A Graph-based Dataset of Commit History of Real-World Android apps

Geiger, Franz-Xaver; Malavolta, Ivano; Pascarella, Luca; Palomba, Fabio; Di Nucci, Dario; Bacchelli, Alberto

Publication date
2018

Document Version
Peer reviewed version

Published in
Proceedings of the 15th International Conference on Mining Software Repositories, MSR. ACM, New York, NY

Citation (APA)

Important note
To cite this publication, please use the final published version (if applicable). Please check the document version above.
A Graph-based Dataset of Commit History of Real-World Android apps

Franz-Xaver Geiger
Vrije Universiteit Amsterdam
The Netherlands
f.geiger@student.vu.nl

Ivano Malavolta
Vrije Universiteit Amsterdam
The Netherlands
i.malavolta@vu.nl

Luca Pascarella
Delft University of Technology
The Netherlands
l.pascarella@tudelft.nl

Fabio Palomba
University of Zurich
Switzerland
palomba@ifi.uzh.ch

Dario Di Nucci
Vrije Universiteit Brussel
Belgium
ddinucci@vub.ac.be

Alberto Bacchelli
University of Zurich
Switzerland
bacchelli@ifi.uzh.ch

ABSTRACT

Obtaining a good dataset to conduct empirical studies on the engineering of Android apps is an open challenge. To start tackling this challenge, we present AndroidTimeMachine, the first, self-contained, publicly available dataset weaving spread-out data sources about real-world, open-source Android apps. Encoded as a graph-based database, AndroidTimeMachine concerns 8,431 real open-source Android apps and contains: (i) metadata about the apps' GitHub projects, (ii) Git repositories with full commit history and (iii) metadata extracted from the Google Play store, such as app ratings and permissions.

CCS CONCEPTS
• Software and its engineering → Maintaining software;

KEYWORDS
Android, Mining Software Repositories, Dataset

1 INTRODUCTION

Since mobile apps differ from traditional software and require to tackle new problems (e.g., power management and privacy protection [5, 7, 15, 16]), researchers are conducting empirical studies—especially by mining software repositories—to understand and support mobile software development.

As an example of recent research on apps, Malavolta et al. analyzed more than 11,000 apps published in the Google Play store and investigated the end users’ perceptions about various hybrid development frameworks [12]. Also, Linares-Vásquez et al. mined 54 Android apps from the Google Play store to find programming practices that may lead to an excessive energy consumption [5].

A common challenge when investigating apps is accessing candidate subjects (i.e., the app binaries or source code). A widely adopted approach is to gather information from open-source software (OSS) market places, F-Droid1 [4, 9, 13]. Nevertheless, relying on F-Droid impacts the number of projects that can be considered, as it only contains metadata of 2,697 apps.2 Moreover, for every study, researchers have to (i) systematically explore several online repositories to find analyzable apps, (ii) filter out source code not intended for the Android platform, and (iii) verify apps’ consistency within official distribution channels.

To improve this situation, we propose AndroidTimeMachine, a graph-based dataset with data linked from different sources concerning the development and publication process of 8,431 OSS Android apps. We combine information from GitHub and Google Play to create a unified dataset including (i) metadata of GitHub projects, (ii) full commit and code history, and (iii) metadata from the Google Play store. This dataset is the largest collection of published OSS Android apps with linked source code and store meta-data that we know of. The connected nature of this dataset and the included revision history allow a holistic view on OSS Android apps from development to publication on Google Play.

AndroidTimeMachine is composed of two main parts: A graph-based database (which facilitates understanding and navigation by focusing on links between apps, repositories, commits, and contributors) and a Git server hosting a mirror of all 8,431 GitHub repositories (thus providing a self-contained snapshot of the apps within the dataset). AndroidTimeMachine is publicly accessible at http://androidtimemachine.github.io and it is available as a Docker container image, which runs an instance of a Neo4J database with all the metadata and a GitLab server hosting all the mirrored GitHub repositories.

2 DATASET

Creating AndroidTimeMachine involved retrieving large quantities of information from several sources and combining it by linking it based on available identifiers. During this process we had to deal with limitations on how these sources select and publish data and how they restrict access, e.g., through rate limits. We detail the
process we used to identify the Android apps in our dataset (Section 2.1), the structure of our Neo4j database (Section 2.2), and the distribution of our dataset (Section 2.3). Furthermore, we showcase how the data can be used (Section 2.4) and point out limitations in Section 2.5.

2.1 Apps Identification

To create our dataset we defined a 4-step process (see Figure 1), which: (1) identifies open-source Android apps hosted on GitHub, (2) extracts their package names, (3) checks their availability on the Google Play store, and (4) matches each GitHub repository to its corresponding app entry in the Google Play store.¹

Figure 1: App Identification Process

Step 1. Identification of Android manifest files in GitHub.

Step 1 aims at finding all repositories on GitHub potentially containing the source code of an Android app. Since each Android app is required to contain an XML file named AndroidManifest.xml (which describes the app metadata and how it interacts with the Android system [11]), we performed this step by searching for AndroidManifest.xml files across all repositories on GitHub. Our search has been performed on the publicly-available GitHub mirror available in BigQuery.² This mirror contains information about files in all open-source repositories on GitHub, making it a good interface for finding repositories containing certain file types [3]. Our query returned 378,610 AndroidManifest.xml files across 124,068 repositories (search performed in October 2017).

