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

Empirical studies on the engineering of Android apps need to be based on open datasets and tools to allow comparisons, improve generalizability, and enable replicability. However, obtaining a good dataset is problematic and this state of things slows down empirical research on this topic.

In this paper, we contribute to overcome this challenge by presenting the first, self-contained, publicly available dataset weaving spread-out data sources about real-world, open-source Android apps. Our dataset is encoded as a graph-based database and contains the following information about 8,431 real open-source Android apps: (i) metadata about their GitHub projects, (ii) Git repositories with full commit history and (iii) metadata extracted from the Google Play store, such as app ratings and permissions. The dataset is available in Docker images to ease adoption.

KEYWORDS

Android, Mining Software Repositories, Dataset

1 INTRODUCTION

Mobile computing devices (*e.g.*, mobile phones and tablets) lure developers' attention by offering new outlets on the growing market of mobile apps (from here on, *apps*) [8]. However, apps differ from traditional software and require new solutions to new problems, *e.g.*, power management, privacy protection, software fragmentation [5, 9, 16, 17]. For this, researchers are investigating mobile software by mining software repositories and studying common issues.

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 [14]. 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 difficulty of researchers when investigating apps is the access to *candidate subjects* (*i.e.*, the app binaries or source code). A widely adopted approach is gathering information from open-source software (OSS) market places, F-Droid.[1] Unfortunately, relying on these market places impacts the number of projects that can be considered, as few developers actually publish their apps on such niche services [4, 11, 15]. For every study, researchers need to (i) systematically explore several online repositories to find analyzable apps, (ii) discard traditional software, and (iii) verify apps' consistency within official distribution channels.

To improve this situation, we propose 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 first large-scale collection of published OSS Android apps with linked source code and store meta-data. The connected nature of this dataset and the included revision history allows a holistic view on OSS Android apps from development to publication on Google Play.

From an implementation perspective, the dataset 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). The dataset is publicly available as a Docker container image to simplify reuse and extensions. The Docker image is self-contained and runs an instance of a Neo4J database with all the metadata and a GitLab server hosting all the mirrored GitHub repositories.

2 DATASET

Creating the dataset 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.

MSR'18, May 2018, Gothenburg, Sweden
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$0.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] https://f-droid.org/en/

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. [2]

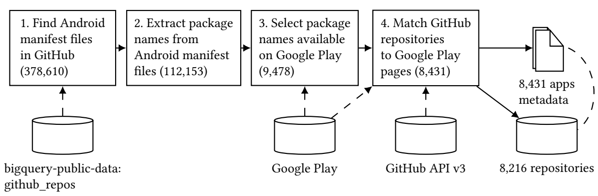

Figure 1: App Identification Process

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 [13]), we performed this step by searching for AndroidManifest.xml files across all repositories within the GitHub ecosystem. Our search has been performed on the publicly-available GitHub mirror available in BigQuery.[3] 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 step4. The root element of every AndroidManifest.xml file 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 [10]; this was taken care of in the following step(s).

Step 3. Selection of package names in Google Play. 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*.[4]

[2] All code and queries are available at https://github.com/af60f75b/open_source_android_apps

[3] https://cloud.google.com/bigquery/public-data/github

[4] https://github.com/dweinstein/node-google-play-cli

Step 4. App-repository matching. In this step, Google Play pages got mapped to GitHub repositories, via heuristics. Specifically, 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. 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. 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 (i) forks, (ii) watchers, and (iii) subscribers. We discarded 1,047 package names for which we could not determine a unique match.

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

A key objective of our work is to make data about OSS Android applications easily accessible and queryable. To that end, 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 persisted using Neo4j, a well-established graph database management system.[5] Given that we chose Neo4j for our dataset, researchers can easily reuse algorithms from graph theory for investigating new questions and problems (*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); morever, 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. These nodes include a string property id that is the package name used to identify the app. 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 property (*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 performed during the lifetime of the 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 Commit. Finally, Git tags and branches are stored as nodes of type Tag and

