Analyzing Reviews and Code of Mobile Apps for Better Release Planning

Adelina Ciurumelea, Andreas Schaufelbühl,
Sebastiano Panichella, Harald C. Gall
University of Zurich,
Department of Informatics, Switzerland
ciurumelea@ifi.uzh.ch, andreas.schaufelbuehl@uzh.ch,
{panichella,gall}@ifi.uzh.ch

***Abstract*—The mobile applications industry experiences an unprecedented high growth, developers working in this context face a fierce competition in acquiring and retaining users. They have to quickly implement new features and fix bugs, or risks losing their users to the competition. To achieve this goal they must closely monitor and analyze the user feedback they receive in form of reviews. However, successful apps can receive up to several thousands of reviews per day, manually analysing each of them is a time consuming task. To help developers deal with the large amount of available data, we manually analyzed the text of 1566 user reviews and defined a high and low level taxonomy containing mobile specific categories (*e.g.* performance, resources, battery, memory, etc.) highly relevant for developers during the planning of maintenance and evolution activities. Then we built the User Request Referencer (URR) prototype, using Machine Learning and Information Retrieval techniques, to automatically classify reviews according to our taxonomy and recommend for a particular review what are the source code files that need to be modified to handle the issue described in the user review. We evaluated our approach through an empirical study involving the reviews and code of 39 mobile applications. Our results show a high precision and recall of URR in organising reviews according to the defined taxonomy.**

***Index Terms*—Mobile Applications, User Reviews, Text Classification, Code Localization**

I. Introduction

Google Play and the Apple App Store are the leading distribution platforms for mobile applications, each has more than 2 millions apps [1] and enable the download of hundreds of millions of apps every day [2]. The ease of distribution and the large number of potential users, has made mobile applications development an attractive field for software engineers, but has also led to an increasing competition between developers [3] [4]. To stay ahead of their competition, app developers need to continuously monitor and respond to the needs of their users.

One interesting feature of market places, like Google Play and the App Store, is that they do not only facilitate the distribution of mobile applications, but also allow users to easily rate and post reviews for the apps they download. These reviews represent a rich source of information for app developers to understand the needs and desires of their users. While reviews contain valuable feedback directly from the app's users, the amount of unstructured text information they include can be overwhelming for developers. Extremely popular apps, like Facebook, receive thousands of reviews per day [5]. Open source apps might have less reviews, but they also have limited resources and analysing even hundreds of reviews will result in sacrificing valuable development time that could be spent fixing bugs and implementing new features.

There is a clear need for tools that facilitate the automated analysis of user reviews and the research community has proposed several approaches for achieving this goal. Previous work has observed that users often report bugs and feature requests in reviews [6] [5], summarize the user experience for specific features [7], request enhancements [8] and new features [5] [9] and include comparisons with other apps. Nevertheless reviews present specific challenges, additionally to their high frequency, they consist of unstructured text with a low descriptive quality, as they are written by users without any technical knowledge, therefore they are often difficult to automatically parse and analyze [10], [11].

Several approaches have been investigated for the automated classification of user reviews. Chen *et al.* [11] developed a tool to classify reviews as either informative or non-informative, while observing that only a third of reviews are informative. Other researchers proposed different methods for selecting useful reviews for maintenance activities [8], [10], [12], [13], [14], [15], [16], [7], [17], [18], [19], [20], [21] using techniques based on topic modelling, sentiment analysis and natural language processing to classify reviews according to a limited set of classes (*e.g.* bugs, feature requests). Recently, Villarroel *et al.* [15] developed a tool that additionally to the automated classification is able to cluster and prioritize reviews. Most approaches classify reviews according to a very limited set of classes and then cluster them based on textual similarity, this results in a list of unstructured review clusters that still need to be manually analysed to understand what topics they discuss. We believe that such set of unstructured review clusters is of limited use for developers trying to distill actionable change tasks from the user feedback.

In our work we investigated what are the specific fine grained topics that users address in reviews and are relevant for developers while planning maintenance and evolution tasks for their applications. To achieve this goal we manually analysed 1566 user reviews and built a multi-level taxonomy, oriented on mobile specific software maintenance and evolution tasks, which allows developers to analyse the user review content at a high and low level granularity. For example using our taxonomy, the following review from our dataset: *"For info (in case dev not already aware!), there is a graphical glitch when scrolling output in marshmallow on a nexus 5."*, will be assigned to the *Usage* and *Compatibility* high level categories and for a finer grained analysis to the *UI*, *Device* and *Android version* low level categories.

Starting from the taxonomy we built **URR** (**U**ser **R**equest **R**eference), a prototype that is able to group reviews according to the high and low level categories of our taxonomy. Using our approach, a developer will be able to analyse either reviews belonging to a very specific category, or all reviews grouped per single or multiple categories. This is in contrast with previous approaches that return unstructured clusters of reviews and leave it up to the developer to understand what specific topics they discuss. Furthermore we customized traditional IR-based technique for source code localization, by considering information related to the specific structure of mobile software projects, to recommend the source code files associated with a specific review analyzed by **URR**.

Thus, the main contributions of our work are the following:

1) A multi-level taxonomy for user reviews targeted towards mobile specific issues that developers address during the maintenance and evolution activities of their apps;
2) The **URR** prototype that is able to (i) organize reviews according to the defined taxonomy and (ii) recommend the source code files that are likely to be modified to handle the mobile specific issues and requests highlighted by the users;
3) A thorough evaluation of each step of our approach using the user reviews and code of 39 mobile applications. Specifically, we assess the accuracy of **URR** in (i) classifying user reviews according to the taxonomy (ii) and recommending related source files; with the help of two external evaluators (from both academia and industry) with a mobile development background;
4) We make available a replication package[1] [22] including (i) the datasets, (ii) and the raw data and results of our evaluation study.

The rest of the paper is organized as follows: in Section II we described the details of our approach; Section III and IV describe the evaluation and the results; in Section V we discuss threats to validity; Section VI discusses related work and Section VII presents our conclusion.

II. Approach

A. Approach Overview

The goal of our work is automating the analysis of user reviews according to relevant mobile specific and actionable issues and linking them to the source code files that should be modified to handle the problems or requests discussed in a review. We want to help developers spend less time manually analysing reviews and more time improving their app according to the user needs. To this end we developed our approach, the **U**ser **R**equest **R**eferencer (**URR**) using the following steps:

1) ***Taxonomy Definition***: we built a fine grained multi-level taxonomy for mobile specific issues that user discuss in reviews;
2) ***User Reviews Classification***: we developed the **URR** prototype to automatically classify reviews according to the taxonomy from the previous step;
3) ***Source Code Localization***: we extended the **URR** prototype to automatically link classified reviews to the related source code files (the ones that have to be modified to address the issues expressed in a review).

