for extracting intermediate representations or models that drive the analysis execution or the artifacts generation

Existing approaches that rely on source code for supporting automated software engineering tasks are untenable in a commercial environment where third-party services are used to outsource software engineering, but without releasing the source code (i.e., the services work directly on executable files). Any type of analysis that relies only on executable files (i.e., dynamic analysis) is known to be limited when compared to static analysis that can be done directly on the code. But, for different legal and organizational reasons, the app' source code is often not available, because developers do not allow third-parties to have access to the apps source code, thus, making it difficult to enable cloud/crowd-source services that could offer services that use state-of-the-art approaches.

In general, state-of-the-art approaches can be enabled in local environments where the source code is available and can be manipulated by the owners. Conversely, state-of-the-practice approaches offered by third-parties (e.g., testing) rely on manual analysis of the apps, or on automated dynamic analysis that operates at the APK level.

2. Challenges and research goals of the proposed research

The main research goal is to provide developers with approaches and tools for supporting different automated software engineering tasks, that work without having the source code. In particular, our goal is to design and implement approaches for automatically extracting models from Android executable files (i.e., APK files), and use those models to support three specific tasks: (i) on demand documentation generation, (ii) test cases automated generation/execution, and (iii) mutation testing. Our preference for these three specific tasks is because automated extraction of models from APKs with testing and documentation purposes is still an open area (few works have been devoted to testing and no work targets documentation), and mutation testing at APK level is a fundamental task that could help to (i) drive test cases generation, or (ii) measure the quality of existing test suites. Security-related tasks such as malware/vulnerabilities detection, data leaks detection, and performance analysis have been widely explored at APK level, and different approaches have been already proposed [9,12], therefore, we will not focus on those.

We now outline the challenges and specific goals for our proposed research work:

- **Analysis of Android executable files.** While decompiling APKs is an option for accessing to the source code and then doing the analyses on it, there are three issues that motivate us to avoid this. First, going from an APK to source is a time-consuming task, and requires going back to the APK in the case where the code needs to be modified (as in the case of mutation testing). Second, decompiling APKs could loss information because there is no guarantee that 100% of the code can be always decompiled. Third, decompiling code has licensing and copyright issues. Thus, we need to build an approach that is able, to extract models from APKs. Existing tools like SOOT [21] operates on Android bytecode to perform points-to, inter-procedural data-flow, and taint analyses, and call graphs extractions. SOOT uses an intermediate representation called Jimple for the analysis, however, previous studies have shown that Jimple preserves 73% - 85% of the information in APKs [13,14]. Therefore, our analysis should rely on a more accurate intermediate representation extracted from APKs, and without incurring on the overhead produced when decompiling from APK to source code.

- **Android apps model extraction.** Based on an intermediate representation extracted from the APKs, we will build different models that allow us to generate documentation, test cases, and mutants (for mutation testing). In addition to traditional models like call graphs, we need to generate models that are tailored to Android apps, i.e., models that consider the Android programming model and its components. The models should be expressive enough for supporting a broad range of software engineering tasks. For example, efficiently testing Android apps with the purpose of detecting faults requires focusing the testing process on scenarios that have a high probability of exercising risky components and statements in the app under testing (AUT). We consider as risky components and statements those code units and statements that are related to API elements (from the official API and third-party libraries) that are fault-prone because of (i) API misuses, (ii) unexpected arguments, (ii) unexpected returns (e.g., a back-end service), (iii) unexpected runtime exceptions, (iv) bugs in the API implementation, and (v) breaking changes in a new release of the API. Therefore, locating those risky components/statements in the AUT and how they are related to the AUT behavior can provide practitioners and automated approaches with a model for designing test cases. Achieving this goal will require to generate a fault-model, like the one used in [15].