[5] http://neo4j.com

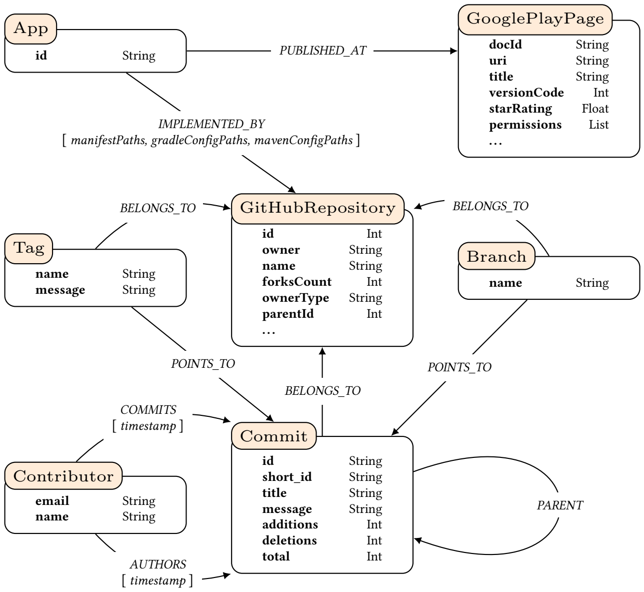

Figure 2: Schema of our dataset, as persisted in Neo4j (a graph database management system).

Branch, respectively. Both node types have a property name. Tags may also include the message stored with the tag.

Relationships between nodes.. Relatioships are directed graph edges between nodes and can contain properties. *PUBLISHED_AT* relations connect App nodes to their corresponding Google Play node. The link between an app and its corresponding GitHub repository (*IMPLEMENTED_BY*) contains the following properties: the paths to its Android manifest files (manifestPaths) and the paths to its build configuration files (gradleConfigPaths or mavenConfigPaths). Branches, tags, and commits are linked to a GitHub repository with edges of type *BELONGS_TO*. The *POINTS_TO* relation connects Branch and Tag nodes to a Commit. Version control history between commits is represented with the *PARENT* relation, which is a many-to-many relation due to the nature of branches and merges of Git. The *COMMITS* and *AUTHORS* relationship indicate the Contributor who authored and committed a change. Both relationships store a timestamp of their respective event.

2.3 Dataset Availability

As explained in Section 2, our dataset is composed of: (i) a Neo4j graph database with metadata of identified apps and (ii) a list of GitHub repositories. For ease of use and reproducibility, we make available a Docker-based containerized version of the entire data with pre-installed software necessary to show, explore, and query the data. Docker containers are a good way of sharing runnable environments with all dependencies included [2].

The total size of all Git repositories in the dataset is 136GB. Since not all researchers may need to access the full dataset, we split the data into two containers, where one Docker image contains the Neo4j database[6] and the second container serves as a snapshot of all GitHub repositories in the dataset cloned to a local Gitlab.[7]

2.4 Dataset Usage

Researchers can access our dataset through the Neo4j and Gitlab web interfaces, as well as through their respective REST-based APIs. The Gitlab web server and its API are accessible on port 80,[8] while the Neo4j instance can be accessed through default ports 7474 for the HTTP protocol and port 7687 for the bolt protocol used for *Cypher* queries.[9] In the Neo4j database, the snapshot attribute of *GitHubRepository* nodes links to the address of the corresponding repository in our GitLab instance. Documentation on how to run the container and access the data is in the Docker image repository.[6]

The connected nature of the graph database facilitates many potential research questions. In the following we showcase queries and analyses supported by our dataset.[10]

Scenario 1: Select all apps belonging to the *Finance* category with more than 10 commits in a given week.

```
WITH apoc.date.parse('2017-01-01', 's', 'yyyy-MM-dd')
    as begin,
  apoc.date.parse('2017-01-08', 's', 'yyyy-MM-dd')
    as end
MATCH (p:GooglePlayPage)<-[:PUBLISHED_AT]-
  (a:App)-[*]-(:Commit)<-[c:COMMITS]-()
WHERE 'Finance' in p.appCategory
  AND start <= c.timestamp < end
WITH a, SIZE(COLLECT(DISTINCT c)) as commitCount
WHERE commitCount > 10
RETURN a.id, commitCount
```