B. A Motivating Example

AcDisplay[2] is an Android app available on Google Play that belongs to the category personalisation. It allows Android users to personalise the way they receive notifications and also acts as a lock screen. It has an average rating of 4.2 stars and over 60000 reviews and is listed with downloads between 1 million and 5 million (Google Play only shows the range of values). The source code is available on GitHub and the project has a single main contributor. AcDisplay represents one of the apps analyzed in our work, in particular, we crawled the reviews belonging to the latest version (3.8.4), gathering a total of 752 user reviews. Although the number of reviews for this version is not exceptionally large, the developer is likely working on the project in their spare time and has limited time resources that could either be spent by manually analysing reviews and keeping track of how many users complain about the same issues or use an automated approach that is able to directly classify and group reviews according to fine grained mobile specific issues. Then the developer can use the remaining time to actually fix or implement the issues highlighted by users.

For example, let us consider the situation in which the developer is interested in investigating whether users talk about compatibility (e.g., mention the specific mobile device, Android version, etc.) in their reviews. The developer can ask **URR** to return all reviews belonging to the high level categories named *Complaint* and *Compatibility* (more details are included in Section II-C). Our prototype will return 61 reviews associated with the categories. After reading the first couple of reviews, the developer decides to further investigates compatibility issues with the Operating System, therefore asks **URR** to return all reviews belonging to the *Complaint* and *Android Version* categories. This time **URR** returns 22 reviews, we list some of them next:

> *"Good but has some issues with Marshmallow I used this on my old phone and if was flawless and I loved it. I noticed that sometimes when I had AcDisplay activated I would not be able to use the fingerprint sensor even after I unlocked*

[1]http://bit.ly/2ekaM5f

[2]https://play.google.com/store/apps/details?id=com.achep.acdisplay

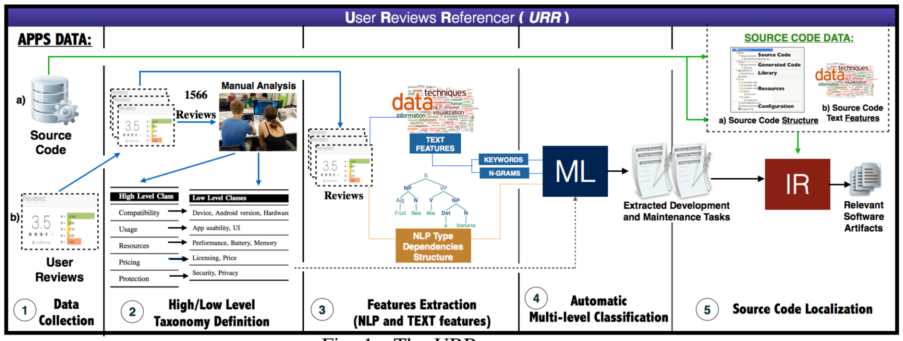
User Reviews Referencer (URR)
APPS DATA:
a) Source Code
b) User Reviews
1566 Reviews
Manual Analysis
High Level Class
Low Level Classes
Compatibility
Device, Android version, Hardware
Usage
App usability, UI
Resources
Performance, Battery, Memory
Pricing
Licensing, Price
Protection
Security, Privacy
Reviews
TEXT FEATURES
KEYWORDS
N-GRAMS
NLP Type Dependencies Structure
ML
Extracted Development and Maintenance Tasks
SOURCE CODE DATA:
a) Source Code Structure
b) Source Code Text Features
IR
Relevant Software Artifacts
1 Data Collection
2 High/Low Level Taxonomy Definition
3 Features Extraction (NLP and TEXT features)
4 Automatic Multi-level Classification
5 Source Code Localization

Fig. 1. The URR process

TABLE I
HIGH LEVEL TAXONOMY

Category	Description
Compatibility	mentions the version of the OS, the specific phone device or a phone hardware component (*SD card, fingerprint sensor*).
Usage	talks about an element of the UI or the ease or difficulty in using the app.
Resources	discusses the memory, battery usage or the performance of the app
Pricing	statements mentioning the licensing model or price of the app.
Protection	statements referencing security issues or user data privacy.
Complaint	the user reports or complains about an issue with the app.

AcDisplay and had to enter a password. This is very frustrating so I cannot use AcDisplay."

"Love the design I love the app. It's super sleek and nice. But ever since my phone updated to marshmallow it's stopped working. Hope it comes back soon."

"On Marshmallow, the screen is buggy and sometimes shows the notification shade."

The developer quickly realises that users report problems with Marshmallow and decides to plan a task for testing the application on this Android version. Manually analysing over 700 reviews would have taken much more of their time.

C. Taxonomy Definition

User reviews for mobile apps tend to discuss a limited set of topics, as observed by Pagano *et al.* in [5]. The goal of this step is defining a set of review categories that reflect mobile apps specific and actionable issues at a finer level of granularity than previous work has done. To develop the taxonomy we performed an iterative content analysis technique, similar to the one described in [23], on a subset of reviews selected from the dataset described in section III-B. Our complete dataset contains only reviews for the latest version, therefore some apps will have less than 200 reviews. To select the subset for the manual analysis we randomly sampled 200 reviews for the apps that had more reviews and selected all the available reviews for the ones with less than 200 reviews, to ensure a heterogeneous set. This resulted in a subset consisting of 1566 reviews from 39 mobile applications. Next we describe how we built a two level taxonomy that enables developers to analyse reviews at both a coarser and finer grained granularity.

1) High Level Taxonomy: Previous work classified reviews as informative and non-informative [11] or as bug report, suggestion for new feature and other [10], [12], [13], [14], [15], [16], [7], [17]. However, we decided to observe in more detail what users talk about in their reviews. One of the authors of the paper manually analysed the content of the 1566 user reviews in two iterations. First they started with an empty list of categories and as they read each review added a new category to the list, if the review belonged to a new category, and then assigned one or more categories to the review according to the list collected so far. At the end of this step overlapping categories have been merged and another iteration was performed on the reviews to assign to each one the set of corresponding categories from the final list. Each time a new category was added to the list, the author also provided a short definition and a list of specific keywords. The final list of categories for the high level taxonomy is included in Table I.

During this step we noticed that users often talk about a specific category in either positive or negative terms, *e.g.* they either say that the app *drains the battery* or is *light on the battery*. We believe that from a developer point of view, both aspects are relevant, as they either report a problem that needs to be fixed or that the users are happy with a certain feature of the app and the developer does not have to invest more time in improving it. Therefore the first 5 categories from Table I include reviews discussing both positive and negative aspects of the specific category and the last one (*Complaint*) can restrict the analysis to only problems (or complaints) faced by users.

An important observation from our analysis, is that a specific user review often discuss several of the topics we listed in Table I. For example, in the following review extracted from our dataset:

"It just shuts down without any warning! I use Android Lollipop and every time I open up anything it would just close down instantly. If they fix this problem I would probably pay for the premium version."

the user reports a problem with the app (*Complaint*), mentions the Android version (*Compatibility*) and talks about paying for the premium version (*Pricing*) if the issue is addressed. Therefore an important requirement for helping developers use reviews to better organize their planning and maintenance activities, is being able to automatically perform multi-label classification on reviews. We provide further details on how this was achieved in section II-D.