- **Generation of actionable test cases, documentation, and mutants.** Based on the models extracted from the APKs, our third goal is to automatically generate test cases, documentation, and mutants. Concerning the test cases, a robust automatic testing framework should generate test cases as streams of events that can be reproduced automatically on a device running the AUT. These test cases should not be low-level event streams, but rather, human readable streams that developers can easily comprehend and execute automatically (i.e., actionable streams). In the case of documentation, it should follow some of the envisioned principles described by Robillard et al. [10], in particular that the documentation could be generated on demand by the developer and support different information needs that are described as queries. Finally, mutants should be generated in a similar way as in MDroid+ [15,16], but without the need of source code.

In the next section, we present our proposal for *Enabling Automated Software Engineering for Closed-Source Mobile Apps*. We anticipate that the research will provide not only practical solutions, but also a comprehensive analysis of the available approaches for automated mobile software engineering.

3. Proposed Research

We propose a novel framework aimed at enabling three automated software engineering tasks for Android apps at APK level, i.e., without the need of source code: (i) on-demand developer documentation, (ii) actionable test cases derivation, and (iii) mutants creation for mutation testing. We believe that working with APK files (i) reduces the need of having source code for implementing static analysis approaches that have been proved to be effective, (ii) enables the execution of automated software tasks by third parties that do not access to the source code, and (iii) avoids the overhead introduced by de-compilation process that extracts "original" source code from APKs. The research program is structured in the following four phases.

Phase 1: Exploring intermediate representations. When going from Java/Kotlin/Flutter source code to APKs (and **vice versa**), different intermediate representations (e.g., DEX code, Smali, Jimple, Jasmin) can be used as reported in [13,14]. Therefore, the first phase of the research program will be devoted to understanding state-of-the-art intermediate representations. As part of this phase we have made some initial progress, in particular for exploring the Smali representation, which is reported as the top one representation that preserves the information from APKs (~97%) and the top-one used in previous work [13,14]. To this, we have started the development of a tool for mutation analysis of APK files (MutAPK) that implements the same mutation operators implemented by MDroid+ [15,16] but modifying directly APK files and working with the Smali representation. MutAPK is **under development** and it is publicly available at https://github.com/TheSoftwareDesignLab/MutAPK**.** Note that despite having an initial preference for Smali, we might end selecting a different intermediate representation. The expected results of this phase is the analysis of intermediate representation, and the selection of the one most suitable for this research program purposes.

Phase 2: Extracting models from APKs. The expected result of this phase is an approach (and corresponding tools) for extracting models statically from APK files, based on the intermediate representations analyzed in **Phase 1.** The models should be expressive enough for enabling different software engineering tasks, such as call graphs, control flow graphs, GUI hierarchies, context models, multi-models for testing [5], and fault-profiles [15]. The time for extracting the models should be lower than extracting the models from source (assuming a de-compilation pipe). The approaches for extracting the models will be presented to the community via journal and conference papers, and we will release the tools as open source prototypes that can be extended by researchers/practitioners to implement their own analyses.

Phase 3: Generating actionable test cases and mutants. The expected result of this phase is two-fold: (i) an approach and prototype that generates actionable test cases from the models extracted automatically from the APKs, and (ii) a robust tool for mutation testing of APK files. The generated test cases should be expressive to be executed manually but also should be generated in formats that can be executed automatically (e.g., Espresso syntax). The research prototypes from **Phase 3** might be used by testing teams, and researchers interested in comparing their own approaches to our risk-driven test cases generator for Android apps; in addition, researchers might build upon the open source prototype. As part of the validation of the final prototype we will compare our tool to the Androtest suite [21], our previous approaches [15,16,17,18], and other more recent approaches [19,20]. In addition, the output of the research prototype generated with **Phase 3** might be used by testing teams to support automated testing, and by researchers interested in automated testing of mobile apps; researchers might also extend the open source prototype to implement their own tools. Our approach and tools could be used by tools like Firebase Test Lab that automatically test APK files; current capabilities of tools/services like Firebase Test Lab are very limited because it relies only on systematic exploring, i.e., there is no automation of test cases because no models are extracted from the APKs code.

