Self-Reported Activities of Android Developers

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

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

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

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

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

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

ABSTRACT

To gain a deeper empirical understanding of how developers work on Android apps, we investigate self-reported activities of Android developers and to what extent these activities can be classified with machine learning techniques. To this aim, we firstly create a taxonomy of self-reported activities coming from the manual analysis of 5,000 commit messages from 8,280 Android apps. Then, we study the frequency of each category of self-reported activities identified in the taxonomy, and investigate the feasibility of an automated classification approach. Our findings can inform be used by both practitioners and researchers to take informed decisions or support other software engineering activities.

KEYWORDS

Android, Empirical Study, Mining Software Repositories

1 INTRODUCTION

Developing Android apps is fundamentally different from developing other types of software [11, 16, 41, 51]: On the one hand, even the smallest error may have quick and large effects (such as negative user reviews, with subsequent loss of future users [32]); on the other hand, Android apps have to deal with potential interaction with other apps, heavy usage of sensors like accelerometer and GPS, limited battery life, limited display size, and so forth. This inherent difference in the development of Android apps limits the possibility to use results, off-the-shelf, from software engineering research done on other kind of software systems. Instead, to use and guide our research to support the engineering of Android apps, first *we need to gain a novel, deeper empirical understanding of how developers work on these apps*.

Our **goal**, in line with this need, is to investigate and understand the various types of activities performed by Android developers in the context of real projects. We focus on *self-reported activities*, which represent one of the most valid ways to comprehend and analyze the development process [22]. As done in previous studies, we tap into the commit messages left by Android developers in GitHub repositories, as a way to study these self-reported activities. Past research has focused on specific aspects of Android apps such as performance [8] and energy consumption [3, 28], here we continue on this line, but broaden the scope to any type of activity, as done for open-source software [39].

Our **research method** follows that of an exploratory investigation, *i.e.*, we started without hypotheses about the contents of the GitHub commit messages and made the types of development activities emerge from the extracted data [52]. To this purpose, we firstly built a dataset of 8,280 Android apps (which are both open-source in GitHub and distributed through the Google Play store) and randomly selected 5,000 commits from their repositories. Then, we (i) manually inspected and categorized all the commits by conducting independent content analysis sessions involving 5 researchers, (ii) collaboratively merged the independently-identified categories into a single taxonomy, (iii) validated the obtained taxonomy with external mobile app developers, (iv) analyzed the frequency of each category in the taxonomy across the 5,000 commits, and (v) investigated how effectively these commits can be automatically classified via standard machine learning techniques.

Our **results** show that Android developers reportedly perform a wide variety of different activities at different levels of abstraction, ranging from bug fixes, release management, access to sensors, *etc.* The most prominent category of activities is app enhancement (new and updated features), followed by bug fixing (mostly in an app-specific manner) and project management (mostly by merging/branching of the repository and by preparing a new app release). Those results confirm the importance of research related to feature management and release planning of Android apps, Android-specific program analyses, and software repository mining. Finally, the automated classification reaches promising initial results.

The **main contributions** of this study are the following:

(1) A taxonomy of self-reported activities performed by Android developers when evolving their apps;
(2) An empirical analysis of the frequency of the self-reported activities performed by Android developers aimed at understanding their main concerns when evolving their apps;
(3) An automated approach for classifying commit messages according to the defined taxonomy.
(4) A comprehensive replication package containing the raw data, analysis scripts, and the automatic classifier produced in our research.[1]

Conference'17, July 2017, Washington, DC, USA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] https://figshare.com/articles/Self-Reported_Activities_of_Android_Developers/5802909

2 METHODOLOGY

The *goal* of the study is to empirically investigate and classify the activities performed by Android developers reported within commit messages, with the *purpose* of understanding the typical actions they perform and easing a variety of decision making mechanisms (*e.g.*, code review triaging or monitoring of the development process). The *perspective* is of both researchers and practitioners, interested in an empirical understanding of the activities performed during the development process.

The choice of considering self-reported activities is driven by the recent advances in program comprehension [22], which demonstrated that the analysis of commit messages represents one of the most valid strategies to comprehend and analyze the development process of a software system. Our study revolves around three research questions and follows well-established guidelines on empirical software engineering [46, 52].

In the first place, we aim at categorizing the developers' self-reported activities through the analysis of commit messages that accompany the changes performed while evolving Android apps:

> **RQ1.** How can self-reported activities of Android developers be categorized?

After having categorized the self-reported activities, we analyze the frequency of each category to quantify the different developers' concerns when developing Android apps:

> **RQ2.** How often does each category of self-reported activities occur?

Finally, we investigate how effectively self-reported activities can be automatically classified from commits via standard machine learning techniques, so that developers and project managers can be automatically supported during their decision making processes:

> **RQ3.** How effective is an automated approach, based on machine learning, in classifying self-reported activities?

In the following subsections, we detail the design choices that allow us to answer our research questions.

2.1 Context Selection and Dataset Creation

We study self-reported activities based on commit messages authored by developers, thus we need *real-world Android applications for which commit history is available*. To ensure the analysis of a proper set of mobile apps having different size and scope as well as being published on the Google Play store, we design and conduct the selection process shown in Figure 1.

In step 1 we identify the GitHub repositories containing the source code of Android applications. Then, to properly link a GitHub repository to its corresponding app in Google Play, we exploit the Android manifest file (step 2). In fact, every Android app must have an AndroidManifest.xml file that includes a *package name* that identifies the application and serves as an identifier of the app on Google Play. The data concerning all open-source repositories

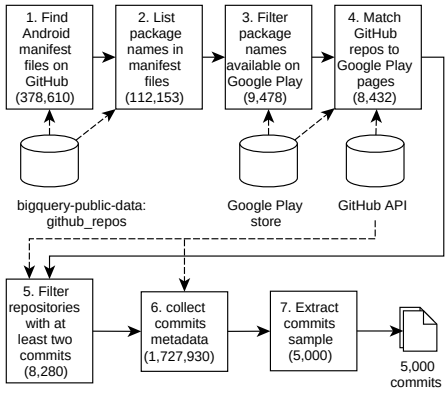

Figure 1: Dataset creation process

on GitHub are available in databases on Google BigQuery.[2] BigQuery list 378,610 AndroidManifest.xml files on GitHub with 112,153 unique package names. Duplication of package names may occur because of forked projects, the frequent usage of example names, or inclusion of manifest files from third-party code [17].

In step 3 we remove unpublished and non-existent apps by checking the existence of the corresponding page on Google Play. As a result, 9,478 package names are listed as apps in Google Play. For some of these apps, one or more repositories contain a matching AndroidManifest.xml file, as detailed above. In step 4 we match the repositories to Google Play entries with an heuristic approach:

(1) if only one repository contains a manifest for a package name, we assume it hosts the code for the app;
(2) if more than one repository with the manifest file exists, we search for links from Google Play meta-data of the app to any of the GitHub repositories. If we find a distinct repository the app entry linked to, we assume the repository to be the canonical source for the app;
(3) if no such unique link exists, we select the most popular repository based on number of (i) forks, (ii) watchers, and (iii) subscribers, as listed by GitHub.