TABLE II
COMPLETE TAXONOMY

HLC	LLC	Description
Compatibility	**Device**	mentions a specific mobile phone device (*i.e. Galaxy 6*).
	Android Version	references the OS version. (*i.e. Marshmallow*).
	Hardware	talks about a specific hardware component.
Usage	**App Usability**	talks about ease or difficulty in using a feature.
	UI	mentions an UI element (*i.e. button, menu item*).
Resources	**Performance**	talks about the performance of the app (*slow, fast*).
	Battery	references related to the battery (*i.e. drains battery*).
	Memory	mentions issues related to the memory (*i.e. out of memory*).
Pricing	**Licensing**	references the licensing model of the app (*i.e. free, pro version*)
	Price	talks about money aspects (*i.e. donated 5$*).
Protection	**Security**	talks about security/lack of it.
	Privacy	issues related to permissions and user data.

2) Low Level Taxonomy: Although the set of high level categories is more detailed than previous work [10], [12], [13], [14], [15], [16], [7], [17], [11], we decided to refine it further in a lower level taxonomy (using the same methodology adopted for defining the high-level taxonomy). The refined list is included in Table II. Previous papers often cluster reviews according to textual similarity [15], this results in an unstructured set of review clusters which share similar terms. Thus, it is up to the developer to read and understand what each cluster is talking about. By classifying reviews according to fine grained mobile specific categories, developers are able to restrict their analysis to very precise issues. These categories are actionable as they give the developer a clear indication of what particular aspect of the app needs to be addressed during their maintenance and evolution activities.

D. *U*ser *R*equest *R*eferencer (*URR*)

We have developed **URR**, a tool prototype to automatically classify reviews according to the taxonomy defined in the previous sections and link them to the source code. Next we describe in detail the implementation of our prototype:

1) Feature Extraction: To automatically classify reviews using Machine Learning (ML) techniques, we first had to extract a set of features from the review text. We initially went through the standard steps for text preprocessing: removing stop words belonging to the English stop words list and punctuation and applying the Porter Stemmer [24] to reduce words to their root form. Next we describe the features extracted from the preprocessed text:

- **Term frequency based features**: as a first set of features for training the ML classifier, we considered the *tf-idf* scores associated with each term present in a given review. The *tf-idf* term weighting scheme assigns a higher score to rare words and a lower one to words occurring frequently across all reviews. More formally, the *tf-idf* score [25] is defined according to the formula:

$$\text{tf-idf}_{t,d} = tf_{t,d} \times idf_t = tf_{t,d} \times log(\frac{N}{df_t})$$

where $tf_{t,d}$ is the raw frequency (number of occurrences) of word t in review d while, idf_t is the inverse document frequency (called also term specificity) of terms occurring in various reviews.
- **N-grams based features**: additionally to the *tf-idf*, we extract as text features the 2-grams and 3-grams of terms in each review to capture groups of words that are representative for a specific taxonomy category, this step was applied on the preprocessed text. For example, for the review *"Always ran out of memory."*, we extracted the following 2-grams: *Always ran*, *ran out*, *out of*, *of memory* and 3-grams: *Always ran out*, *ran out of*, *out of memory*. Besides capturing expressions or groups of words that are used together, n-grams are able to capture negations in review sentences.

Although for each review we stored the author, date, star rating and the review text, we only used the previously mentioned features for the training. We are aware that the star rating might be a very good prediction for the *Complaints* class, but we decided to only focus on the review text in this version of our work.

2) Training Machine Learning Classifiers: **URR** uses the Gradient Boosted Regression Trees (GBRT) model for classifying reviews implemented in the Graphlab library [26]. Although we have experimented with several different models, we have chosen this one as it has returned the best results and is reported in the literature as one of the most effective machine learning models for predictive analytics. Specifically, GBRT makes predictions by combining decisions from a sequence of decisions trees. During the training process at each iteration a new tree is built for maximizing the correctness of classification on the training data. While building the tree models, the feature that best partitions the data is selected at each node. An interesting detail of the model is that after adding a tree, it evaluates the accuracy on the training set and gives a weight to each misclassified training example. In this way the next tree will try harder to correctly classify those.

The high and low level taxonomy contains a large number of categories (17 + 1 for *Complaint*) and as we already mentioned single reviews often address multiple topics. Therefore one of the requirements of the **URR** prototype was being able to automatically perform multi-label classification, that is given a review return the list of high or low level categories it belongs to. A typical method for solving this kind of problem is called the one-vs-all strategy, where a separate classifier is trained for each class (or category). Therefore for building our prototype we trained a separate classifier for each category from the high and low taxonomy. During this step we use the entire set of 7754 reviews as described in Section III-B, but before training each classifier we first split the reviews set using a stratified sampling technique into an 80% train set and a 20% test set. The reviews from the test set are later used during the evaluation. We would like to note that the splitting into the train and test set is done for each category, and each classifier is trained and evaluated independently on a different split of the same data set of 7754 reviews.

3) Classifying Reviews according to the Taxonomy: Each of the classifiers trained previously will output for a given review whether it belongs to a category or not. For example, in order to classify reviews according to the high level taxonomy, URR will return the list of categories for which it received a positive answer. It will perform a similar process for the low

TABLE III
STRUCTURE CATEGORY RULES AND
MATCHING HIGH LEVEL CATEGORIES (HLC).

Structure Category	Identification Rules	HLC
UI	filepath contains 'res', 'resources', 'ui' or 'Activity'	Usage (only UI)
Android Manifest	filepath contains 'AndroidManifest'	Compatibility Protection (only Privacy)
Content Provider	file text contains 'content', 'provider' or 'Content-Provider'	Resources
Service	file path contains 'Service'	No matched category

level taxonomy. At the end, each review will be labelled with a list of categories it belongs to from the high and low level taxonomy.

4) Source Code Localization: We extended the **URR** prototype to find, for a given review, what are the source code files that likely need to be modified in order to handle the issue mentioned in a review. This recommendation is important for developers while planning their maintenance activities, as it enables them to estimate the impact of a change and locate it. The implementation of the source code localization uses classical IR-based methods from the Apache Lucene API (version 5.5). Our implementation for linking the reviews of a single app to the source code consists of the following steps:

- ***Data Collection***: For each app, **URR** first downloads the source code and selects the corresponding reviews from the database.
- **Preprocessing**: The source code is preprocessed using camel case splitting, as it is a typical pattern for naming variables, methods and classes in Java. Then for both source code and reviews **URR** applies stop word removal and stemming (Porter Stemmer [27]) to reduce noise and convert words to their root form. We extended the standard English stop words list to include Java identifiers (*e.g.* public, void, String, etc.).
- ***Indexing the Source Code and Reviews***: **URR** then indexes the Java and XML files of the app and the reviews with Lucene using the *tf-idf* term weighting score [25].
- **Pre-Localization**: Reviews are usually written by non-technical users without any knowledge of the implementation details of the app, for this reason our problem is different than more traditional source code linking approaches based on IR. However, Android software projects often have a common/standard structure (or packages), as identified by Hu *et al.* [28], and we are able to take advantage of that to improve the search for a given review using the assigned categories from the high level taxonomy. Based on the work of Hu *et al.* [28] we define a set of structure categories and the corresponding rules for identifying source files. Each structure category is then matched with a category from our high level taxonomy as described in Table III. We were not able to match all categories of the high and low level taxonomy and all structure categories. Nevertheless, when we are able to define a match we use this extra information to assign to each document (both reviews and source code files) in the created index an additional search parameter called *structure category*. In this way each document has two search parameters: (i) the *text* which corresponds to the preprocessed text content and (ii) *structure category* which corresponds to one of the categories reported in Table III. This information is then used in the following step.
- ***Searching by Relevance***: After finishing the indexing, **URR** performs a search using the Lucene API and returns the top scoring source files. Lucene uses the Vector Space Model (VSM) as an Information Retrieval (IR) model to compute the textual similarity between user reviews and the source code. During the search **URR** integrates the information related to the structure category by adding a boosting score. Specifically, when a Review$_i$ and an Artifact$_j$ have a matching taxonomy category, then we use the boosting functionality of Lucene to increase the similarity score between them by a given *bonus score* percentage. A preliminary analysis returned the best results for a 30% score, but we plan to do a more in depth analysis during future work to find the best value.