Phase 4: Generating on demand developer documentation. Finally, we expect to propose an approach for automated extraction (from APKs) of developer documentation. We aim at contributing to the on-demand developer documentation vision [10], by allowing developers and practitioners to query and generate content based on the models extracted from the APKs. We envision different usage scenarios here such as re-documentation for manual testing, architecture recovery, features documentation, among others. We will release the tools as open source prototypes that can be extended by researchers/practitioners to implement their own analyses.

4. References

1. M.E. Joorabchi, A. Mesbah, and P. Kruchten. *Real challenges in mobile apps*, in *ESEM'13*. 2013.
2. M. Fazzini and A. Orso. 2017. *Automated cross-platform inconsistency detection for mobile apps*. In ASE'17. 2017.
3. S.R.Choudhary,A.Gorla,andA.Orso.*Automated test input generation for android: Are we there yet?*, in ASE'15, 2015
4. P.S. Kochhar, F. Thung, N. Nagappan, T. Zimmerman, and D. Lo. *Understanding the Test Automation Culture of App Developer*s, in ICST'15, 2015
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. 2017
6. F Palomba, M. Linares-Vásquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, A. De Lucia. *Crowdsourcing user reviews to support the evolution of mobile apps*, Journal of Systems and Software, Volume 137, 2018.
7. L. Villarroel, G. Bavota, B. Russo, R. Oliveto and M. Di Penta. *Release Planning of Mobile Apps Based on User Reviews*, in ICSE'16. 2016.
8. A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall. 2016. *What would users change in my app? summarizing app reviews for recommending software changes*, in FSE'16. 2016.
9. L. Li, T. F. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, L. Traon. *Static analysis of android apps: A systematic literature review*, Information and Software Technology, Volume 88, 2017.
10. M. P. Robillard et al. *On-demand Developer Documentation*, in ICSME'17. 2017.
11. Google. Firebase Test Lab for Android. https://firebase.google.com/docs/test-lab/
12. A. Sadeghi, H. Bagheri, J. Garcia and S. Malek. *A Taxonomy and Qualitative Comparison of Program Analysis Techniques for Security Assessment of Android Software*, Transactions on Software Engineering, vol. 43, no. 6, pp. 492-530, June 1 2017.
13. 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*," in IEEE Access, vol. 6, pp. 12382-12394, 2018.
14. Y. L. Arnatovich, H. B. Kuan Tan, S. Ding, K. Liu, L. K. Shar. *Empirical Comparison of Intermediate Representations for Android Applications*, in SEKE'14. 2014
15. M. Linares-Vásquez, G. Bavota, M. Tufano, K. Moran, M. Di Penta, Christopher Vendome, C. Bernal-Cárdenas, and D. Poshyvanyk. *Enabling mutation testing for Android apps*, in FSE'17.2017.
16. K. Moran, M. Tufano, C. Bernal-Cárdenas, M. Linares-Vásquez, G. Bavota, Christopher Vendome, M. Di Penta, and D. Poshyvanyk. *MDroid+: A Mutation Testing Framework for Android*, in ICSE'18. 2018.
17. M. Linares-Vásquez, M. White, C. Bernal-Cárdenas, K. Moran and D. Poshyvanyk. *Mining Android App Usages for Generating Actionable GUI-based Execution Scenarios*, in *MSR'15*, 2015
18. K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, and D. Poshyvanyk. *Automatically Discovering, Reporting and Reproducing Android Application Crashes*, in ICST'16. 2016
19. Ke Mao, Mark Harman, and Yue Jia. 2016. *Sapienz: multi-objective automated testing for Android applications*, in ISSTA'16. 2016
20. T. Gu *et al*., *AimDroid: Activity-Insulated Multi-level Automated Testing for Android Applications*, in ICSME'17. 2017
21. Sable. Soot – A Java optimization framework. https://github.com/Sable/soot

5. Data Policy

The research prototypes will be made open-source and the source code will be hosted at GitHub. The datasets and results from the experiments conducted with apps, will be also freely available at a website.