Scenario 2. Select all contributors who worked on more than one app in a given month.

```
WITH apoc.date.parse('2017-07-01', 's', 'yyyy-MM-dd')
    as start,
  apoc.date.parse('2017-08-01', 's', 'yyyy-MM-dd')
    as end
MATCH (app1:App)-[*]-(:Commit)<-[c1:COMMITS|AUTHORS]-
  (c:Contributor)
  -[c2:COMMITS|AUTHORS]->(:Commit)-[*]-(app2:App)
WHERE a1.id != a2.id
  AND start <= c1.timestamp > end
  AND start <= c2.timestamp > end
RETURN contrib
```

Scenario 3. Providing our dataset in containerized form allows future research to easily augment the data and combine it for new insights. The following is a very simple example showcasing this possibility. Assuming all commits have been tagged with self-reported activity of developers, select all commits in which the developer is fixing a performance bug. We apply a simple tagger, but a more advanced model (*e.g.*, [4]) would lead to better results.

```
MATCH (c:Commit)
WHERE c.message CONTAINS 'performance'
SET c :PerformanceFix
```

Also, given these additional labels, performance related fixes can then be used in any kind of query via the following snippet.

[6] https://hub.docker.com/r/af60f75b/neo4j_open_source_android_apps/

[7] https://www.dropbox.com/s/e9ld70s72mxc85y/docker_gitlab_open_source_android_apps.tar.gz?dl=0

[8] Username: root – password: gitlab. Documentation of the Gitlab API is available in the container at endpoint /help/api/README.md and a potentially newer version at https://docs.gitlab.com/ce/api/

[9] Neo4j documentation available at https://neo4j.com/graphacademy/

[10] Some of the examples rely on the Neo4j plugin *APOC*, which can be installed by mapping an external directory into the Docker image: https://guides.neo4j.com/apoc

```
MATCH (c:Commit:PerformanceFix) RETURN c
```

Scenario 4. Metadata from GitHub and Google Play can be combined and compared. Both platforms have popularity measures, *e.g.*, star ratings, which are returned by the following query.

```
MATCH (r:GitHubRepository)<-[:IMPLEMENTED_BY]-
   (a:App)-[:PUBLISHED_AT]->(p:GooglePlayPage)
RETURN a.id, p.starRating, r.forksCount,
   r.stargazersCount, r.subscribersCount,
   r.watchersCount, r.networkCount
```

Scenario 5. Is a higher number of contributors related to the success of an app? The following query returns the average review rating on Google Play and the number of contributors to the source code of each app within the dataset.

```
MATCH (c:Contributor)-[*]-
   (a:App)-[:PUBLISHED_AT]->(play:GooglePlayPage)
WITH a, SIZE(COLLECT(DISTINCT c)) as contribCount
RETURN a:id, p.starRating, contribCount
```

2.5 Dataset Limitations

The dataset we created for this selection of OSS Android apps has some limitations.

First, we only considered applications available in the Google Play store. This limitation is mitigated by the fact that Google Play is the official Android app store and offers the largest selection of Android apps [1]. Another limitation regards the access restriction decided by the developers, *e.g.*, based on location. We mined Google Play from a server in our region, thus limiting the data collection to the apps available here.

Data selection can be biased by the presence of the source code on GitHub. We consider this acceptable considering that, in the recent years, GitHub has been the most known platform for the open-source community and it offers a large and diverse selection of OSS projects [6]. Yet, we cannot exclude that we missed some open-source unlicensed repositories.

During the data collection we faced and addressed several challenges. For example, searching candidate repositories using the GitHub API was not possible due to limitations on the number of results returned by each query. Indeed, even when stratifying search queries (*e.g.*, by filesize, with a byte-level granularity), not all the results could be retrieved. We overcame this issue by using BigQuery.

3 RELATED WORK

Previous studies created data collections of OSS Android applications. For their study on app releases, Nayebi *et al.* [15] linked 69 F-Droid apps with version control repositories. Where available, metadata from Google Play was included. A similar dataset of OSS Android apps was constructed by Krutz *et al.* [11] to facilitate security research [12]. It includes versioned data of 1,179 open-source apps augmented with metadata from F-Droid. Moreover, statistical analysis data of the apps are provided. Das *et al.* [4] used F-Droid as a starting point for identifying open-source Android apps. They built a dataset for the analysis of performance related commits of mobile applications by matching apps listed on F-Droid against GitHub repositories. Later, the apps were filtered considering their availability on Google Play. The final dataset was composed of 2,443 apps.