During step 4, we remove the apps (718) for which we cannot determine a canonical repository, thus reducing the total number of apps we investigate (8,432). In step 5, we exclude repositories with fewer than 2 commits (152) to exclude unmaintained, toy, or demo projects [17]. Our final dataset consists of a total 8,280 mobile apps covering all 34 categories of the Google Play store, for a total of 1,727,930 commits belonging to the main branch of these apps' GitHub repositories (step 6).

In step 7, we randomly select a sample of 5,000 commits that cover 30 categories of Google Play, since the manual analysis of all the collected commits is infeasible. This sample represents a 99% statistically significant sample with a 1% confidence interval of the total number of commits belonging to the dataset.

2.2 RQ1. Self-reported activities categorization

To answer our first research question, we conduct three iterative *content analysis sessions* [19] involving five software engineering

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

researchers, all authors of this paper, (2 graduate students, 2 research associates, and 1 faculty member) with at least five years of programming experience. From now on, we refer to them as *inspectors*. We describe the methodology for these three iterative sessions, followed by the validation method.

Taxonomy Building. Starting from the set of 5,000 commits composing our dataset, overall each inspector *independently* analyzes 1,000 commits.

Iteration 1: The inspectors analyze an initial set of 300 commit messages. Then, they open a discussion on the labels assigned so far and try to reach a consensus on the names and types of the categories assigned. The output of this step is a draft taxonomy that contains some obvious categories (*e.g.*, changes to the Graphical User Interface), while others remain undecided.

Iteration 2: The inspectors firstly re-categorize the 300 initial commits according to the decisions taken during the discussion, then use the draft taxonomy as basis for categorizing another set of 500. This phase is for both assessing the validity of the codes coming from the first step—by confirming some of them and redefining others—and for discovering new codes. After the completion, the inspectors open a new discussion aimed at refining the draft taxonomy, merging overlapping categories or characterizing better the existing codes. A second version of the taxonomy is produced.

Iteration 3: The inspectors re-categorize the 800 commits previously analyzed. Afterwards, they complete the final draft of the taxonomy verifying that each kind of commit message encountered in the final 200 commits is covered by the taxonomy.

Following this iterative process, we defined a hierarchical taxonomy composed of two layers. The top layer consists of 9 categories, while the inner layer contains of 49 subcategories.

Taxonomy Validation. In addition to the iterative content analysis process, we also externally validate the defined taxonomy. To this aim, we involved 2 professional developers having 4 and 5 years of Android programming experience, respectively. They were contacted via e-mail by one of the authors of this paper, who selected them from her personal contacts.

We provided them with a spreadsheet containing a list of 200 commit messages randomly selected from the total 5,000 in the dataset and asked to categorize the commits according to the taxonomy we previously built. During this step, the developers were allowed to either consult the taxonomy (provided in PDF format and containing a description of the commit categories in our taxonomy similar to the one we discuss in Section 3.1) or assign new codes if needed.

Once the task was completed, the developers sent back the spreadsheet file annotated with their categorization. Moreover, we gathered comments on the taxonomy and the classification task. As a result, both the participants found the taxonomy *clear* and *complete*: As a proof of that, the tags they assigned were exactly the same as the ones assigned during the phase of taxonomy building.

2.3 RQ2. Frequency of self-reported activities

In this research question, we aim at analyzing how frequently each category of our taxonomy appears. To this aim, we compute the frequency each category of activities was assigned to a commit message during the iterative content analysis.

In this way, we can overview the main developers' concerns when evolving mobile apps and identify the most popular self-reported activities. In Section 3 we present and discuss bar plots showing the frequency of each category in the taxonomy.

2.4 RQ3. Automated classification of activities

With our final research question we test standard machine learning techniques to automatically classify self-reported activities. As a side effect, the output of this research question poses a baseline against which future approaches aimed at more accurately classifying commit messages can be tested.

While several techniques can classify text of self-reported activities (*e.g.*, keyword-based approaches [49]), we use machine learning since this type of approach can automatically learn the features discriminating the a certain category, thus simulating the behavior of a human expert [36]. Overall, machine learning is a method (supervised, in our case) where a set of independent variables (the *predictors*) are used to predict the value of a dependent variable (in our case, the commit *classification*) using a machine learning classifier (*e.g.*, Logistic Regression [30]). The following subsections detail the design decisions taken to build and validate our approach.

Independent Variables. Our goal is to classify the nature of self-reported activities based on commit messages: the basic information for the classification is therefore given by the words characterizing the commit message. However, not all the words in a commit can be actually representative for the classification of the self-reported activity. For this reason, we need to properly preprocess them [6].

In the context of our work, we use the widespread *Term Frequency - Inverse Document Frequency* (TF-IDF) model [42], which is a weighting mechanism that determines the relative frequency of words in a specific document (*i.e.*, a commit message) compared to the inverse proportion of that word over the entire document corpus (*i.e.*, the whole set of commit messages in our dataset). This approach measures *how characterizing* a given word is in a commit message: For instance, articles and prepositions tend to have a lower TF-IDF since they generally appear in more documents than words used to describe specific actions [42]. More formally, let C be the collection of all the commit messages in our dataset, let w be a word, and let $c \in C$ be a single commit message, the TF-IDF algorithm computes the relevance of w in c as:

$$relevance(w, c) = f_{w,c} \cdot \log(|C|/f_{w,C}) \quad (1)$$

where $f_{w,c}$ equals the number of times w appears in c, $|C|$ is the size of the corpus, and $f_{w,C}$ is equal to the number of documents in which w appears. The weighted words given as output from TF-IDF represent the independent variables for the machine learner.

Dependent Variables. The category of a self-reported activity is the variable to predict. We set the granularity of the dependent variable to the top layer of the taxonomy, *i.e.*, the one reporting the 9 main categories of self-reported activities in our taxonomy.

Machine Learners. In our context, a certain self-reported activity might refer to more than one category: For instance, suppose that in a commit a developer performs both an enhancement and a

bug fix. This is a target for *multi-label* classifiers [50]. These can be of two types: (i) *problem transformation methods*, which transform the multi-label classification into a more single-label classifications, and (ii) *algorithm adaptation methods*, which extend specific classifiers in order to handle multi-label data [50].

Since the two types of multi-label algorithms have similar performance [40], we adopt a problem transformation strategy and use the OneVsRest classifier [15]. Taking as input a standard single-label algorithm, OneVsRest wraps up the process of training a classifier for each possible class. As a result, the input classifier assigns a probability that a certain commit message belongs to each of the categories of our top layer taxonomy: If the probability is higher than 0.5, then the commit message is considered as belonging to it.

With the aim of providing a wider overview of the performance achievable by different single-label classifiers when adopted in combination with OneVsRest, we consider (i) Naive Bayes, (ii) Support Vector Machines (SVM), (iii) Logistic Regression, and (iv) Random Forest. These classifiers make different assumptions on the underlying data, as well as have different advantages and drawbacks in terms of execution speed and overfitting [30]. Before running the models, we identify their best configuration using the Grid Search algorithm [4].

Evaluation Strategy and Metrics. To assess the performance of the proposed machine learning approach, we adopt the *10-Fold Cross Validation* [47]. This methodology randomly partitions the data into 10 folds of equal size, applying a stratified sampling (*i.e.*, each fold has the same proportion of self-reported activity categories). A single fold is used as test set, while the remaining ones are used as training set. The process is repeated 10 times, using each time a different fold as test set. Then, the model performance is reported using the mean achieved over the ten runs.