Mario Linares-Vásquez
Curriculum Vitae

Universidad de los Andes
Systems and Computing Engineering Department
Cra 1 Este No 19A 40 ML 652
Bogotá, Colombia

Tel: (57 1) 339 49 49
Fax: (57 1) 332 43 25
E mail: m.linaresv@uniandes.edu.co
https://sistemas.uniandes.edu.co/~mlinaresv

Professional Preparation

The College of William and Mary (US)	Computer Science	Ph.D	2016
Universidad Nacional de Colombia (CO)	Systems and Computing Engineering	M.Sc.	2009
Universidad Nacional de Colombia (CO)	Systems and Computing Engineering	B.E.	2005

Appointments

Universidad de los Andes (CO)	Assistant Professor	from June 2016
College of William and Mary (US)	Research Assistant	August 2012 – May 2016
College of William and Mary (US)	Visiting Scholar	August 2011 – January 2011

Honors and Awards

- **ACM SIGSOFT Distinguished Paper Award**, at 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), 2017
- **ACM SIGSOFT Distinguished Paper Award**, at the 24th IEEE International Conference on Program Comprehension (ICPC), 2016
- **ACM SIGSOFT Distinguished Paper Award**, at the 23rd ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), 2015
- **The College of William and Mary CS Department Stephen K. Park Graduate Research Award,** 2015
- **International Student Achievement Award 2015 – Jack Wolf Scholarship**, Reves Center of International Studies at The College of William Mary, 2015
- **ACM Student Research Competition (First Place),** at the 37th IEEE/ACM International Conference on Software Engineering (ICSE), 2015
- **Best Paper Award,** at the 29th IEEE International Conf. on Software Maintenance (ICSM), 2013

Five Products Most Closely Related to the Proposal

1. **Linares-Vásquez, M.,** Bavota, G., Tufano, M., Moran, K., Di Penta, M., Vendome, C., Bernal-Cárdenas, C., and Poshyvanyk, D., "Enabling Mutation Testing for Android Apps", *in Proceedings of 11th Joint Meeting of the European Software Engineering Conference and the 25th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (ESEC/FSE'17)*, Paderborn, Germany, September 4-8, 2017, pp. 233-244
2. Moran, K., **Linares-Vásquez, M.,** Bernal Cárdenas, C., Vendome, C., and Poshyvanyk, D., "*Automatically Discovering, Reporting and Reproducing Android Application Crashes*", in Proceedings of 9th IEEE International Conference on Software Testing, Verification and Validation (ICST'16), Chicago, IL, April 10 15, 2016, pp. 33-44
3. Moran, K., **Linares-Vásquez, M.,** Bernal Cárdenas, C., and Poshyvanyk, D., *"Auto Completing Bug Reports for Android Applications"*, in Proceedings of 10th Joint Meeting of the European Software Engineering Conference and the 23rd ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'15), Bergamo, Italy, August 31 September 4, 2015, pp. 673 686.
4. **Linares-Vásquez, M.,** Bavota, G., Bernal Cárdenas, C., Oliveto, R., Di Penta, M., and Poshyvanyk, D., *"Optimizing Energy Consumption of GUIs in Android Apps: A Multi objective Approach"*, in Proceedings of 10th Joint Meeting of the European Software Engineering Conference and the 23rd ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'15), Bergamo, Italy, August 31 September 4, 2015, pp. 143 154.