Step 2. Extraction of Android package names.

Repositories may contain more than one manifest file, e.g., when they host the code of more than one app (e.g., free and paid versions) or include third-party code (e.g., libraries with their own manifest file). This complicates matching repositories to apps and warrants the heuristic algorithm in step 4. In every AndroidManifest.xml file, the root element must also include a package attribute containing the unique identifier of the app in the Google Play store. In this step we queried the BigQuery table containing the raw contents of all AndroidManifest.xml files and extracted the package names of their corresponding apps. The result of this query was a collection of 112,153 package names. This step still contained duplicated package names, mainly due to frequent usage of common names for test or toy projects, inclusion of libraries, or because repositories got forked [8]; this was taken care of in the following step(s).


In this step we aimed at excluding all test, library, or toy projects. By using the package name as app identifier, we filtered out all those apps for which there was no corresponding webpage in the Google Play store. This filtering step excluded all unpublished and non-existent package names, leading to 9,478 potentially-real app identifiers. Metadata for these apps was downloaded from the app store using a publicly available web scraper called node-google-play.³

Step 4. App-repository matching.

In this step, Google Play pages got mapped to GitHub repositories, via heuristics. We linked a package name to a repository if the repository was the only one containing an AndroidManifest.xml file for a given package name (77.1%). If more than one repository existed with the same package name, we searched metadata of the Google Play entry for mentions of GitHub repository URLs. We matched a repository to the package name if we found links to exactly one repository (6.6%). Finally, in cases in which neither of the two previous approaches resulted in a match, we selected the most popular repository based on number of forks, watchers, and subscribers (5.0%). We discarded 1,047 package names for which we could not determine a unique match or which were not accessible on GitHub anymore.

These four steps resulted in a collection of 8,431 real Android apps whose source code is available in 8,216 GitHub repositories.

2.2 Database Structure

To make data about OSS Android applications easily accessible and queryable, we designed and populated a graph-based database representing all the data gathered during the app identification process and the metadata related to each GitHub commit within the dataset (e.g., number of changes and contributors). The database is persistent using Neo4j (i.e., a graph DBMS), thus researchers can use algorithms from graph theory for investigations (e.g., reconstructing the chain of commits across the whole lifetime of the app and identifying apps in a certain category with at least n active developers in a certain timeframe); moreover, our dataset can be accessed: (i) with Cypher, a domain-specific graph query language, (ii) via a native Java API, and (iii) via a dedicated HTTP REST API.

Figure 2 shows the structure of the database. Data points are stored as nodes connected by relationships (i.e., the edges of the graph); both nodes and edges can have properties.

Node types and their properties. Android apps are represented as nodes of type App. They include the package name used to identify the app as string property id. The node type GooglePlayPage holds the metadata we mined from the Google Play entry of the app, such as its title, package name, average rating, and requested permissions. The GitHubRepository node represents a GitHub project with its id (i.e., the fixed internal identifier for repositories on GitHub). All other properties of GitHubRepository nodes represent a subset of data accessible through GitHub API v3, such as the owner, forks count, and repository name. A Commit node describes a commit of the GitHub repository. The id property is the full hash of the commit. The node also contains short id, number of changed lines (additions, deletions, total), as well as the commit title and message. Both authors and committers are represented by the node type Contributor. This node type has an email and a name property. Contributor nodes get merged by email, i.e., only the latest name seen during creation of the database is accessible. They can be differentiated by their relationship to a

¹[https://github.com/androidtimemachine/open_source_android_apps]
²[https://cloud.google.com/bigquery/public-data/github]
³[https://github.com/dweinstein/node-google-play-cli]
⁴[http://neo4j.com]
with pre-installed software necessary to show, explore, and query
As explained in Section 2, our dataset is composed of: (i) a Neo4j
environments with all dependencies included [2].
the data. Docker containers are a good way of sharing runnable
available a Docker-based containerized version of the entire data
GitHub
timestamp
Contributor
App
PUBLISHED_AT
edges between nodes and can contain properties. Relationships are directed graph
branches and merges of
relationship connects
Contributor
name
email
C
BElONGS_TO
PARENT
Commit
id
createdAt
message
short_id
package
COMMITS
AUTHORS
Branch
name
Tag
name
authors
versionCode
String
message
String
祗
String
String
String
RELATIONS
message
String
timestamp
name
String
Sarah
Lo"
https://hub.docker.com/r/androidtimemachine/neo4j_open_source_android_apps/
 https://androidtimemachine.github.io/dockerImages
https://github.com/AndroidTimeMachine/neo4jopen_source_android_apps/tree/master/data
https://docs.gitlab.com/ce/api/
https://neo4j.com/graphacademy/
We only considered applications available in the Google Play store. The graph representation used for structuring the data eases the analysis of the relationships between source code and metadata. The dataset is provided as Docker container to improve its accessibility and extensibility.

ACKNOWLEDGMENTS

Bacchelli and Palomba gratefully acknowledge the support of the Swiss National Science Foundation through the SNF Project No. PP00P2_170529. Di Nucci is funded by the Belgian Innoviris TeamUp project INTIMALS.

REFERENCES