III. EVALUATION

A. Research Questions

We conducted a study to evaluate the accuracy of URR in (i) classifying reviews according to the taxonomy defined in section II-C and (ii) in recommending the source code artifacts that need to be modified to address issues referenced in user reviews. We designed our study towards the following research questions:

1) *RQ$_1$: To what extent does the **U**ser **R**equest **R**eferencer organize reviews according to meaningful maintenance and evolution tasks for developers?*
2) *RQ$_2$: Does the **U**ser **R**equest **R**eferencer correctly recommend the software artifacts that need to be modified in order to handle user requests and complaints?*

We evaluated the results returned by **URR** through an empirical study using 7754 user reviews from 39 mobile applications. We involved two external evaluators with mobile applications development background to provide an objective quantitative and qualitative evaluation of our approach.

B. Dataset

Our dataset consists of 39 open source apps from the Google Play Store, also available on the F-Droid [29] repository. These apps were selected to cover 17 categories and different sizes. This ensured a variety of users and reviews and consequently a diverse vocabulary which was needed to increase the generalizability of our approach.

We crawled reviews belonging to the latest version of the apps from the Google Play Store during the period June - July 2016. Google Play only allows selecting reviews for the latest or all versions of an app. Because we later link reviews to the source code, we needed to make sure that we could match them to the correct version of an app, therefore we restricted the crawling process to reviews belonging to the latest version.

TABLE IV
OVERVIEW OF THE DATASET

Category	Apps	LOC	Classes	Reviews
Internet	2	23893	92	234
Reading	1	20233	149	507
Travel & Local	1	88272	764	39
Theming	2	12983	60	775
Games	3	55263	530	770
System	2	10706	52	224
Science & Education	2	22464	165	118
Development	6	119308	794	924
Communication	2	150109	1062	566
Education	1	59193	258	214
Finance	2	45781	496	312
Multimedia	3	121891	769	492
Personalisation	2	30088	175	710
Productivity	1	12479	71	134
Social Network	2	22464	815	104
Tools	6	95647	533	1491
Instance Messaging	1	49277	457	140
Overall	39	940051	7242	7754

Table IV lists the key characteristics of our dataset: (i) the number of apps per category, (ii) the total number of Java source code lines (excluding empty lines), (iii) the total number of Java classes for each app category, and (iv) the number of user reviews we crawled. More details about each specific app and all the collected reviews can be found in our online replication package [22].

As we mentioned in Section II-C we manually labeled a set of 1566 user reviews, while building the taxonomy. During this process we also collected for each category a set of specific expressions and keywords. This was later used to automatically label the entire set of 7754 user reviews using regular expressions. We wanted to take advantage of the entire set of reviews during the training process of the machine learning models, but manually labelling it would have been too time consuming. Therefore we decided to use an automated approach based on regular expressions, although we are aware that it likely introduced false results. Nevertheless machine learning models are often able to deal with some noise in the data and will learn to generalise better than using a set of hand crafted regular expressions. In order to verify this assumption we performed a manual evaluation of the classifications results as described in Section III-C1 using external evaluators.

C. Evaluation Methodology

To answer our research questions from Section III we performed two experiments, one for each question using the dataset described in Section III-B. In the following paragraphs we explain the details of the experiments, how we selected the data for the external evaluation and the metrics we used.

1) ***Experiment I****:* In the first experiment we assess how well **URR** is able to organize reviews according to meaningful and actionable maintenance and evolution tasks for developers. We first perform a quantitative evaluation for measuring the precision, recall, and the F1 score obtained by our approach while classifying reviews according to the high and low level taxonomy from Section II. Additionally we conduct a short survey to gain more qualitative insights.

We mentioned in section II that we created a golden set of 1566 reviews, where we labelled each review with the categories from the high and low level taxonomy that it belongs to. Because this dataset was involved in the process of building the taxonomy, evaluating our approach on it would not yield accurate results. Additionally, using the labels assigned automatically through regular expressions would also not be accurate. Therefore we asked two external evaluators to manually evaluate the output returned by **URR**. One of the evaluators is a PhD student from the University of Zurich that was not otherwise involved in the paper and the second evaluator is a software analyst and developer at Cedacri S.p.A. company[3]. The selection of the evaluators was not random, we involved them because they have different and complementarily backgrounds: the first one has an academic profile while, the second has industrial experience.

In Section II we observed the necessity for our approach to be able to perform multi-label classification, that is for a given review to return the list of categories (or classes) that it belongs to. This makes the evaluation process more difficult, as the ordinary way of performing stratified sampling will not work. Hence, we decided to evaluate our approach per category, by performing the following steps *for each category*:

- ***Selection of the dataset for evaluators***: Before training each classifier, we described in Section II-D that we first split the entire dataset into a test and train set using stratified sampling. The test set contains around 1500 reviews representing 20% of the data, we run the **URR** classifier for the current category on this set and store the predictions for each review. We then randomly selected 200 reviews from it containing the review text and the prediction of the classifier while ensuring that half of reviews are classified as belonging to the current category and half of them as not belonging and store them in a separate file.
- ***Instruction for evaluators***: We then ask the evaluators to say if the **URR** classification is correct or not for each review from the previously saved file. In this way we make sure that we are able to compute both the number of false positives and false negatives. These are then later needed to compute the evaluation metrics. Because the evaluation is performed manually, we are not able to apply k-fold cross validation.

Our taxonomy contains 6 high level and 12 low level categories, therefore in this step we obtain 18 different files for each category which includes a total of 3600 user reviews. Because we built each file separately for a specific category, it is likely that some reviews will show up in several files. We explained to the reviewers that the categories are not exclusive, a single reviews can belong to multiple ones and this represents one of the advantages of our approach.

Although we provided a clear definition for each category, in some cases it is still difficult to say if a certain review belongs to a specific category or not. Whenever the evaluators did not

[3]http://www.cedacri.it/cedacri/en/index.html

agree, they discussed and reached a final conclusion. They disagreed on 384 reviews, representing 10.67% from the total evaluation dataset. The total time necessary for the evaluation was 4 full days.

