Poster: Prototyping Functional Android App Features with ProDroid

Donghwi Kim
KAIST
dhkim09@kaist.ac.kr

Soo Young Park
KAIST
sypark0614@kaist.ac.kr

Jihoon Ko
KAIST
jihoonko@kaist.ac.kr

Steven Y. Ko
University at Buffalo
stevko@buffalo.edu

Sung-Ju Lee
KAIST
profsj@kaist.ac.kr

Abstract

We present ProDroid, a framework that provides Android app developers an ability to quickly produce functional prototypes. With ProDroid, developers can create a new app that imports various kinds of functionality provided by other *existing* Android apps. Our evaluation shows that with the help of ProDroid, a developer was able to import a function from an existing Android app into a new prototype with only 55 lines of Java code, while the function itself requires 10,334 lines of Java code to implement.

CCS Concepts

• **Software and its engineering → Reusability**.

Keywords

Android; Functional prototyping; Development frameworks

ACM Reference Format:

Donghwi Kim, Soo Young Park, Jihoon Ko, Steven Y. Ko, and Sung-Ju Lee. 2019. Poster: Prototyping Functional Android App Features with ProDroid. In *The 17th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '19), June 17–21, 2019, Seoul, Republic of Korea.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3307334.3328621

1 Introduction

The ability to quickly prototype an application is critical in software development. Using prototypes, developers can evaluate their design decisions in a realistic fashion, solicit feedback from potential users to check if their applications meet the users' specifications or expectations, and shape the final designs and features before releasing actual products. The process of iterating over different prototypes can significantly improve the final user experience and save the cost of having to fix problems after release.

When developing a prototype, it is crucial to be able to demonstrate not only the UI of an application, but also the functionality of it. This is especially true for mid- to final-stage prototypes. In an early stage of development, it is perhaps acceptable to just show a set of static images of UI mockups [1] or a prototype with limited interactivity (e.g., mockup UI clicks and transitions). However, at later stages of development, it is necessary to be able to use a prototype and evaluate it in real-life situations [3]. A functional prototype is also necessary for software development outside commercial domains, such as academic research, where resource constraints often prevent investing in full-scale software development. Although leveraging open source projects can help in creating functional prototypes, it still requires significant development effort due to the need for extracting code from an unfamilar code base that is potentially large.

We present ProDroid, a framework for Android app developers to quickly prototype experimental features for ongoing app development. ProDroid enables developers to import different kinds of functionality from other *existing* Android apps without having the source code or understanding the internals of those apps. By allowing developers to leverage existing apps' functions quickly and easily, ProDroid enables developers to iterate over functional prototypes and test out different features rapidly. ProDroid makes this possible by adopting *programming by demonstration* [2] and combining it with our proposed *background execution of existing Android apps*.

ProDroid adopts programming by demonstration in its development tool, where a developer "demonstrates" a series of UI actions on an existing Android app. The UI actions are the ones that trigger the functionality that the developer wants to use from the existing app in her prototype. Once a developer demonstrates such UI actions, ProDroid generates a piece of code that the developer can embed into her prototype. This generated code is essentially a series of ProDroid commands that ProDroid executes at run time.

In order to execute those commands, ProDroid implements a new run-time system that directly performs UI actions on an existing app to ultimately execute the function that the UI actions trigger. The salient feature of this run-time system is that it performs all UI interactions with an existing app *completely in the background*, without displaying anything on the screen. This provides an illusion that a prototype tester is interacting with a single app, which is important when evaluating the UX of a prototype.

The main contributions of ProDroid are as follows. First, We design and develop a new Android app development tool for easy prototyping of functional app features. Second, We develop a technique to execute an existing app's functionality completely in the background. The novelty of our technique lies in converting user-visible, foreground tasks into background ones. Our technique works with existing apps on off-the-shelf Android devices; we do not impose any disruptive barrier to entry, such as operating system modifications or source code access to existing apps. Third, We evaluate the usefulness of ProDroid by conducting a developer study with three Android developers involving app feature prototyping. Our result shows that ProDroid has enabled a participant to import an app function from an existing app by writing only 55 lines of Java code. Without ProDroid, the participant would have needed to migrate 10,344 lines of Java code from an open-source code base.

2 ProDroid Overview

Figure 1 summarizes a prototype development process using ProDroid. We present three design principles (DP) and app development

MobiSys '19, June 17–21, 2019, Seoul, Republic of Korea

ACM ISBN 978-1-4503-6661-8/19/06.
https://doi.org/10.1145/3307334.3328621

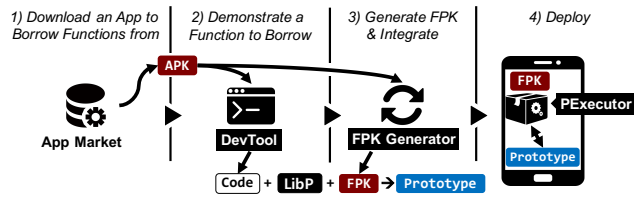

Figure 1: ProDroid usage model.

steps with the following example use case scenario: Alice and her team are developing a chatting app. Alice has an idea of a new app feature, smart snoozing, that snoozes chat notifications while users are asleep. To collect feedback from her team members, she decides to quickly build a functional prototype with ProDroid.