The performance of the experimented models are reported using widespread classification metrics such as *precision*, *recall*, and *F-Measure* (the harmonic mean between precision and recall) [2].

2.5 Threats to Validity

We report possible threats to the validity of the study and how we mitigated them.

Taxonomy validity. To ensure that the correctness and completeness of the categories of self-reported activities identified, we iteratively built the taxonomy by merging and splitting categories if needed. As an additional validation, we asked 2 professional developers to classify a set of 200 commits according to the proposed taxonomy. They assigned to the sampled commits the same categories as the ones assigned during the phase of taxonomy building, also reporting the completeness and clarity of the categories we identified. We cannot exclude the missing analysis of specific commit types out of the categories identified, however the validation session gives us confidence of the reliability of the taxonomy.

Automated approach validity. To build a multi-label classification technique, we exploited the OneVsRest method [15], which has been shown to have similar performance than other approaches [40]. Problems like multi-collinearity [31] are mitigated by the preprocessing and the TF-IDF modeling.

To provide an overview of the performance achieved when using OneVsRest in combination with different single-label classifiers, we tested four categories of machine learners.

External validity. As for the generalizability of the results, we conducted this study on a statistically significant sample of 5,000 belonging to 8,280 open-source mobile apps that are published on the Google Play store. The proposed taxonomy may differ when considering closed-source apps; at the same time, the performance of the experimented automatic approach might be lower/higher than the one reported herein.

3 RESULTS

We report the results of our study by research question.

3.1 RQ1. Categories of self-reported activities

The manual analysis of the 5,000 commits led to the creation of the taxonomy of Android developers activities shown in Figure 2. The taxonomy is composed of two layers: The top layer (9 items) groups together activities with similar overall purpose (*e.g.*, app enhancement, bug fixing), whereas the subcategories (49 items) in the lower level provide a finer-grained categorization. In the following we describe each category with the corresponding subcategories.

A. App enhancement. This category represents the activities aimed at adding or improving existing features of the mobile app. This is clearly at the core of mobile apps development and, as we will see in Section 3.2, its related commits involve a large number of changed source code files.

> Example commit. *"[Wear] Implemented Favourites feature for wearable companion app."* - thecosmicfrog/LuasataGlance (commit: 57c92a8784db5ac003af82b91aaee2135f41c3c4)

A.1 - New feature: Implementation of new app features (*e.g.*, a new screen for sharing a content on social media). In the commit messages developers mostly describe the newly added feature, without implementation details.

A.2 - Feature changes: Activities referring to the change or enhancement of already existing features of the mobile app. By looking at the commit messages, these are more related to changes in the business logic of the mobile app, rather than about bug fixes or code refactoring.

A.3 - Usability: Activities related to changes aimed at improving the usability and user experience of the app. This category is different from the category E since here developers are focusing on the business logic (*e.g.*, how to share a content with less taps), whereas in E they are focusing on the presentation (*e.g.*, colors).

A.4 - Language: Activities related to internationalization, translations of textual contents, *etc.*. Mostly, the commit messages explicitly refer to the support of additional languages and the refinement of existing translations.

A.5 - Android lifecycle: Activities about the management of Android components lifecycle events and transitions. In the commit messages developers refer to technical aspects related to the Android programming model, such as the onCreate method.

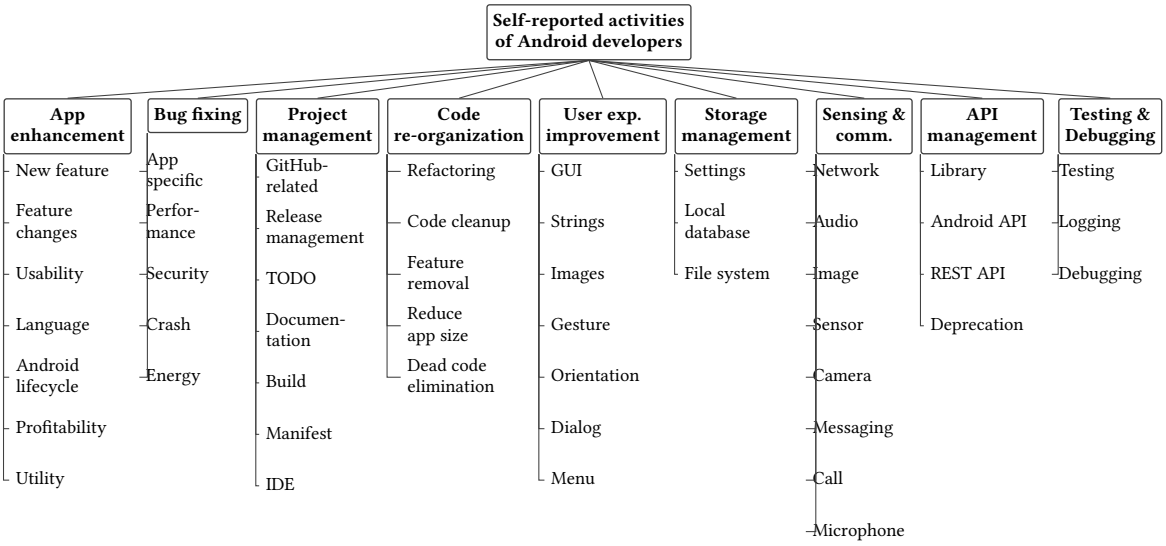

Figure 2: Taxonomy of self-reported activities of Android developers

A.6 - Profitability: Developers add/improve profitability aspects of the app. In the commit messages developers refer to activities such as adding ways to receive donations and displaying ads.

A.7 - Utility: Developers mention utility classes or methods, potentially used across the whole app, such as those for serializing/deserializing dates, strings manipulation, and app-specific exception handlers.

B. Bug fixing. This category represents development activities where app issues that appear in the mobile app are fixed.

> Example commit. *"v2.2 Testing new fix for massive battery usage caused by GPS not being disabled during application pause."* - GrahamBlanshard/WiseRadar (commit: 7c35abb8512bb89b65750175e4ab07c26a813677)

B.1 - App specific: Bux fixing specific to the domain of the app. Commits belonging to these category do not relate to generic software qualities (*e.g.*, performance), yet they have been marked as bug fixes.

B.2 - Performance: Activities aiming at improving the performance. Examples of commits in this category are Android wake locks, memory leaks and optimization of string operations.

B.3 - Security: Fixing security issues in the app. Example commits belonging to this category include sanitizing the input provided by users and removal of unused permissions.

B.4 - Crash: Fixing crash problems. Commits in this category are at different levels of abstraction, ranging from fixing null pointer exceptions to correcting issues for specific Android devices.

B.5 - Energy: Optimizing battery consumption optimization and managing potential energy leaks. Commits included in this category mostly include refactoring of the code in terms of better use of sensors (*e.g.*, GPS) and WiFi as well as Bluetooth scanning.

C. Project management. In this category developers manage app releases, documentation, the build process, the GitHub repository itself (*e.g.*, merges), IDE-related issues (*e.g.*, Android Lint configuration).