Given the evaluation results, we report for each high and low level category in Section IV the following widely adopted machine learning metrics: precision, recall and F1 score. The precision is defined as the number of true positives divided by the total number of reviews *classified* as belonging to the positive class (in this case the taxonomy category). The recall is defined as the number of true positives divided by the total number of reviews that *actually* belong to the positive class. The F1 score considers both the precision and recall and we use the general definition form which is the harmonic mean between precision and recall. We manually analysed the reviews classified incorrectly by **URR** to try to understand how we could improve our approach, we discuss our findings in the results section.

After the evaluation of the **URR** results we asked the two participants to answer the questions from a ***Post-experiment questionnaire***[4]. The questionnaire has two parts, in the first part we investigate the perceived difficulty in analysing user reviews and the potential usefulness of our approach. In the second part we ask the evaluators to rate the importance of each category of our taxonomy and then provide a short comment on their rating. We discuss the results in Section IV.

2) ***Experiment II:*** To answer RQ_2, we needed to build an oracle that, for each user review classified by URR (evaluated in RQ_1), reports all the relevant artifacts involved in the proposed changes. Once we have such an oracle, we are able to compute the precision, recall and F1 score for our approach. Therefore, the second experiment includes the following steps:

- ***Sampling of the data***: we asked one of the previous evaluators to build the oracle. The task of the evaluator was to specify whether a software artifact returned by **URR** and associated to a given user review is correct or not (false positive). We also asked the evaluator to inspect the source code and the user reviews of mobile apps to discover missing artifacts (false negatives). This is more difficult and time consuming than the previous task, therefore we had to create a much smaller evaluation dataset. We randomly selected 91 reviews from two of the apps in our dataset.
- ***Creation of the oracle***: we ran the **URR** prototype using the dataset obtained in the previous step and saved for each review the list of source code artifacts returned as output. We then asked the evaluator to report the set of false positive and false negatives for each review. This task required 4 full work days.
- ***Evaluation***: Given the oracle from the evaluator we were able to compute the precision, recall and F1 score obtained by our tool.

[4]http://bit.ly/2dXc9qb

IV. Results

A. RQ_1: *To what extent does the **U**ser **R**equest **R**eferencer organize reviews according to meaningful maintenance and evolution tasks for developers?*

High Level Taxonomy. Table V reports the precision, recall and F1 scores for the high level categories as reported by our external evaluators. Our approach shows very good results for most categories, the overall F1 score is between 82% and 93%. Nevertheless the category *Compatibility* shows the lowest precision. We further investigated the reviews that were classified incorrectly by **URR**. One such example is the following: *"It works flawlessly. It's probably the best. (note: I have upgraded my review from 4 to 5 stars, because the data consumption has been improved)."*. The **URR** classifier likely classified it as belonging to *Compatibility* because of the word *note*, this is also part of the name of a very popular Android phone: *Samsung Galaxy Note*. Users report this name using various forms: *Samsung Note*, *Samsung Galaxy*, *Note 7* or just *note phone*, therefore the classifier learned that this word is a strong indicator for the class *Compatibility*. We tried to remediate this problem by using n-grams features, but in this case it was not enough. Another potential solution is augmenting the training set with enough reviews containing the word *note* with a different meaning than the phone.

The other category with a precision less than 80% is *Resources*. After examining the reviews classified incorrectly we concluded that this result is caused by the high degree of variability of the vocabulary utilized by users (i.e., they often use different keywords and linguistic patterns) to explain resources issues. Again this problem can be easily addressed by extending the dimension of the training set including more instances of reviews discussing resources issues.

Low Level Taxonomy. Table VI reports the precision, recall and F1 scores achieved for the low level categories. As expected, because of the finer grained nature of the low level taxonomy, the precision and recall for some categories are a bit lower than the ones of the high level taxonomy. Regardless, the **URR** results are very good for most categories with few exceptions: *Hardware*, *Performance* and *Memory*. We analysed more in depth why our evaluation reports poorer results for those cases, next we discuss each one separately:

- *Hardware*: after investigating reviews classified incorrectly we noticed that there is a large number of hardware components that users can refer using different terms, abbreviations and ways of writing (*e.g. SD Card, sd-card, sdcard*, etc.), therefore it was difficult for the classifier to learn how to classify them correctly given the current training set;
- *Performance*: one misclassified example is *"One thing I only wish this app was able to switch wallpapers faster than an hour, like maybe every 30 seconds and every couple minutes."*, in this case the user utilised the word *faster* to characterise a feature of the app, but often users describe an app as being fast or slow referring to its performance. Therefore the classifier also learned that this

TABLE V
RESULTS: HIGH LEVEL CATEGORIES (HLC) METRICS

HLC	Precision	Recall	F1 Score
Compatibility	71%	97%	82%
Usage	89%	94%	91%
Resources	79%	99%	88%
Pricing	85%	97%	90%
Protection	89%	98%	93%
Complaint	85%	80%	82%

TABLE VI
RESULTS: LOW LEVEL CATEGORIES (LLC) METRICS

HLC	LLC	Precision	Recall	F1 Score
Compatibility	**Device**	85%	98%	91%
	Android Version	89%	86%	87%
	Hardware	61%	95%	74%
Usage	**App Usability**	92%	91%	91%
	UI	83%	93%	88%
Resources	**Performance**	64%	97%	77%
	Battery	78%	95%	86%
	Memory	68%	95%	79%
Pricing	**Licensing**	91%	98%	94%
	Price	85%	96%	90%
Protection	**Security**	87%	98%	92%
	Privacy	83%	96%	89%

word is a strong indicator for the *Performance* category, the stemming step from our approach likely amplified the problem (it converted *fast* and *faster* to the same root);

- *Memory*: in reviews related to this category, users often report that an app takes too much or little space, therefore often the word *space* is a strong signal for it. On the other hand the word *space* can also have a different meaning, *e.g. "Annoying bar Top bar takes too much space"*, which the *URR* prototype mistakenly classified as a review talking about *Memory* issues.

URR returns very good results for most categories from the high and low level taxonomy and for the few categories with lower precision the results are still promising. As we can observe from Table VII, that shows examples of classified reviews, our prototype is able to correctly classify reviews that highlight important issues encountered by users, which developers should address during their maintenance and evolution activities.

Nevertheless, we plan to address the shortcomings of our approach and improve it in future work. As with most techniques machine learning techniques, one way to increase the precision and recall is to extend the training set. Additionally in Section II-D we mentioned how we expanded our initial set of manually labeled 1566 reviews to a set of 7554 reviews using regular expressions. While this approach was fast, it likely introduced false positives, therefore another possible improvement is manually re-labeling the entire dataset. We expect that in some cases this might still not be enough and in order to increase the recall and precision further we would have to introduce more sophisticated features. For example, to improve the results for the *Compatibility* category we could employ a Named Entity Recognition [30] approach to locate and extract the names of mobile phones in user reviews and use that as a feature during the training of the **URR** classifiers.

Post Experiment Survey. To qualitatively answer RQ1 we analyzed the replies from the two evaluators collected in the post-experiment survey. All detailed answers are reported in our replication package [22].