These dataset are hindered by the inherent limitation of relying on F-Droid. In our dataset, we did not considered F-Droid and identified candidate repositories by searching the AndroidManifest.xml file; this approach is able to provide a more realistic sample of open-source Android apps.

4 CONCLUSIONS AND FUTURE WORK

We compiled a comprehensive dataset of 8,431 real-world open-source Android apps. It combines source and commit history information available on GitHub with the metadata from Google Play store. The graph representation used for structuring the data eases the analysis of the relationships between source code and metadata. The dataset can be used as a starting point for future studies considering that it provided as Docker container, being thus easily accessible and extensible.

To sum up, the contributions made in this paper are:

(1) A comprehensive dataset of 8,431 real Android apps, containing information about (i) metadata about their Github projects, (ii) Git repositories with full commit history, and (iii) metadata from the Google Play store.
(2) A graph-based database which facilitates the access and navigation of the dataset. It also comprises a Git server that hosts a mirror of all the Github repositories considered.
(3) A Docker container image that simplifies the use and extension of the provided dataset.

In the future, we plan to maintain and extend the current dataset with information coming from the user reviews of the considered apps.

REFERENCES

[1] Ben Martin. 2017. The Global Mobile Report - comScore's cross-market comparison of mobile trends and behaviours. (2017). ComsCore white paper.
[2] Ryan Chamberlain and Jennifer Schommer. 2014. Using Docker to support reproducible research. *DOI: https://doi. org/10.6084/m9. figshare* 1101910 (2014), 44.
[3] Jürgen Cito, Gerald Schermann, John Erik Wittern, Philipp Leitner, Sali Zumberi, and Harald C. Gall. 2017. An empirical analysis of the Docker container ecosystem on GitHub. In *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 323–333.
[4] Teerath Das, Massimiliano Di Penta, and Ivano Malavolta. 2016. A Quantitative and Qualitative Investigation of Performance-Related Commits in Android Apps. In *2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016*. 443–447.
[5] Dario Di Nucci, Fabio Palomba, Antonio Prota, Annibale Panichella, Andy Zaidman, and Andrea De Lucia. 2017. Software-based energy profiling of android apps: Simple, efficient and reliable?. In *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*. IEEE, 103–114.
[6] Georgios Gousios and Diomidis Spinellis. 2017. Mining software engineering data from GitHub. In *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, 501–502.
[7] Harald Gruber and Frank Verboven. 2001. The diffusion of mobile telecommunications services in the European Union. *European Economic Review* 45, 3 (2001), 577–588.
[8] Adrian Holzer and Jan Ondrus. 2011. Mobile application market: A developer's perspective. *Telematics and informatics* 28, 1 (2011), 22–31.
[9] Mona Erfani Joorabchi, Ali Mesbah, and Philippe Kruchten. 2013. Real challenges in mobile app development. In *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*. IEEE, 15–24.
[10] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21, 5 (2016), 2035–2071.
[11] Daniel E. Krutz, Mehdi Mirakhorli, Samuel A. Malachowsky, Andres Ruiz, Jacob Peterson, Andrew Filipski, and Jared Smith. 2015. A dataset of open-source

Android applications. In *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 522–525.

[12] Daniel E. Krutz, Nuthan Munaiah, Anthony Peruma, and Mohamed Wiem Mkaouer. 2017. Who Added That Permission to My App? An Analysis of Developer Permission Changes in Open Source Android Apps. IEEE, 165–169. https://doi.org/10.1109/MOBILESoft.2017.5

[13] Li Li. 2017. Mining androzoo: A retrospect. In *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*. IEEE, 675–680.

[14] Ivano Malavolta, Stefano Ruberto, Tommaso Soru, and Valerio Terragni. 2015. Hybrid mobile apps in the google play store: An exploratory investigation. In *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems*. IEEE Press, 56–59.

[15] Maleknaz Nayebi, Homayoon Farrahi, and Guenther Ruhe. 2016. Analysis of marketed versus not-marketed mobile app releases. In *Proceedings of the 4th International Workshop on Release Engineering*. ACM, 1–4.

[16] Christoffer Rosen and Emad Shihab. 2016. What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering* 21, 3 (2016), 1192–1223.

[17] Anthony I Wasserman. 2010. Software engineering issues for mobile application development. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, 397–400.