> Example commit. *"Merge pull request #4 from RyDroid/readme, Update of README"* - uberspot/AnagramSolver (commit: 322ca43654065ca00d1a8757059154cd1c5d1155)

C.1 - GitHub-related: GitHub-specific aspects of the project. The commits in this category mostly mention the creation/merging of branches and the execution of the first commit in the repository.

C.2 - Release management: Activities to prepare a new app release. The commit messages deal, for example, with changing the app version number and preparing a new entry in the changelog.

C.3 - TODO: Activities on future actions to be done as potential enhancements or fixes. In this cases, commit messages deal both with low-level items (*e.g.*, removal of a code smell) and higher-level concerns (*e.g.*, implementation of a new feature).

C.4 - Documentation: Activities on the documentation of the app. Commit messages in this category mainly deal with adding/refining comments in the source code and the documentation of the app (*e.g.*, description of the app functionalities, its requirements, UI mockups).

C.5 - Build: Activities on improving project compilation. Commit messages related to these activities usually relate to the creation of the app binaries (*i.e.*, its APK file), rules for building the app, and migration to/from building systems.

C.6 - Manifest: Changes in the Android manifest of the app. Usually, commits belonging to this category concern updating the target SDK of the app, cleanup of default unused tags in the manifest,and adding/changing views definitions in the manifest.

C.7 - IDE: Activities related to the configuration of the IDE (*e.g.*, Android Studio, Eclipse). Mainly commit messages include the definition of a new Eclipse project for the app and upgrade of the latest version of the IDE.

D. Code re-organization. These activities are aimed at improving the structure, size, and readability of the code (*e.g.*, refactoring, cleaning, improvement of the code) or of the project organization, without changing behavior of the app.

> Example commit. *"Refactored the PlayerService by moving parts of the code into smaller classes"* - bottiger/SoundWaves (commit: a1911b5229ce1d1c3b5ca11066c8c32e14c5cf68)

D.1 - Refactoring: Refactoring of the source code. Messages usually refer to moving code to specific methods of the lifecycle of Android activities, import statements reorganization and extracting methods from classes.

D.2 - Code cleanup: Source code cleaning activities (lighter weight than refactoring). Commits include the removal of unused API keys or unused string resources and deletion of dummy objects.

D.3 - Feature removal: Activities in which some features of the app are removed. Messages are mostly about *what* has been removed, not about *why*.

D.4 - Reduce app size: Activities aimed at reducing the app size to make it a more lightweight download. Commit messages belonging to this category mostly regard the removal of unused files or media resources.

D.5 - Dead code elimination: Elimination of source code never executed at run-time. Commit messages refer mostly to removing legacy Android activities, unused layouts and variable assignments, as well as redundant initialization code.

E. User experience improvement. This category represents the activities related to the user experience of the app, including screen layouts, elements colors and padding, text boxes appearance, buttons, messages shown to the user, as well as gestures support.

> Example commit. *"Increase the opacity of the showcase background to 96% (in line with material guidelines)"* - ccomeaux/boardgamegeek4android (commit: 016ea1ee32dea3351be49c12bcc215f231039380)

E.1 - GUI: Changes to the graphical user interface aimed at improving the user experience. Commit messages in this category are mostly about color schemes, buttons and UI layout.

E.2 - Strings: Activities related to the management of strings in the app. In the commit messages in this category usually developers discuss about static strings shown to the user.

E.3 - Images: Activities related to graphic elements such as icons, images, and graphics shown to the user. Commit messages are usually about changes of icon sets, addition or change of fixed images shown to the user, and logos.

E.4 - Gesture: Management of gestures of the users. Messages are mostly about features such as scroll to refresh, swipe for performing some action, and disabling scroll in some specific parts of the app.

E.5 - Orientation: Management of device orientation. Messages are mostly about the detection of orientation change (*e.g.*, for playing a video), the creation of a dedicated layout for landscape orientation, and the margins when in landscape mode.

E.6 - Dialog: Activities related to dialogs, toasts, and pop-ups used to show notifications to the user. Messages usually concern adding/removing confirmation dialogs, adding toasts for giving feedback to the user, and fixing the style of pop-ups.

E.7 - Menu: Activities related to menus and navigation bars in the UI. Commit messages usually regard adding/removing menu items, reordering items in navigation bars as well as menus, and adding contextual menus where needed.

F. Storage management. This category of activities concerns changes involving archives, access to the file system, files storage, local settings, and persisting data via local database.

> Example commit. *"Don't overwrite DB Status LOCAL_CREATED with LOCAL_EDITED. To prevent errors on synchronization (create -> edit -> sync)"* - stefan-niedermann/nextcloud-notes (commit: eb6e2b0d74e7c283d6f7921f8cee1ed4191193d7)

F.1 - Settings: Activities on locally-stored user settings and preferences. Commits related to this category are mostly about adding/removing specific items in user preferences and the integration with the Android system settings or with its Preference API.[3]

F.2 - Local database: Activities related to data management via local databases (*e.g.*, SQLite). Commit messages are usually about improving the queries to a SQLite database defined locally in the app and the addition of a local SQLite database in the app.

F.3 - File system: Activities related to the management of files in the local file system of the mobile device. Messages are usually about storing files in the SD card, cleaning up old temporary files, and checking if some locally stored configuration files exist.

G. Sensing & Communication Activities belonging to this category are related to (i) access to the device sensors (*e.g.*, camera), recording and playing media streams (*e.g.*, making a video), and (ii) communication features of the device (*e.g.*, access to the WiFi/4G networks, making calls, messaging).

> Example commit. *"Flip gray image as well, so the image is not rotated when the phone is rotated. This caused a bug when switching between front and back camera"* - Lauszus/FaceRecognitionApp (commit: 71210a870755e384d35b66e1272abd2c44480b05)

G.1 - Network: Activities related to the usage of the network. Commit messages usually include management of different levels of available bandwidth, switching to secure protocols, management of network errors, and management of TCP sockets leaks.

G.2 - Audio: Activities related to audio playback. Commit messages are usually about the management of the Android audio focus for playing sounds when in background,[4] and the management of audio playback sessions.

G.3 - Image: Activities about the management of images in the app. Commit messages in this category are usually about backup, elaboration, and download of images.

G.4 - Sensor: Activities aimed at accessing device sensors. Commit messages regard mostly the interaction with the GPS sensor, the accelerometer, and the gyroscope.

G.5 - Camera: Activities related to the usage of the device camera. Commit messages in this category are usually about taking a

[3] https://developer.android.com/guide/topics/ui/settings.html
[4] http://developer.android.com/guide/topics/media-apps/audio-focus.html

picture when using the app, when and how to show the preview of a taken picture, usage of the flash light, switching between front and rear camera.

G.6 - Messaging: Activities related to SMS/MMS messages. Commit messages in this category are usually about sending/receiving messages and developing fallbacks when SMS/MMS messages cannot be handled.

G.7 - Call: Activities related to making and receiving phone calls. Messages usually regard making a call to a specific phone number, receiving calls, and silencing calls.

G.8 - Microphone: Activities related to the usage of the device microphone. Commit messages in this category are about recording audio and controlling the microphone settings.

H. API management This category regards the activities related to the interaction of the app with external APIs. In this context, by external API we mean the software used by the app, but not owned/developed by the app developers themselves (*e.g.*, the APIs of the Android platform or REST APIs).