DP1 *Utilizing existing apps without manual modification.* To detect whether a user is sleeping or not, Alice wants to import sleep tracking functionality from an existing app. From an online app market, she finds a sleep tracker app that uses smartphone sensors to infer a user's sleep state. She downloads the sleep tracker app, and uses its package file (called APK) as input to ProDroid.

DP2 *Providing ease of programming for developers.* After obtaining the APK file, Alice feeds the file to ProDroid's developer tool named DevTool running on her computer. DevTool allows Alice to demonstrate UI actions that trigger the sleep tracking function provided by the downloaded sleep tracker app. To do so, DevTool launches the sleep tracker app in a special Android emulator that Alice can use to demonstrate UI actions. Using the emulator, Alice navigates through the sleep tracker app's UIs to where sleep states are displayed. Alice then informs DevTool that the UI action demonstration is finished and DevTool generates a corresponding code segment. This code segment uses ProDroid's API to describe ProDroid commands that perform demonstrated UI actions.Alice embeds this code segment into a new method that she implements for her prototype, `isSleeping()`, which returns `true` if 'sleeping" is displayed or `false` otherwise. Since ProDroid's API is provided as a library (named LibP), Alice links LibP with her chatting app, which now contains `isSleeping()` method.

ProDroid transforms an APK file into an ProDroid-compatible form named Functionality Package (FPK). ProDroid provides a tool named FPK Generator for this purpose. Thus, Alice runs FPK Generator on her computer to generate an FPK file for the sleep tracker app. At run time, ProDroid uses this FPK file to execute the sleep tracking functionality that Alice uses in her prototype app.

DP3 *Making new apps easily deployable on off-the-shelf Android devices.* To test Alice's new prototype, she and her team install two apps; Alice's new prototype and a special Android app named PExecutor provided by ProDroid. After installing the two apps, Alice or her team member uses the new prototype just like any regular Android app. When the prototype app must execute the embedded sleep tracking functionality, the app communicates with PExecutor and PExecutor executes the function in the background using the sleep tracking app's FPK. PExecutor does all of the above as a regular Android app, and there is no need to implement anything in the Android OS. This makes it possible to easily deploy our solution on off-the-shelf Android devices.

3 Developer Study

To evaluate the usability of ProDroid, we have conducted a user study with three Android app developers. We asked them to prototype an app feature by borrowing the function of reading an encrypted DB from an unfamiliar app. We provided ProDroid with ProDroid API documents. Before the experiment, we explained the basic mechanism of ProDroid and how to use the provided API. All three participants have Android app developing experience, while the quality of the developed apps and the depth of Android understanding vary widely. We labeled their skills by the longest lines of code they have ever contributed for a single Android project—beginner (< 1,000 LoC), intermediate (< 10,000 LoC), and otherwise.

Table 1: Developer study outcome.

ID	Android skill	LoC	Time	API understanding
P1	Intermediate	55	90 mins	5 (out of 5)
P2	Beginner	59	150 mins	5 (out of 5)
P3	Beginner	-	-	4 (out of 5)

For each developer, we measured how long one took to complete the task. As Table 1 shows, P1 successfully completed the task within 90 minutes, requiring only 55 lines of code. P2 had less Android experience and took 150 minutes to complete the task, with 59 lines of code. P3 had difficulties completing the task, as he barely had experience with the event-driven programming model of Android. Despite this, we observed that he had no trouble accessing provider app's UI components. Regardless, all three developers, including the last, rated their API understanding fairly well. According to our analysis, 10,334 lines Java/C code must be migrated from 28,996 lines of codebase to accomplish the same task without ProDroid. This shows that although prior development experience can influence the outcome, ProDroid API is easily usable, and ProDroid effectively lightens the development burden for the given prototyping task.

4 Conclusions

We have presented ProDroid that enables developers to quickly produce functional prototypes of an Android app. ProDroid repurposes the UI of an Android app as an interface where developers define functions to import. Compared to existing prototyping approaches, ProDroid is the only solution that enables the creation of a functional prototype and it requires neither the understanding of function implementation details nor access to source code for other Android apps where imported functions are implemented. We have conducted a usability study with three Android app developers and they have indicated that ProDroid provides an API that is easy to understand and usable. With ProDroid, a developer has imported an another app's function that requires 10,334 lines of Java code by writing only 55 lines of Java code. We believe ProDroid makes a step toward new mobile app prototyping where developers can quickly produce functional prototypes.

Acknowledgments

This research was financially supported by the Ministry of Trade, Industry and Energy(MOTIE) and Korea Institute for Advancement of Technology(KIAT) through the International Cooperative R&D program (N0002099). This work was also supported in part by the generous funding from the National Science Foundation, CNS-1350883 (CAREER) and CNS-1618531.

References

[1] 2019. Marvel app. https://marvelapp.com/.
[2] Allen Cypher and Daniel Conrad Halbert. 1993. Watch what I do: programming by demonstration. MIT press.
[3] M Cameron Jones, Ingbert R Floyd, and Michael B Twidale. 2009. Patchwork prototyping with open source software. In Software Applications: Concepts, Methodologies, Tools, and Applications. IGI Global, 1641–1656.