TABLE VII
EXAMPLES OF REVIEWS CLASSIFIED BY URR

LLC	Review
Device	*"In samsung devices (note 5) when the display turns on the buttons light up.Pls remove it."*
Android Version	*"The app is causing random power off of my Android One device running Android 6.0.1."'*
Hardware	*"doesnt read from SD Card on my tablet just reads from internal memory"*
App Usability	*"Simple text chat windows have worked for decades, reducing usable screen space and adding a distracting background is in no way an improvement."*
UI	*"Add more gestures to view the background unblurred since double tap locks the screen on some phones"*
Performance	*"Poor performance versus using DDG in browser DuckDuckGo is great to use, but I cannot recommend using the app itself."*
Battery	*"Has bugs On receiving USSD screen flickers and constantly drains battery. Lost 20% battery in an hour"*
Memory	*"Still on the first page. Some memory error thingy and asking me to try again shortly. Tried again 1 minute later and still the same."*
Licensing	*"Good app and no ads in free version It was my favourite."*
Price	*"Liked it so much I paid the $3 for the VIP version."*
Security	*"Please add security code so I didn't need to unlock my lock screen two times at the same time."*
Privacy	*"Wtf twitter? Outrageous and unnecessary special permissions since twitter bought the app. The whole point of this app is privacy!"*

The two evaluators confirmed our belief that manually analysing hundreds and thousands of user reviews is a time consuming and difficult task, and classifying reviews according to the **URR** taxonomy can be very useful. Furthermore, they estimated that they could save up to 75% percent of the time required by a manual analysis, using our approach. Another important aspect highlighted by the two participants is that the proposed taxonomy is complete and likely does not miss any useful user review categories. However one of the evaluators found the classification of some reviews concerning RAM or CPU usage a little ambiguous. For example, a review like the following *"RAM Sucker This tiny app reduces my over 500 mb RAM to below 200. Deleting"* can be included in both hardware or performance categories, but the evaluator suggested adding the *Data Usage* as an additional low level category under *Resources*. Finally, from a software development point of view, they considered the categories *Usage*, *Resources* and *Compatibility* to be the most important. They viewed the *Pricing* and *Privacy* categories as least important, nevertheless we argue that these categories reflect relevant issues from the user perspective.

In summary, we conclude that:

RQ$_1$: From the evaluation results we conclude that the **URR** prototype is able to classify reviews according to the high and low level taxonomies with very high precision and recall. According to the evaluators, our approach could save up to 75% of the time required for a manual analysis. The taxonomy is complete and able to categorize reviews according to relevant mobile specific issues and the most important categories are *Usage*, *Resources* and *Compatibility*.

*B. RQ$_2$: Does the **U**ser **R**equest **R**eferencer correctly recommend the software artifacts that need to be modified in order to handle user requests and complaints?*

Table VIII reports the precision, recall and F1 scores achieved by URR when recommending related source code artifacts for specific user reviews. The metrics were computed using the oracle built by the external evaluator during Experiment II as described in Section III-C. User reviews have highly

TABLE VIII
URR'S RESULTS ON LOCALIZING CODE
ARTIFACTS ACCORDING TO USERS REVIEWS FEEDBACK

Quality of Reviews	Precision	Recall	F1 Score
Difficult to Link	41%	**83%**	55%
Easier to Link	**52%**	79%	**63%**
Difficult and Easier to Link	**51%**	**79%**	**62%**

variable quality, therefore we additionally asked the evaluator to estimate whether the reviews from the selected dataset are *Easier* or *Difficult* to link. We wanted to investigate whether the results achieved by out approach are influenced by the quality of reviews.

URR achieves an overall recall of 79% and a precision of 51%, this results are promising as it means that from the recommended software artifacts half of them will be relevant for the specific user review. A developer with knowledge of the application is able to easily filter out the irrelevant ones while taking advantage of our automated approach. Given these results we also have to take into consideration the very low overlap between the technical vocabulary used by software developers in source code artifacts and the informal vocabulary of user reviews. This means that IR methods based on textual similarity will in general, not be able to provide very good results. We were able to partially compensate for the limitations of such methods using the *boosting functionality* of Lucene to increase the similarity scores between user reviews and source code artifacts sharing the same *structure category*.

It is interesting to observe that **URR** is able to achieve a better precision (+11%) and F1 score (+8%) for *Easier* to link reviews. Still the results are not very good, confirming our conjecture that, even when using the *boosting bonus* functionality of Lucene, reviews are difficult to link using textual similarity based methods. Our evaluator pointed out, after the validation, several relevant aspects:

1) ***"Reviews which express a general opinion about the app are very hard to be linked"***, *e.g."The interface is just amazing! You guys could make a similar app for PDF files- if you think about it. It'd be great if you guys got around to doing that."*;
2) However, ***"...reviews that are more precise in their intent and are referred to some app features or some UI elements to improve are more easier to link"***, *e.g."Almost! Great minimalist lock screen". The wave proximity sensor unlocking function should be improved though. As well as adding more options for swiping from upper/lower right/left- not just dialer and camera. And it would also be perfect if you can edit the size of the clock."*;
3) ***"Review quality (number of details, well-formed english, typos or other errors) may affect a lot the linking"***.

In summary, we conclude that:

RQ$_2$: URR achieves promising results recommending related software artifacts for specific user reviews with a recall of 79% and a precision of 51%. Moreover, better quality reviews are easier to link and this can also be seen from the precision and recall **URR** achieves for reviews labeled as *Easier* to link by the evaluator.

V. THREATS TO VALIDITY

Threats to construct validity. Threats to construct validity concern the way in which we set up our study. The main threat to internal validity in our study is that the assessment of **URR** and its taxonomy is performed on data (e.g., the truth set creation) provided by human subjects. Indeed, there is a level of subjectivity in deciding whether a user feedback contained in a review belongs to a specific category of the taxonomy or not. To counteract this issue, we asked two evaluators to evaluate the results of our approach. Furthermore, they discussed their results whenever they had divergent opinions, until they reached a final decision.

Threats to internal validity. These threats concern confounding factors that could influence our results. A first threat involves the taxonomy definition since some of the categories could be overlapping or missing. To alleviate these issues we defined a low level (or fine-grained) set of categories associated to the high level ones to minimize the risk of having an incomplete taxonomy while facilitating the separation of useful users reviews discussing different and relevant aspects. The evaluators involved in our experiments considered our taxonomy to be complete and not miss any significant categories.

Another threat to the internal validity is represented by the possibility that the chosen machine learning algorithm overfits the data. To handle this problem we always select the dataset for manual evaluation, from a 20% randomly selected sample of the entire dataset, that was not used during the training of the machine learning classifiers.

Threats to external validity. External threats concern the generalizability of our findings. We validated our approach on dataset of reviews from 39 open source application available on the F-Droid and Google Play websites. To increase the generalizability of approach, we have selected apps with different sizes and from 17 categories. Furthermore, during the definition of our taxonomy we focused on issues relevant for mobile applications, that are not specific to a certain platform. Nevertheless our dataset is a small sample compared to the total number of apps available on Google Play and might be affected by the app sampling problem [31].