> Example commit. *"Upgrade to broken-out Google Play Services v8.4.0"* - zulip/zulip-android (commit: ae2992f67dfec003e11cd1073b6e1f71849fd235)

H.1 - Library: Activities related to used Android libraries. Commit messages in this category are mostly about library substitution and usage of a new library.

H.2 - Android API: Activities related to the interaction of the app with the Android platform APIs. Messages refer to code changes for supporting new Android versions, retrofitting the code for supporting older Android versions, and fallbacks for fixing bugs manifesting only when the app is running on one specific version of the Android platform.

H.3 - REST API: Activities related to the interaction with REST APIs. Commit messages in this category regard changing URLs and ports of REST endpoints, adapting to new formats of the payloads of HTTP responses produced by REST endpoints, and management of authentication as well as sessions.

H.4 - Deprecation: Activities regarding reaction to deprecation, *e.g.*, by moving to supported versions. Commit messages are about removing or changing calls to deprecated code.

I. Testing & Debugging. This category covers the activities related to logging information about the app at run-time, testing (*e.g.*, test cases implementation, tests execution), and debugging.

> Example commit. *"test: ensure tests for retrieval of all persons in local database and repository passes"* - chikecodes/Debt-Manager (commit: a4bc070540c2b2726f42a78d0afa86d13d6c333f)

I.1 - Testing: Activities related to testing. Commit messages in this category are about adding, fixing, or updating test cases, and ensuring that all tests are passing.

I.2 - Logging: Activities related to logging information at run-time and to reporting crashes. Commit messages in this category mention removing logging messages before publishing the app, adding logging statements for inspecting app behaviour at development time, logging errors in the IDE console, and integrating third-party logging as well as crash reporting libraries (*e.g.*, Crashlytics,[5] Timber[6]).

I.3 - Debugging: Activities related to the debugging of the app. Commit messages refer to finding not-yet-localized bugs, manually checking test results, and raising the need for debugging a specific feature.

Discarded commits. During our manual analysis, we identified 115 commits with non-informative commit messages, which we discarded when building the taxonomy. There are three types of discarded commits: (i) 105 commits without any informative commit message (*e.g.*, just one single character, three dots, one generic word), (ii) 9 commits with funny but non-informative commit message, and (iii) 1 commit reporting about an easter egg in the app. This low number of discarded commits (*i.e.*, 2% over the total) gives more credibility to the completeness of the proposed taxonomy.

> **Result 1:** Our taxonomy comprises 9 top layer and 49 subcategories reporting a large variety of developers' self-reported activities.

3.2 RQ2. Frequency of self-reported activities

After having categorized and described the *diversity* of activities that Android app developers report to do while evolving their apps, we now focus on determining how each of these activities is prevalent in our dataset.

Figure 3 shows the distribution of the commit messages across the categories of self-reported development activities. Each block in the figure reports the cumulative value for its corresponding top level category (*e.g.*, 1,690 commits are in the category *A - App enhancement*) and the absolute value for its subcategories (*e.g.*, of the 1,690 commits belonging to category A, 623 belong to the *A.1 - New feature* category and 581 to the *A.2 - Feature changes* category).

App enhancement is the most frequent among the high-level self-reported activities. This result can be explained by the highly dynamic ecosystem like the Google Play store, where developers are involved in very rapid release cycles [24], which are mainly driven by user ratings and reviews [14, 23, 34, 45]. In fact, the two most frequent subcategories are the development of new features (*New feature*) and their improvement (*Feature changes*). Other quite recurring types of app enhancement include the improvement of *usability* and *internationalization* of the apps. Specially the latter is a likely consequence of the global nature of the Google Play store, which imposes to take the language spoken by the app users in consideration.

Bug fixing is the second most frequent category of self-reported activities of Android developers. We conjecture that this high frequency is linked to how the app quality can have a dramatic impact on the success of an Android app [32], thus forcing developers to pay special attention to continuously correct bugs [20, 24]. Also, this frequency may be explained by Android bug reports being of high quality [5], thus easing the bug fixing process, mainly via long textual descriptions of the bug, the steps to reproduce the bug,

[5] http://fabric.io/kits/android/crashlytics
[6] http://github.com/JakeWharton/timber

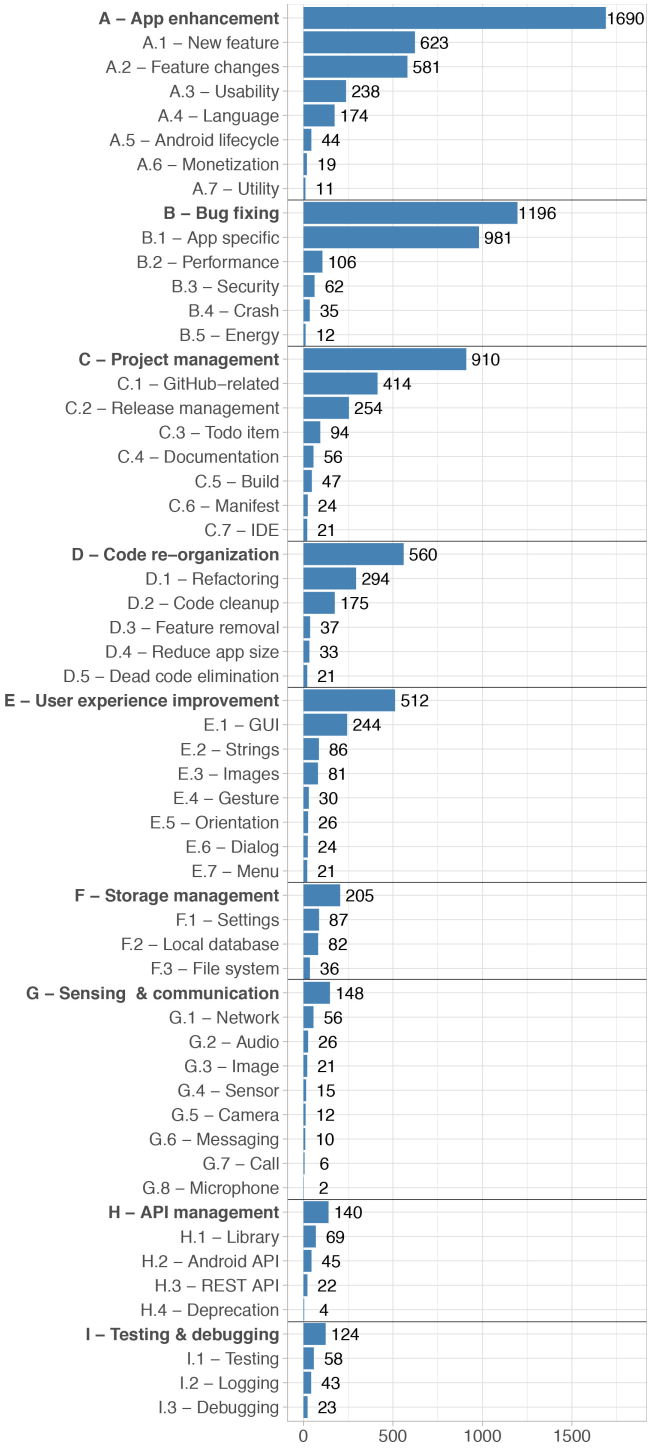

Figure 3: Frequencies of self-reported activities.

and explanations of the difference between expected and the actual outputs. In the majority of the cases, fixed bugs are about aspects *specific* to the app domain (*e.g.*, fixing the value shown in a specific card), whereas in other cases they are related to well-known key dimensions of the quality of a mobile app, such as *performance* [38], *security* [26], presence of *crashes* [12], and *energy* efficiency [11].