5. **Linares-Vásquez, M.,** White, M., Bernal Cárdenas, C., Moran, K., and Poshyvanyk, D., *"Mining Android App Usages for Generating Actionable GUI based Execution Scenarios"*, in Proc. of the 12th IEEE Conf. on Mining Software Repositories (MSR 15), Florence, May 16 17, 2015, pp. 111 122

Other Android related Papers:

1. Palomba, F., **Linares-Vásquez, M**., Bavota, G., Oliveto, R., Di Penta, M., Poshyvanyk, D., and De Lucia, A., *"Crowdsourcing User Reviews to Support the Evolution of Mobile Apps*", *Journal of Systems and Software (JSS), accepted*
2. **Linares-Vásquez, M.,** Vendome, C., Tufano, M., and Poshyvanyk, D., *"How Developers Micro Optimize Android Apps"*, Journal of Systems and Software (JSS), vol. 130, August 2017
3. **Linares-Vásquez, M.,** Bernal-Cárdenas, C., Moran, K., Poshyvanyk, D., *"How do Developers Test Android Applications?"*, in Proceedings of 33rd IEEE International Conference on Software Maintenance and Evolution (*ICSME'17*), Industry Track, Shanghai, China, September 20-22, 2017, pp. 613-622
4. **Linares-Vásquez M.,** Bavota G, Escobar C., *"An Empirical Study on Android related Vulnerabilities"*, in Proceedings of 14th International Conference on Mining Software Repositories (MSR 17), 2017
5. **Linares-Vásquez, M.**, Bavota, G., Bernal Cárdenas, C., Oliveto, R., Di Penta, M., and Poshyvanyk, D., *"Mining Energy Greedy API Usage Patterns in Android Apps: an Empirical Study"*, in Proceedings of 11th IEEE Working Conference on Mining Software Repositories (MSR 14), Hyderabad, India, May 31 June 1, 2014, pp. 2-11
6. **Linares-Vásquez, M.**, Holtzhauer, A., Bernal Cárdenas, C., and Poshyvanyk, D., *"Revisiting Android Reuse Studies in the Context of Code Obfuscation and Library Usages"*, in Proc. of 11th IEEE Conf. on Mining Software Repositories (MSR 14), India, May 31 June 1, 2014, pp. 242-251
7. **Linares-Vásquez, M.**, Bavota, G., Bernal Cárdenas, C., Di Penta, M., Oliveto, R., and Poshyvanyk, D., *"API Change and Fault Proneness: A Threat to Success of Android Apps"*, in Proc. of 21st ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 13), Saint Petersburg, Russia, August 18 26, 2013, pp. 477-487
8. Bavota, G., **Linares-Vasquez, M.**, Bernal Cardenas, C., Di Penta, M., Oliveto, R., and Poshyvanyk, D. *"The Impact of API Change and Fault Proneness on the User Ratings of Android Apps"*, IEEE Transactions on Software Engineering (TSE), vol. 41, no. 4, April 2015, pp. 384-407
9. White, M., **Linares-Vásquez, M.**, Johnson, P., Bernal Cárdenas, C., and Poshyvanyk, D., *"Generating Reproducible and Replayable Bug Reports from Android Application Crashes"*, in Proceedings of 23rd IEEE International Conference on Program Comprehension (ICPC 15), Florence, Italy, May 18 19, 2015, pp. 48-59

Synergetic Activities

- **Innovations in software engineering education.** Designed a new software engineering course on automated testing of web and mobile apps for the Master in Software Engineering program at Universidad de los Andes; and, redesigned the mobile apps development undergraduate class at Universidad de los Andes to follow a flipped learning approach and to be taught also in English.

- **Professional service.** Serving on the Editorial Board of Information and Software Technology Journal (IST). Reviewer for IEEE Transactions on Software Engineering, International Journal on Empirical Software Engineering, Journal of Software: Evolution and Process, Journal of Systems and Software, and IEEE Software Magazine. Tool demo co-chair for SANER'18, Student-Volunteer co-chair for ICSME´18, and PC member for GREENS'18, ICPC'18 (ERA track), MobileSoft'18, WAMA ́17, ASE`17 (Research track), ICPC'17 (ERA track), SANER ́17 (Research and Tool Demo track), MSR'16 (Mining Challenge track), WAMA ́16, SANER ́16 (Tool Demo track), ICPC'15 (ERA track).