VI. RELATED WORK

A. Analyzing Mobile Applications Reviews

Multiple research papers have investigated the nature of available information in mobile app stores, especially user reviews and tried to automated the process of extracting relevant information from them.

Harman *et al.* [32] first introduced the concept of app store mining and identified the correlation between the rating of an app and the download rank. Khalid *et al.* [33] manually analysed low rated user reviews of iOS apps and identified what are the most common user complaints. Pagano *et al.* [5] conducted an exploratory study on a large number of reviews from the Apple AppStore and built a taxonomy of common topics users talk about in their feedback. The topics identified by Pagano are very general (*praise, helpfulness, feature information,* etc.) and not specific to mobile applications as opposed to our taxonomy. They found that users often report bugs and shortcomings of an app in reviews and that those reports have a strong influence on the rating of an app. Tian *et al.* [34] investigated the characteristics of high rated apps.

AR-MINER [11] represents one of the first automatic approaches to classify user reviews into informative and non-informative content. The paper concluded that only 35.1% of reviews are informative, further motivating the need for tools that automate the process of selecting relevant reviews. Gu and Kim [35] focused on analysing sentiments in user reviews, their approach SUR-MINER summarises sentiments and opinions of reviews and classifies them according to 5 predefined classes (aspect evaluation, bug reports, feature requests, praise and others).

Panichella *et al.* [10] uses a combination of Natural Language Processing, Text and Sentiment analysis techniques to classify reviews according to the following classes: Information Giving, Information Seeking, Feature Request and Problem Discovery. While useful, the list of classes is too general and does not address specific mobile issues.

The closest related work to ours is CLAP [15], this approach is able to automatically categorize reviews into suggestion for new feature request, bug report and other. The tool then clusters the reviews and the developer is presented with a set of clusters that share similar terms. But they still have to analyse each cluster and determine what specific mobile issue they discuss. Furthermore a review will only be assigned to a single cluster, although users often address multiple topics in a single review. Our approach is able to classify reviews according to both high and low level well defined mobile specific issues (*e.g.* performance, battery, memory, etc.), therefore the developer knows right away what the topic is of the returned group of reviews and additionally whenever a review discusses multiple topics, it will be assigned to multiple categories. We believe this will significantly reduce the manual work required to analyse and understand reviews.

B. Linking Informal Documentation to Source Code

Traceability relations between textual artifacts (*e.g.*, requirements and source code) often tend to be incomplete, inconsistent or outdated. Recovering links between such software artifacts is particularly helpful for performing impact analysis and change management during software evolution. Several researchers have investigated different techniques—based on Information Retrieval (IR) — such as Vector Space Model (VSM) [36], Latent Semantic Indexing (LSI) [37], or Jensen & Shannon (JS) similarity model [38] to recover traceability links between different kinds of software artifacts (see, e.g., [39], [40], [38], [41], [42]). Additional approaches have been proposed for (i) locating features/bugs in the source code [43], [44], or tracing different informal textual documentation (e-mails, forum discussions, commit messages, and bug reports) [45], [46], [47], [48], [44], [49], [50], [51] to the source code.

All these methods "are based on the assumption of a consistent lexicon between different artifacts to be traced" (e.g., between requirements and source code)[52], [53]. However, linking user reviews to source code artifacts is a different and more challenging problem for two main reasons: (i) the assumption that a consistent lexicon is used in user reviews and source code is not valid; (ii) often user reviews are very short or the problem users are facing is not properly stated and this makes it harder, if not impossible, to establish a link with the corresponding source code (as also reported by our study participants in Section IV). For this reason, our approach uses more fine grained IR-based techniques constructed using both textual and structural information (i.e., the organization of mobile software projects) of mobile applications (Section II-D4).

The problem of tracing user feedback reported in reviews of mobile apps with their source code is not yet extensively investigate in literature and, to the best of our knowledge, only the approach by Palomba *et al.* [54] is closer to the one we proposed in this paper. Specifically, Palomba *et al.* [54] proposed a tool, called CRISTAL, which traces informative crowdsourced reviews to the source code commits to monitor the extent to which developers accommodate requests reported in user reviews. Thus, CRISTAL is useful for monitoring the changes already applied during the history of a project but, differently from **URR**, is not able to recommend the location of the changes for the current version of an app.

VII. Conclusion

In this paper we present **URR** a novel approach that is able to organise reviews according to well defined fine grained maintenance and evolution tasks (battery, performance, memory, privacy, etc.) and recommend the related source code artifacts. We built our approach on top of a multi-level taxonomy oriented on mobile specific and actionable issues developed through the manual analysis of the reviews from 39 mobile applications.

We thoroughly evaluated our approach and obtained very good results for classifying reviews according to the defined taxonomy. Furthermore, our prototype returns promising results for recommending the related source code files for specific user reviews, especially when taking into account the structure of mobile applications projects (Section II-D4). The external inspectors that performed the evaluation, estimated that our approach could save up to 75% of the time required to manually analyse reviews. We plan to enhance **URR** by: (i) improving the machine learning classifier with additional features; (ii) extending the study to involve the reviews and source code of more mobile applications and (iii) replicating it with additional developers.

Acknowledgments

We would like to thank Carmine Vassallo and Giovanni Grano for their help in the evaluation of our approach. Finally, we acknowledge the Swiss National Science foundation's support for the projects Essentials (SNF Project No. 200020-153129) and SURF-MobileAppsData (SNF Project No. 200021-166275).

REFERENCES

[1] Statista, "Number of apps available in leading app stores as of june 2016," Tech. Rep., 2016, https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

[2] ——, "Cumulative number of apps downloaded from the google play as of may 2016," Tech. Rep., 2016, https://www.statista.com/statistics/281106/number-of-android-app-downloads-from-google-play/.

[3] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *IEEE Transactions on Software Engineering (TSE)*, 2016.

[4] ——, "A survey of app store analysis for software engineering," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2016.

[5] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study." in *In Proceedings of the 21st IEEE International Requirements Engineering Conference (RE 2013)*. IEEE Computer Society, 2013, pp. 125–134.

[6] V. N. Inukollu, D. D. Keshamoni, T. Kang, and M. Inukollu, "Factors Influencing Quality of Mobile Apps:Role of Mobile App Development Life Cycle," *ArXiv e-prints*, Oct. 2014.

[7] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*, Aug 2014, pp. 153–162.

[8] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," in *10th Working Conference on Mining Software Repositories (MSR'13)*, 2013, pp. 41–44.

[9] L. V. G. Carreno and K. Winbladh, "Analysis of user comments: An approach for software requirements evolution," in *35th International Conference on Software Engineering (ICSE'13)*, 2013, pp. 582–591.

[10] S. Panichella, A. Di Sorbo, E. Guzman, C. Visaggio, G. Canfora, and H. Gall, "How can I improve my app? Classifying user reviews for software maintenance and evolution," in *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, 2015, pp. 281–290.

[11] N. Chen, J. Lin, S. Hoi, X. Xiao, and B. Zhang, "AR-Miner: Mining informative reviews for developers from mobile app marketplace," in *36th International Conference on Software Engineering (ICSE'14)*, 2014, pp. 767–778.