Project management aspects of the app covers almost a fifth of all self-reported activities. In those cases, developers are mostly referring to *GitHub-related activities* (e.g., merging a branch) or about a new *releases* of the app (e.g., changing the app version number, changing app-store-related metadata). Developers also use GitHub for leaving *todo items* for keeping note of what should be done/fixed in future iterations. Interestingly, documentation seems to be not really a prominent activity of Android developers (only 56 activities reported in our study). This result may be due to the fact that developers do not use GitHub for storing and managing the documentation of their apps, maybe in favour of more flexible, easy-to-use, and designer-friendly document sharing platforms.

Code re-organization activities are reported in 560 cases (11% on the total) by developers, with a strong predominance of *refactoring* and code *cleanup*. Those activities seem to be regarded as important by developers, despite the noticeable lack of refactoring approaches working in the context of Android applications [25]. This may also be a result of the need for quick release cycles for Android apps, where it may be the case that maintainability-related activities like code refactoring and cleanup might overlap with more functional evolutions of the app [29]. Other less-recurrent activities are feature removal, app size reduction, and dead code elimination. All of them aim at making the app more lightweight both at run-time and during the initial download of the app binary (*i.e.*, the APK file) from the Google Play store.

User experience improvement activities are almost as prevalent as *code re-organization* and this is aligned with previous research findings. In fact, past research has provided evidence that Android developers are aware of the importance of the user experience they are providing with their apps and are putting a huge emphasis on it [16]. In this area, according to our study, developers activities are mostly dedicated to the *GUI* of the app (*e.g.*, layout, animations, views), followed by a proper formatting and phrasing of textual feedback shown to the user (*i.e.*, *strings*), and the proper management of *images* (*e.g.*, images size, asynchronous loading). Other less recurrent activities are about users gestures, the the device orientation, dialogs and toasts, and (navigation) menus. The difference of the frequencies of the above described activities may be due to the fact that the GUI, strings, and images are in the vast majority of Android apps and can strongly vary across apps and projects. Differently, (i) gestures, dialogs and menus are quite standard today, both from a design and Android APIs perspectives, and (ii) the explicit management of device orientation is not widespread.

Developers **store** and manage their data locally in the app, mainly for keeping app's functionalities reliable and responsive even when the mobile device does not have a reliable connection [21]. Android *settings* and access to *local databases* (*e.g.*, SQLite) are the most recurrent subcategories, followed by access to the *file system*. This result is quite surprising since the Android settings system is based on a single class, Preference, that provides a relatively basic API to developers. Intuitively, Android developers can store settings as key-value pairs, where (i) the value of each setting can be only a primitive type (*e.g.*, boolean, integer, string) and (ii) the graphical representation of each setting is managed by the Android platform. It will be interesting to (empirically) assess how Android developers

interact with the Android settings system and why such a relatively high number of settings-related activities are performed.

Sensing & communication activities are reported in 148 cases. Among them, developers mostly interact with the *network* (e.g., by making HTTP requests, managing cached results, or managing situations where the device does not have an Internet connection). Other less common activities are related to multimedia features of mobile devices (e.g., *audio* recording, *camera*) and other *sensors* (e.g., GPS, accelerometer). Sending and receiving *messages* and making phone *calls* are in the lower part of our ranking of activities, suggesting that they are becoming less and less used by developers in favour of their Internet-based alternatives (*e.g.*, VoIP, push notifications, etc.). Surprisingly, the usage of the *microphone* is reported in only 2 cases and this is in contrast with the current trend of voice-operated apps, such as Amazon Alexa, Apple Siri, and various Google products (*e.g.*, Google Translate, Google Map).[7]

API management activities are predominated by access to third-party *libraries* and the interaction with official *Android APIs*. This results is a confirmation that using third-party libraries is a common practice for Android developers [18, 27]; moreover, it is reasonable to find a non-negligible number of commit messages referring to the interaction with the Android API, since Android apps are by their nature tightly integrated with the Android platform (*e.g.*, for managing activities' lifecycle events, accessing sensors, and showing views in the device display). The interaction with *REST APIs* is less prominent as it is the management of *deprecated* methods. The latter shows that app developers are little influenced by deprecation, similarly to developers of other systems [43, 44].

Testing and debugging are the least reported activities (only 124). Among them, testing is leading with 58 activities, followed by logging (43) and debugging (23). We suspect that those activities are so infrequent in our dataset, because developers may have embedded them into other self-reported activities (that is, when a developer implements a new feature, testing and debugging may also be present, but are not referred to). Future studies can investigate whether this is confirmed and it has an impact on developers' perception of the importance of these tasks.

Result 2: Enhancement and bug fixing operations are the most popular self-reported activities, followed by project management and code re-organization ones. Interestingly, the least reported activity is related to testing and debugging.

3.3 RQ3. Automated classification of activities

Our third research question seeks to understand to what extent it is possible to use traditional machine learning approaches to automatically classify commit messages into our taxonomy.

Table 1 reports the results achieved by the four different multi-label classification approaches we experimented. The models relying on SVM and Logistic Regression provide the best balance between precision and recall (on average, the F-Measure is 68%). This is possibly due to the use of Grid Search as technique for setting the parameters of the classifiers: as shown by recent work [9, 48], a proper configuration of these algorithms strongly improve their performance. The other classifiers, *i.e.*, Naive Bayes and Random Forest, have a lower ability to correctly classify self-reported activities. Their average F-Measure is 13 and 7 percentage points lower than SVM, respectively. Thus, in our scenario, the choice of the machine learning algorithm has an impact on the classification performance.

Considering the classification for the single categories, self-reported activities related to *Bug Fixing* are better classified by all the classifiers. A possible explanation is related to the *characteristics* words used by developers when reporting this type of activities. In fact, in the commit messages in our dataset, we often found the use of specific words like '*fix*' and '*bug*', or references to issue reports (*e.g.*, '*#19823*'), which give a strong signal that the classifiers are able to capture.

Similarly, the *Project Management* and *Enhancement* categories are classified with a similar accuracy by SVM and Logistic Regression, possibly because of the specificity of the activities performed by developers during these tasks.

Other categories have a higher variability, thus showing that there is no set of words that can be easily used as features to discriminate them. For instance, considering the cases of *Storage*, *API Management*, SVM is 14 and 26 percentage points more effective than Logistic Regression, respectively. At the same time, in the classification of *UI* activities, Logistic Regression has an F-Measure 19 percentage points higher than SVM. This indicates that for some particular categories the underlying classification algorithm makes some difference and allows an improved categorization of self-reported activities. As part of our future research agenda, we aim at further investigating how the classifiers can be used as an ensemble to improve the results [35] (*e.g.*, by means of a dynamic switching based on the characteristics of the commit messages [10]).

Finally, the investigated classifiers are not able to identify any of the commit messages related to *Sensing & Communication*. We further looked at the prediction results to investigate the reasons behind this result: we found that the misclassification is mainly due to the overlap between the terms used in *Sensing & Communication* and *Enhancement*. In other words, discriminating the two categories represents an arduous task for a machine learning algorithm since it cannot properly learn the words characterizing the two types of self-reported activities. This final result highlights a limitation of our approach. The machine learning algorithm is based on the implicit assumption that commit messages are representative of the action performed by developers, because as humans we have been able to classify them. However, a human analysis—as the one conducted in **RQ1**—may often be able to correctly characterize commits because of external factors that are often implicit (*e.g.*, experience or information contextualization [13]); these external implicit factors are not available to the machine learning approach, hence it may fail in cases where the overlap between terms in two categories is high [1, 33, 34, 37].

[7] http://info.localytics.com/blog/voice-activated-apps-are-changing- everything.-heres-how

Table 1: Performance of the Experimented Machine Learning Approaches when combined with OneVsRest

Category	SVM			Logistic Regression			Naive Bayes			Random Forest		
	Precision	Recall	F-M	Precision	Recall	F-M	Precision	Recall	F-M	Precision	Recall	F-M
Project Management	81%	76%	78%	80%	72%	76%	87%	64%	74%	85%	72%	78%
Storage	71%	50%	59%	50%	42%	45%	67%	25%	36%	86%	38%	52%
UI	41%	35%	38%	39%	54%	46%	42%	15%	22%	61%	20%	30%
Debug	58%	54%	56%	55%	61%	58%	71%	18%	29%	67%	7%	13%
Code Re-Organization	70%	69%	69%	68%	68%	68%	75%	47%	58%	79%	50%	61%
Bug Fixing	87%	74%	80%	87%	73%	79%	71%	47%	57%	87%	71%	78%
Enhancement	70%	66%	68%	66%	77%	71%	65%	52%	58%	72%	54%	62%
API Management	72%	74%	73%	44%	52%	48%	55%	19%	29%	33%	3%	6%
Sensing & Communication	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Average	**71%**	**66%**	**68%**	**69%**	**68%**	**68%**	**69%**	**46%**	**55%**	**75%**	**55%**	**61%**

Result 3: While for categories like *Bug Fixing* and *Project Management* the classification performance is up to 80%, machine learning approaches can classify developers' self-reported activities with an average F-Measure of 68%. However, our analysis revealed some possible points of improvement (*e.g.*, exploiting the complementarity among classifiers).

4 RELATED WORK

The analysis of self-reported activities represents one of the most valid ways to comprehend and analyze the development process of a software system [22]. Despite this, so far self-reported developers' activities have been investigated only by targeting different type of systems, *e.g.*, generic open-source software [39], or by focusing on specific aspect of Android apps, such as performance [8] and energy consumption [3, 28].

Ray *et al.* [39] analyzed a large dataset from GitHub to understand the effect of programming languages on software quality. Analyzing different aspects, including commit messages, and controlling confounding effects, they found that language design has modest effect on software quality.

Das *et al.* [8] investigate to what extent developers take care of performance issue analyzing commit messages. The analysis, conducted on 2,443 open source Android apps, showed that most of the commits that lead to performance issue are related to GUI, code smell fixing, network related code, and memory management. Moura *et al.* [28] conducted a study similar to ours on an initial sample of 2,189 commits from the Github repository to analyze energy-aware commits. Analyzing a final dataset of 371 commits from 317 real world apps, they found that software developers heavily rely on low-level energy management approaches, such as frequency scaling and multiple levels of idleness. Moreover, energy saving techniques can impact the application correctness. With the same aim, Bao *et al.* [3] extended Moura *et al.* by analyzing 468 commits from 154 Android apps. They discovered six power management activities for Android apps and discovered that power management activities vary with respect to the Android store category of the app. With respect to these works, we analyzed commits related not only on performance issues and energy management.

5 CONCLUSIONS AND IMPLICATIONS

Our work aimed at understanding and classifying self-reported activities of Android developers. Analyzing 5,000 commit messages from 8,280 apps, we defined a taxonomy of self-reporting activities, studied their frequency, and investigated the feasibility of an automated approach for categorizing them.

Our results showed that changes applied by developers are mostly related to enhancement or bug fixing operations: these categories are clearly the ones for which more automatic support would be needed. A very few commits are instead related to the management of APIs and testing, possibly highlighting the lack of specific tools supporting developers during these operations. Finally, a machine learning approach can correctly classify self-reported activities with an average F-Measure of 68%.

Our findings have a number of **implications** for both Android developers and researchers. *Android developers* can use our taxonomy of development activities for taking more informed decisions when assigning code reviews to team members. For example, commits related to the Rest API category may be assigned to those members who are also involved in the development of the back-end of the mobile app (who potentially are more knowledgeable of the interaction between the app and its back-end). Also, categorized commits can be used (i) for getting a clear idea about which activities are being performed by developers during the whole project lifecycle, (ii) for identifying potential blocking activities where developers are spending the majority of their working time, or (iii) as decision support system when allocating resources to the project. Finally, developers can use our classifier for automatically categorizing code commits according to our taxonomy of activities.

We support *researchers* by increasing our empirical understanding of the types of (self-reported) activities performed by Android developers in real projects. This is a key step to guide future research in the area. Specifically, the most recurrent activities may be a good indicator for future research on Android apps development. Moreover, both the taxonomy and our automatic classifier have the potential to strengthen the reliability of other mining approaches that use commit messages as input (e.g., [3, 7, 8, 28, 32]). It is our hope that our results and the shared dataset will help and guide future research on support the engineering of Android apps.

ACKNOWLEDGMENT

Bacchelli and Palomba gratefully acknowledge the support of the Swiss National Science Foundation through the SNF Project No. PP00P2_170529.

REFERENCES

[1] Alberto Bacchelli, Tommaso Dal Sasso, Marco D'Ambros, and Michele Lanza. 2012. Content classification of development emails. In *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 375–385.

[2] R. Baeza-Yates and B. Ribeiro-Neto. 1999. *Modern Information Retrieval*. Addison-Wesley.

[3] Lingfeng Bao, David Lo, Xin Xia, Xinyu Wang, and Cong Tian. 2016. How Android App Developers Manage Power Consumption?-An Empirical Study by Mining Power Management Commits. In *Mining Software Repositories (MSR), 2016 IEEE/ACM 13th Working Conference on*. IEEE, 37–48.

[4] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.

[5] Pamela Bhattacharya, Liudmila Ulanova, Iulian Neamtiu, and Sai Charan Koduru. 2013. An empirical analysis of bug reports and bug fixing in open source android apps. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*. IEEE, 133–143.

[6] Gobinda G Chowdhury. 2003. Natural language processing. *Annual review of information science and technology* 37, 1 (2003), 51–89.

[7] Luis Fernando Cortés-Coy, Mario Linares-Vásquez, Jairo Aponte, and Denys Poshyvanyk. 2014. On automatically generating commit messages via summarization of source code changes. In *Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on*. IEEE, 275–284.

[8] 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.

[9] Sergio Di Martino, Filomena Ferrucci, Carmine Gravino, and Federica Sarro. 2011. A genetic algorithm to configure support vector machines for predicting fault-prone components. In *International Conference on Product Focused Software Process Improvement*. Springer, 247–261.