[12] S. Panichella, A. Di Sorbo, E. Guzman, C. Visaggio, G. Canfora, G. Gall, H.C., and H. Gall, "Ardoc: App reviews development oriented classifier," in *Foundations of Software Engineering (FSE), 2016 ACM SIGSOFT International Symposium on the*, 2016, p. to appear.

[13] E. Ha and D. Wagner, "Do android users write about electric sheep? examining consumer reviews in google play," in *Consumer Communications and Networking Conference (CCNC), 2013 IEEE*, Jan 2013, pp. 149–157.

[14] J. Oh, D. Kim, U. Lee, J.-G. Lee, and J. Song, "Facilitating developer-user interactions with mobile app review digests," in *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '13. New York, NY, USA: ACM, 2013, pp. 1809–1814. [Online]. Available: http://doi.acm.org/10.1145/2468356.2468681

[15] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta, "Release planning of mobile apps based on user reviews," in *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 2016.

[16] H. Yang and P. Liang, "Identification and classification of requirements from app user reviews," in *Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering (SEKE)*. Knowledge Systems Institute, 2015, pp. 7–12.

[17] E. Guzman, M. El-Halaby, and B. Bruegge, "Ensemble Methods for App Review Classification: An Approach for Software Evolution." in *Proc. of the Automated Software Enginering Conference (ASE)*, 2015, pp. 771–776.

[18] E. Guzman, O. Aly, and B. Bruegge, "Retrieving Diverse Opinions from App Reviews." in *Proc. of the Empirical Software Engineering and Measurement Conference (ESEM)*, 2015, pp. 1–10.

[19] C. Iacob, R. Harrison, and S. Faily, "Online reviews as first class artifacts in mobile app development," in *Mobile Computing, Applications, and Services*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, G. Memmi and U. Blanke, Eds. Springer International Publishing, 2014, vol. 130, pp. 47–53.

[20] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? on automatically classifying app reviews," in *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*, 2015, pp. 116–125.

[21] A. Di Sorbo, S. Panichella, C. Alexandru, J. Shimagaki, C. Visaggio, G. Canfora, and H. Gall, "What would users change in my app? Summarizing app reviews for recommending software changes," in *Foundations of Software Engineering (FSE), 2016 ACM SIGSOFT International Symposium on the*, 2016, p. to appear.

[22] S. P. H. G. Adelina Ciurumelea, Andreas Schaufelbühl, "Analyzing reviews and code of mobile apps for a better release planning organization replication package," University of Zurich, Tech. Rep., 2016, https://doi.org/10.5281/zenodo.161842.

[23] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, "What do mobile App users complain about? a study on free iOS Apps," *IEEE Software*, no. 2-3, pp. 103–134, 2014.

[24] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[25] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[26] Turi. Gradient boosted regression trees. [Online]. Available: https://turi.com/learn/userguide/supervised-learning/boosted_trees_regression.html

[27] M. F. Porter, "Readings in information retrieval," K. Sparck Jones and P. Willett, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, ch. An Algorithm for Suffix Stripping, pp. 313–316. [Online]. Available: http://dl.acm.org/citation.cfm?id=275537.275705

[28] C. Hu and I. Neamtiu, "Automating gui testing for android applications," in *in Proceeding of the 6th international workshop on Automation of software test*, Riverside, CA, USA, 2011, pp. 77–83.

[29] FDroid. (2016) F-droid ? free and open source android app repository. [Online]. Available: https://f-droid.org

[30] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007. [Online]. Available: http://www.ingentaconnect.com/content/jbp/li/2007/00000030/00000001/art00002

[31] W. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang, "The app sampling problem for app store mining," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 123–133. [Online]. Available: http://dl.acm.org/citation.cfm?id=2820518.2820535

[32] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: MSR for app stores," in *9th IEEE Working Conference of Mining Software Repositories, MSR 2012, June 2-3, 2012, Zurich, Switzerland*. IEEE, 2012, pp. 108–111.

[33] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, "What do mobile app users complain about?" *IEEE Software*, vol. 32, no. 3, pp. 70–77, May 2015.

[34] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan, "What are the characteristics of high-rated apps? a case study on free android applications," in *31st IEEE International Conference on Software Maintenance and Evolution (ICSME 2015)*, 2015, pp. 301–310.

[35] X. Gu and S. Kim, "What parts of your apps are loved by users?" in *30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015)*, 2015, p. to appear.

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

[37] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[38] A. Abadi, M. Nisenson, and Y. Simionovici, "A traceability technique for specifications," in *Proceedings of the 16th IEEE International Conference on Program Comprehension*. IEEE CS Press, 2008, pp. 103–112.

[39] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[40] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proceedings of 25th International Conference on Software Engineering*, Portland, Oregon, USA, 2003, pp. 125–135.

[41] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *Proceedings of the 32Nd*

ACM/IEEE International Conference on Software Engineering - Volume 1, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 95–104. [Online]. Available: http://doi.acm.org/10.1145/1806799.1806817

[42] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 522–531. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486857

[43] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013.

[44] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, Nov 2013, pp. 345–355.

[45] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, 2010, pp. 375–384.

[46] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey, "Crowd documentation: Exploring the coverage and dynamics of API discussions on stack overflow," Georgia Tech, Tech. Rep. GIT-CS-12-05, 2012.

[47] S. Panichella, J. Aponte, M. Di Penta, A. Marcus, and G. Canfora, "Mining source code descriptions from developer communications," in *IEEE 20th International Conference on Program Comprehension (ICPC'12)*, 2012, pp. 63–72.

[48] C. Vassallo, S. Panichella, M. Di Penta, and G. Canfora, "Codes: Mining source code descriptions from developers discussions," in *Proceedings of the 22Nd International Conference on Program Comprehension*, ser. ICPC 2014. New York, NY, USA: ACM, 2014, pp. 106–109. [Online]. Available: http://doi.acm.org/10.1145/2597008.2597799

[49] A. D. Sorbo, S. Panichella, C. A. Visaggio, M. D. Penta, G. Canfora, and H. C. Gall, "DECA: development emails content analyzer," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, 2016, pp. 641–644.

[50] ——, "Development emails content analyzer: Intention mining in developer discussions (T)," in *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, 2015, pp. 12–23.

[51] G. Schermann, M. Brandtner, S. Panichella, P. Leitner, and H. C. Gall, "Discovering loners and phantoms in commit and issue data," in *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, ICPC 2015, Florence/Firenze, Italy, May 16-24, 2015*, 2015, pp. 4–14.

[52] A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella, "Applying a smoothing filter to improve ir-based traceability recovery processes: An empirical investigation," *Information & Software Technology*, vol. 55, no. 4, pp. 741–754, 2013.

[53] G. Capobianco, A. D. Lucia, R. Oliveto, A. Panichella, and S. Panichella, "Improving ir-based traceability recovery via noun-based indexing of software artifacts," *Journal of Software: Evolution and Process*, vol. 25, no. 7, pp. 743–762, 2013.

[54] F. Palomba, M. Linares-Vasquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "User reviews matter! tracking crowdsourced reviews to support evolution of successful apps," in *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, Sept 2015, pp. 291–300.