[10] Dario Di Nucci, Fabio Palomba, Rocco Oliveto, and Andrea De Lucia. 2017. Dynamic Selection of Classifiers in Bug Prediction: An Adaptive Method. *IEEE Transactions on Emerging Topics in Computational Intelligence* 1, 3 (2017), 202–212.

[11] 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.

[12] Giovanni Grano, Adelina Ciurumelea, Sebastiano Panichella, Fabio Palomba, and Harald C. Gall. 2018. Exploring the Integration of User Feedback in Automated Testing of Android Applications. In *Proceedings of the 25th International Conference on Software Analysis, Evolution, and Reengineering*. IEEE Press, to appear.

[13] John J Gumperz. 1992. Contextualization and understanding. *Rethinking context: Language as an interactive phenomenon* 11 (1992), 229–252.

[14] Mark Harman, Yue Jia, and Yuanyuan Zhang. 2012. App store mining and analysis: MSR for app stores. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*. IEEE, 108–111.

[15] Jun Huang, Guorong Li, Qingming Huang, and Xindong Wu. 2015. Learning label specific features for multi-label classification. In *Data Mining (ICDM), 2015 IEEE International Conference on*. IEEE, 181–190.

[16] 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.

[17] 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.

[18] Li Li, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. An investigation into the use of common libraries in android apps. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, Vol. 1. IEEE, 403–414.

[19] William Lidwell, Kritina Holden, and Jill Butler. 2010. *Universal Principles of Design, Revised and Updated: 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach through Design* (2nd ed.). Rockport Publishers.

[20] Dayi Lin, Cor-Paul Bezemer, and Ahmed E Hassan. 2017. Studying the urgent updates of popular games on the steam platform. *Empirical Software Engineering* 22, 4 (2017), 2095–2126.

[21] Yingjun Lyu, Jiaping Gui, Mian Wan, and William GJ Halfond. 2017. An Empirical Study of Local Database Usage in Android Applications. In *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*. IEEE, 444–455.

[22] Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke. 2014. On the comprehension of program comprehension. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 23, 4 (2014), 31.

[23] William Martin, Federica Sarro, Yue Jia, Yuanyuan Zhang, and Mark Harman. 2017. A survey of app store analysis for software engineering. *IEEE Transactions on Software Engineering* 43, 9 (2017), 817–847.

[24] Stuart McIlroy, Nasir Ali, and Ahmed E Hassan. 2016. Fresh apps: an empirical study of frequently-updated mobile apps in the Google play store. *Empirical Software Engineering* 21, 3 (2016), 1346–1370.

[25] Tom Mens and Tom Tourwé. 2004. A survey of software refactoring. *IEEE Transactions on software engineering* 30, 2 (2004), 126–139.

[26] Francesco Mercaldo, Corrado Aaron Visaggio, Gerardo Canfora, and Aniello Cimitile. 2016. Mobile malware detection in the real world. In *Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on*. IEEE, 744–746.

[27] Israel J Mojica, Bram Adams, Meiyappan Nagappan, Steffen Dienst, Thorsten Berger, and Ahmed E Hassan. 2014. A large-scale empirical study on software reuse in mobile apps. *IEEE software* 31, 2 (2014), 78–86.

[28] Irineu Moura, Gustavo Pinto, Felipe Ebert, and Fernando Castor. 2015. Mining energy-aware commits. In *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 56–67.

[29] Meiyappan Nagappan and Emad Shihab. 2016. Future trends in software engineering research for mobile apps. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, Vol. 5. IEEE, 21–32.

[30] Nasser M Nasrabadi. 2007. Pattern recognition and machine learning. *Journal of electronic imaging* 16, 4 (2007), 049901.

[31] Robert M Obrien. 2007. A caution regarding rules of thumb for variance inflation factors. *Quality & Quantity* 41, 5 (2007), 673–690.

[32] Fabio Palomba, Mario Linares-Vásquez, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. 2018. Crowdsourcing user reviews to support the evolution of mobile apps. *Journal of Systems and Software* 137 (2018), 143–162.

[33] Fabio Palomba, Annibale Panichella, Andrea De Lucia, Rocco Oliveto, and Andy Zaidman. 2016. A textual-based technique for smell detection. In *Program Comprehension (ICPC), 2016 IEEE 24th International Conference on*. IEEE, 1–10.

[34] Fabio Palomba, Pasquale Salza, Adelina Ciurumelea, Sebastiano Panichella, Harald Gall, Filomena Ferrucci, and Andrea De Lucia. 2017. Recommending and localizing change requests for mobile apps based on user reviews. In *Proceedings of the 39th International Conference on Software Engineering*. IEEE Press, 106–117.

[35] Annibale Panichella, Rocco Oliveto, and Andrea De Lucia. 2014. Cross-project defect prediction models: L'union fait la force. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*. IEEE, 164–173.

[36] Maja Pantic, Alex Pentland, Anton Nijholt, and Thomas S Huang. 2007. Human computing and machine understanding of human behavior: a survey. In *Artifical Intelligence for Human Computing*. Springer, 47–71.

[37] Luca Pascarella and Alberto Bacchelli. 2017. Classifying code comments in Java open-source software systems. In *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 227–237.

[38] Lenin Ravindranath, Jitendra Padhye, Sharad Agarwal, Ratul Mahajan, Ian Obermiller, and Shahin Shayandeh. 2012. AppInsight: Mobile App Performance Monitoring in the Wild.. In *OSDI*, Vol. 12. 107–120.

[39] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2014. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 155–165.

[40] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. 2009. Classifier chains for multi-label classification. *Machine Learning and Knowledge Discovery in Databases* (2009), 254–269.

[41] 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.

[42] Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management* 24, 5 (1988), 513–523.

[43] Anand Sawant, Romain Robbes, and Alberto Bacchelli. 2017. On the reaction to deprecation of clients of 4+1 popular Java APIs and the JDK. *Empirical Software Engineering* (2017), online first.

[44] Anand Ashok Sawant, Mauricio Aniche, Arie van Deursen, and Alberto Bacchelli. 2018. Understanding Developers' Needs on Deprecation as a Language Feature. In *Proceedings of the 40th ACM/IEEE International Conference on Software Engineering (ICSE 2018)*. forthcoming.

[45] Simone Scalabrino, Gabriele Bavota, Barbara Russo, Rocco Oliveto, and Massimiliano Di Penta. 2017. Listening to the Crowd for the Release Planning of Mobile Apps. *IEEE Transactions on Software Engineering* (2017).

[46] Forrest Shull, Janice Singer, and Dag IK Sjøberg. 2008. *Guide to advanced empirical software engineering*. Vol. 93. Springer.

[47] Mervyn Stone. 1974. Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society. Series B (Methodological)* (1974), 111–147.

[48] Stephen W Thomas, Meiyappan Nagappan, Dorothea Blostein, and Ahmed E Hassan. 2013. The impact of classifier configuration and classifier combination on bug localization. *IEEE Transactions on Software Engineering* 39, 10 (2013), 1427–1443.

[49] Yuan Tian, Julia Lawall, and David Lo. 2012. Identifying linux bug fixing patches. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 386–396.
[50] Grigorios Tsoumakas and Ioannis Katakis. 2006. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining* 3, 3 (2006).
[51] 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.
[52] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. 2012. *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers.