This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

Listening to the Crowd for the Release Planning of Mobile Apps

Simone Scalabrino, *Student Member, IEEE,* Gabriele Bavota, *Member, IEEE*
Barbara Russo, *Member, IEEE,* Rocco Oliveto, *Member, IEEE,* Massimiliano Di Penta *Member, IEEE*

Abstract—The market for mobile apps is getting bigger and bigger, and it is expected to be worth over 100 Billion dollars in 2020. To have a chance to succeed in such a competitive environment, developers need to build and maintain high-quality apps, continuously astonishing their users with the coolest new features. Mobile app marketplaces allow users to release reviews. Despite reviews are aimed at recommending apps among users, they also contain precious information for developers, reporting bugs and suggesting new features. To exploit such a source of information, developers are supposed to manually read user reviews, something not doable when hundreds of them are collected per day. To help developers dealing with such a task, we developed CLAP (**C**rowd **L**istener for rele**A**se **P**lanning), a web application able to (i) categorize user reviews based on the information they carry out, (ii) cluster together related reviews, and (iii) prioritize the clusters of reviews to be implemented when planning the subsequent app release. We evaluated all the steps behind CLAP, showing its high accuracy in categorizing and clustering reviews and the meaningfulness of the recommended prioritizations. Also, given the availability of CLAP as a working tool, we assessed its applicability in industrial environments.

Index Terms—Release Planning, Mobile Apps, Mining Software Repositories

1 Introduction

The wide diffusion of mobile devices is making the apps market a tremendous success. Developers can easily join such a market by publishing their apps in an online store, making them available for download to interested users. The five leading app stores (*i.e.,* Google Play[1], Apple App Store[2], Windows Store[3], Amazon Appstore[4], and BlackBerry World[5]) count, overall, over five million apps, with the most popular (*i.e.,* Google Play) accounting alone for over two million apps. These numbers basically highlight one bold fact: Succeeding in this market is not as simple as joining it, since the competition is strong and users interested in a specific "type" of app (*e.g.,* a GPS navigator) can generally choose among hundreds of similar apps (*e.g.,* a simple search for "GPS navigator" on Google Play results in over 200 apps).

To guide the users in the choice of the best apps to download, the app stores feature user reviews, having the purpose of (i) allowing users to indicate on a five-star scale how much they liked the app (review's rating), and (ii) explaining, in a free text form, why the users like or do not like the app, report bugs or request new features. User reviews thus also represent an important source of information that developers can exploit to guide the successful evolution of their apps (*e.g.,* by rapidly fixing bugs reported by users and/or by implementing recommended features). Past studies have shown that various indicators of the app quality, such as the used APIs change- and fault-proneness [9], [39], the presence of ads [35], or, in general, characteristics of the apps or of the device on which it is deployed [38] significantly correlate with the app rating. Also, there is empirical evidence that satisfactorily addressing requests made through user reviews is likely to increase the app rating [32]. However, manually read each user review and verify if it contains useful information is not doable for popular apps receiving hundreds of reviews per day. For such reasons, researchers have developed approaches to analyze the content of user reviews with the aim of crowd-source requirements [15]–[17], [23], [26], [28], [33]. Among others, AR-Miner [15] is able to discern informative reviews, group and rank them in order of importance.

While approaches to identify and classify relevant and informative reviews have been proposed, it would be desirable to have a fully-automated (or semi-automated) solution that, given the user reviews for an app, recommends which ones—being them requests for new features or for bug fixes—should be addressed in the next release.

In this paper we propose CLAP, an approach to (i) automatically categorize user reviews into *functional bug report*, *suggestion for new feature*, *report of performance problems*, *report of security issues*, *report of excessive energy consumption*, *request for usability improvements* and *other* (including non-informative reviews); (ii) cluster together related reviews in a single request, and (iii) recommend which review cluster developers should satisfy in the next release. Unlike AR-Miner [15], CLAP classifies reviews into specific categories (*e.g., report of security issues*), providing additional insights to the developer about the nature of the review. Also, while AR-Miner simply provides a ranking of the user reviews based on their importance as assessed by a pre-defined

- *S. Scalabrino and R. Oliveto are with the University of Molise, Italy. E-mail: {simone.scalabrino, rocco.oliveto}@unimol.it*
- *G. Bavota is with the Università della Svizzera italiana (USI), Switzerland. E-mail: gabriele.bavota@usi.ch*
- *B. Russo is with the Free University of Bozen-Bolzano, Italy. E-mail: barbara.russo@unibz.it*
- *M. Di Penta is with the University of Sannio, Italy. E-mail: dipenta@unisannio.it*

1. https://play.google.com/store
2. https://www.appstore.com/
3. https://www.microsoft.com/store/apps
4. https://www.amazon.com/getappstore
5. https://appworld.blackberry.com

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

formula, CLAP learns from past history of the same app or of other apps to determine the factors that contribute to determining whether a review should be addressed or not. As compared to other existing techniques (*e.g.,* [16], [17], [28], [33]), CLAP offers a complete solution, going from the reviews categorization up to their prioritization in sight of the next app release. CLAP combines natural language processing techniques and machine learning for review classification, uses clustering techniques for grouping reviews, and, finally, again a machine learner that recommends the implementation of specific clusters of reviews by relying on features such as the number of reviews in a cluster or the number of different hardware devices affected by a bug.

We thoroughly evaluated each step of CLAP, as well as of the whole, publicly available tool. First, we performed a study to assess the accuracy of CLAP in classifying reviews. The second validation stage aimed at comparing the review clusters generated by CLAP with respect to manually-produced clusters. The third validation assessed the ability of CLAP to recommend changes to implement in sight of the next app releases. Last, but not least, in the fourth validation stage we provided our tool to managers of three Italian software companies to get quantitative and qualitative feedback about the applicability of CLAP in their everyday decision making process. CLAP is publicly available as a Web application[6]. Everyone can use it by registering and importing from the Google Play store the reviews' of his/her apps. Also, the user just interested in having a look to the CLAP features can login with a demo account (login: *demonstration*, password: *playwithclap*).

This paper extends our ICSE 2016 paper [40] in the following aspects:

- *We extended* CLAP *to provide a more fine-grained categorization of users' reviews*. Indeed, the original version of CLAP was only able to classify users' reviews falling in three possible categories: *functional bug report*, *suggestion for new feature*, and *other* (*i.e.,* non-informative reviews). However, our industrial contacts, involved in the study reported in this paper, stressed the importance of mining reviews reporting bugs related to non-functional requirements. Thus, we extended our approach to also consider additional four categories of reviews: *report of security issues*, *report of performance problems*, *report of excessive energy consumption*, and *request for usability improvements*.
- *We consider new predictor variables when categorizing users' reviews*. In particular, we also exploit the length of the reviews and the category of the app (*e.g.,* game, productivity) for which the review has been posted.
- *We re-run and expanded our experiment on reviews' categorization.* Having new categories of reviews (see previous point) made necessary to re-run the evaluation of the first step behind CLAP (*i.e.,* the reviews' categorization). To this aim, we manually labeled the category of 3,000 user reviews (as compared to the 1,000 considered in our ICSE paper), and contrast the automatic CLAP categorization against our manually defined "oracle". We also compared the classification accuracy achieved by CLAP and AR-MINER.
- *We compared the reviews' clustering performed by* CLAP *with those performed by* AR-MINER.
- *We re-run and expanded our experiment on reviews' clusters prioritization.* In our ICSE paper we considered in this study reviews from five apps, while in this paper we run such a study on 725 reviews from 14 apps.
- *We describe in much more details our tool and deployed it.* CLAP is publicly available as a Web application[7]. Everyone can use it by registering and importing from the Google Play store the reviews' of his/her apps. Also, the user just interested in having a look to the CLAP features can login with a demo account (login: *demonstration*, password: *playwithclap*).

Paper structure. Section 2 details the various phases of CLAP, and also describes the tool prototype. Section 3 details the empirical evaluation definition, design, and planning, whereas its results are reported in Section 4. Section 5 discusses the threats to the study validity. Related work is discussed in Section 7. Finally, Section 8 concludes the paper and outlines directions for future work.

2 CLAP IN A NUTSHELL

CLAP provides support to developers for the release planning of mobile apps by automatically analyzing its user reviews through a three-step process detailed in the following subsections. As previously said, CLAP has been implemented as a publicly available web application. Its current implementation supports the automatic import of the user reviews from Google Play, although it can be generalized to work with any other app store. Figure 1 shows an overview of the workflow of CLAP.

2.1 Categorizing User Reviews

The first step aims at classifying user reviews into seven categories: *functional bug report*, *suggestion for new feature*, *report of performance problems*, *report of security issues*, *report of excessive energy consumption*, *request for usability improvements* and *other*.

The rationale is that, as it will be clear in Section 2.3, developers can use different motivations to decide upon fixing bugs, implementing requests for new features or improving the app non-functional requirements. Other tools such as AR-MINER [15] classify reviews into informative and non-informative. In our case, we make a more specific classification of informative reviews, whereas the non-informative ones fall into the *other* category.

CLAP uses the *Weka* [7] implementation of the Random Forest machine learning algorithm [11] to classify user reviews. The Random Forest algorithm builds a collection of decision trees with the aim of solving classification-type problems, where the goal is to predict values of a categorical variable from one or more continuous and/or categorical predictor variables. In our work, the categorical dependent variable is represented by the type of information reported in the review, and we use the rating of the user reviews and the terms/sentences they contain as predictor variables. We have chosen Random Forest after experimenting with different

6. https://dibt.unimol.it/CLAP/

7. https://dibt.unimol.it/CLAP/

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

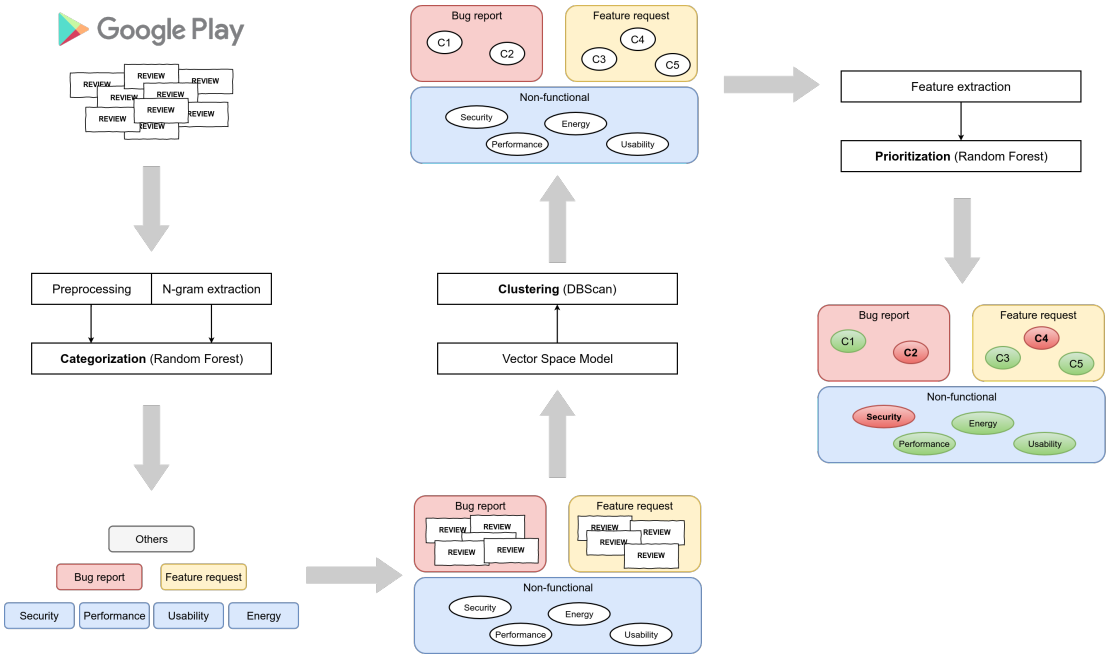

Fig. 1: CLAP workflow

machine learner algorithms (details of the comparison in Section 5).

A possible problem is represented by the fact that some of the categories are rarer than others. For example, reviews reporting functional bugs or suggesting new features are likely to be more than those reporting security issues. In such a context, a machine learning algorithm will tend to assign reviews to more frequent classes, because an error on underrepresented categories is more acceptable than an error on other categories as for overall accuracy. In order to prevent this kind of problem, we use a SMOTE filter [13] to balance the training set. SMOTE creates artificial instances of a specific category, based on the real instances from the training set. We first select the most represented category, then we run SMOTE on all the other categories, so that the number of instances of each category is equal.

The user review classifier exploits the reviews' text to extract features. To this aim, we adopt a customized text preprocessing to characterize each review on the basis of its textual content. The steps of such a process are detailed in the following.

Handling Negations. Mining text in code reviews present challenges related to negation of terms. For instance, reviews containing the words "fix" and "problem" are likely to report a bug, while reviews containing words like "lag" and "slow" are more likely to report performance issues. However, there is a strong exception to this general trend that is due to the negation of terms. Consider the following review: "*I love it, it runs smooth no lags or glitches*". It is clear that in this case, the context in which the words "lag" and "glitches" are used does not indicate any performance issue. However, the presence of these words in the review could lead to misclassifications from the prediction model. We tackle this problem introducing an additional pre-processing step, which aims at removing negated terms from the text. The first step consist in the tokenization of the original text, which is divided into simpler unities. Such unities can be words, numbers or punctuation marks. Then, the sequence of tokens is analyzed and negated terms are removed. When a negation is encountered, one or more subsequent tokens are skipped, based on the kind of negation. Specifically, "no" or "not" imply the skipping of only one following token (*e.g.*, no bugs), while "don't" or "do not" will result in the skipping of two tokens (*e.g.*, does not have bugs), because they are auxiliaries, and it is expected that a verb follows them. Moreover, it is possible that a negation is associated to multiple terms: for example, in the case of "no lags or bugs", if we just skip the first term, we would miss the negation of "bugs". To prevent this situation, when a conjunction (*i.e.*, "and", "or" or a comma) is encountered after a negated term, we treat it like a negation. For example, the sentence "I experienced no lag, freezes or issues" is treated as "I experience no lag no freezes no issues" and, thus, it is transformed in "I experience". The final result is the elimination of the negated terms "freezes", "issues" and "lag". Figure 2 shows a state diagram of the negation handling process. The initial state is *ACCEPT*. For each token in the sequence, there might be a change in the status. When the status is *ACCEPT* or the status changes in *ACCEPT*, the token taken into account is accepted, while it is discarded in all other cases. It is worth noting that some tokens may be eliminated even if there is no negation. For example, when a "do" is encountered, it is discarded, since the state is switched to *AUX*. These cases are deliberately not handled, because such tokens would be eliminated anyway in the next step (*i.e.*, stop-words removal).

In the previous version of CLAP [40] the handling of negations was performed by exploiting the Stanford parser [37]. However, we noticed that in some cases it was not able to accurately remove negated terms. This happened especially in the presence of typos (*e.g.*, "dont" instead of "don't" used in the review), quite common in user reviews.

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

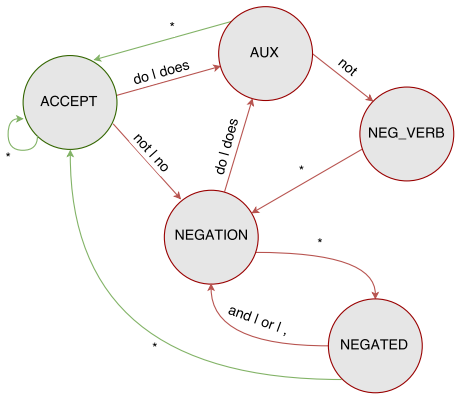

Fig. 2: State diagram of negation terms removal. Tokens on green arrows are accepted, while tokens on red arrows are discarded. The wildcard * means "any other token".

TABLE 1: Evaluation of the negation handling techniques.

	false positives	*false negatives*	*true positives*
State-diagram Approach	0	69	209
Stanford parser	0	194	84

Also, when importing a large number of reviews from the Google Play store, the computational cost of this text preprocessing step was quite high. For this reasons, we designed the simple state-diagram-based negation handling approach described above. We manually evaluated this approach on a set of randomly selected 400 reviews from the dataset by Chen *et al.* [14]. In particular, we selected 200 reviews containing at least one negation and 200 without any negation. We used regular expressions to select the reviews containing/not containing negated terms. We used both our new state-diagram approach as well as the approach based on the Stanford parser, used in the previous version of CLAP [40] to remove negated terms from the set of 400 reviews. Then, one of the authors manually analyzed the 400 original reviews as well as the output produced by the two negations handling approaches. In particular, he counted (i) the number of not negated "useful" words (*i.e.*, words not present in the stop-words list) erroneously removed from both reviews containing/not containing negations (*false positives*), (ii) the number of negated terms correctly removed (*true positives*), and (iii) the number of missed (not removed) negated terms (*false negatives*).

Table 1 reports the results of such evaluation. The implemented state-diagram approach removes about 45% more negations than the one based on the Stanford parser. Also, while the simple state-diagram approach takes ∼10 seconds to process the 400 reviews, the one based on the Stanford parser requires ∼80 seconds.

Stop-words. Many words, such as conjunctions, pronouns, adverbs and so on, are present in natural language texts, but they do not add informative content. In this phase we remove terms belonging to the English stop-words list[8].

Stemming In natural language, the same root term could appear in many forms. For example, the term "bug" could also appear in plural form ("bugs") or as an adjective ("buggy"). It could be helpful for the machine learning classifier to reduce all such different forms to a single one, in order to treat them as if they were exactly the same word. To mitigate this problem, we apply the Porter stemmer [34] to reduce all words to their root. Continuing the previous example, the terms "bugs", "buggy" and "bug" will be all reduced to "bug".

Unifying Synonyms. One possibility to unify synonyms would be to use existing thesaurus such as WordNet [30]. However, in the context of user reviews, we found that general-purpose thesaurus are not adequate. Thus, we rely on a customized dictionary of synonyms that we defined by manually looking at 1,000 reviews (not used in the empirical evaluation) we collected for a previous work [9]. Examples of synsets we obtained are {*crash*, *bug*, *error*, *fail*} (terms related to a bug/crash), {*add*, *miss*, *lack*, *wish*} (terms related to the need for adding a new feature) or {*battery*, *energy*} (terms related to a report of excessive energy consumption). Noticeably, words such as *energy* and *battery* would not be considered synonyms by a standard thesaurus, while they are very likely to indicate the same concept in mobile apps reviews.

N-grams extraction. Besides analyzing the single words contained in each review, we extract the set of *n-grams* composing it, considering $n \in [2 \dots 4]$. For instance, the extraction of n-grams from a review stating "*The app resets itself; Needs to be fixed*" will result in the extraction of n-grams like *resets itself*, *needs to be fixed*, *etc.* Note that the three preprocessing steps described above are not performed on the n-grams (*i.e.,* they only affect the single words extracted from the review). This is done to avoid loosing important information embedded in n-grams. For example, managing negations is not needed when working with n-grams, since n-grams extracted from a review like "*I love it, it runs smooth no lags or glitches*" will include *no lags*, *no lags or glitches*, *etc.* Synonyms merging also is not applied to n-grams to avoid changing their meaning. Anyhow, n-grams containing only stop-words (*e.g.,"but I"*) are removed, because they could be present in many reviews, thus being useless for discriminating among categories of reviews.

After text preprocessing has been performed, the resulting bag of words and n-grams can be used as features for the classifier. In addition, we use three additional features that can help the machine learning algorithm to categorize the review correctly: (i) the user rating provided along with the review, (ii) the length of the review's text, and (iii) the app's category.

Rating. It is likely that the rating of a review is somehow related to its "category". For example, reviews with very positive ratings might be more likely to be less informative (category *other*), because users may just express their satisfaction; on the other hand, negative reviews might be more likely to indicate a bug report or a suggestion for a new feature.

Review length. We expect the length of a review to be a good proxy of the quantity of information present in it. We expect this predictor to be helpful, for example, in spotting reviews recommending the implementation of new features (since we expect such reviews to be particularly long).

App's category. When using single words as predictor variables (as in our approach), the semantic meaning of each word is lost, because there is no context. While n-grams

8. https://code.google.com/p/stop-words/

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

help in alleviating this problem, they may not be sufficient. Consider, for example, the word "freeze": in the context of a messaging app, it may indicate that the app stopped working, thus being the symptom a bug report. Nevertheless, if the same word is used in a review about a game, it may indicate that the game lags too much, thus possibly indicating a *report of performance problems*. To help the classifier in identifying the semantic context of the review, we include the category of the app as a predictor variable. Play Store divides apps in more than 40 categories. In order to reduce the number of possible values for such a variable, we defined 10 macro-categories in which we mapped each of the Google Play categories[9].

Training data, with pre-assigned values for the dependent variables is used to train the classifier, in our case the Random Forest model. The constructed decision trees are represented by a set of yes/no questions that split the training sample into gradually smaller partitions that group together cohesive sets of data, *i.e.,* those having the same value for the dependent variable.

2.2 Clustering Related Reviews

In order to identify groups of related reviews (*e.g.,* those reporting the same bug), we cluster reviews belonging to the same category (*e.g.,* those in *functional bug report*). Clustering reviews is needed for two reasons: (i) developers having hundreds of reviews in a specific category could experience information overloading, wasting almost all advantages provided by the review classification, and (ii) knowing the number of users who are experiencing a specific problem (bug) or that desire a specific feature, already represents an important information about the urgency of fixing a bug/implementing a feature. Note that we only cluster reviews classified as *functional bug report* or *suggestion for new feature* since only for such categories we could have a benefit identifying different groups. Indeed, reviews belonging to the category *other* are not informative, so they do not need to be analyzed at all, while reviews belonging to the four considered non-functional requirements (*report of security issues*, *report of performance problems*, *report of excessive energy consumption* and *request for usability improvements*) should already represent cohesive clusters (*i.e.,* issues with the same non-functional requirement), and thus they should not be further partitioned.

Review clustering is performed by applying DBSCAN [19], a clustering algorithm identifying clusters as areas of high element density, assigning the elements in low-density regions to singleton clusters (*i.e.,* clusters only composed of a single element). DBSCAN, differently from other clustering approaches widely exploited in the literature (*e.g.,* k-means), does not require the *a-priori* definition of the number of clusters to extract, an information clearly not available when the clustering process starts.

In CLAP, the elements to clusters are the reviews in a specific category and the distance between two reviews r_i and r_j is computed as: $dist(r_i, r_j) = 1 - VSM(r_i, r_j)$, where VSM is the Vector Space Model [8] cosine similarity between r_i and r_j. Before applying VSM the text in the reviews is processed as described in the categorization step (Section 2.1), with the only exception of the synonyms merging. Indeed, merging synonyms before clustering could be counterproductive since, for example, a review containing "freezes" and a review containing "crash" could indicate two different issues. DBSCAN does not require the definition a-priori of the number of clusters to extract. However, DBSCAN requires the setting of two parameters: (i) $minPts$, the minimum number of points required to form a dense region, and (ii) ϵ, the maximum distance that can exist between two points to consider them as part of the same dense region (cluster). We set $minPts = 2$, since we consider two related reviews sufficient to create a cluster, while in Section 3 we describe how to set ϵ.

9. The mapping from the apps' categories in Google Play to our 10 macro-categories is available in our replication package [36].

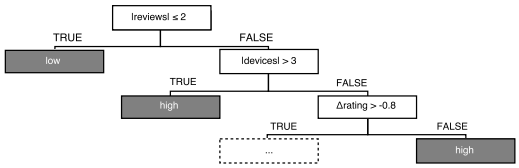

Fig. 3: Example of classification tree generated by CLAP when prioritizing clusters.

2.3 Prioritizing Review Clusters

The clusters of reviews belonging to the *functional bug report* and *suggestion for new feature* categories as well as those grouping together reviews pointing to issues in non-functional requirements (*i.e., report of security issues*, *report of performance problems*, *report of excessive energy consumption* and *request for usability improvements*) are prioritized with the aim of supporting release planning activities. Also in this step CLAP makes use of the Random Forest machine learner with the aim of labeling each cluster as *high* or *low* priority, where *high* priority indicates clusters CLAP recommends to be implemented in the next app release. Thus, the dependent variable is represented by the cluster implementation priority (*high* or *low*), while we use as predictor features:

The number of reviews in the cluster **($|reviews|$).** The rationale is that a bug/issues reported (feature suggested) by several users should have a higher priority to be fixed (implemented) than a bug/issue (feature) experienced (wanted) by a single user.

The average rating of the cluster **($\overline{rating}$).** We hypothesize that a review cluster having a low average (*i.e.,* arithmetic mean) rating has a higher chance to indicate a higher priority bug/issue (or a feature to be implemented urgently) than a cluster containing highly-rated reviews, and thus should have a higher priority. For example, people frustrated by the presence of critical bugs as well as by the miss of a killer feature are more likely to lowly rating the app.

The difference between the average rating of the cluster and the average rating of the app **($\Delta rating_{app}$).** This feature aims at assessing the impact of a specific cluster on the app total rating. We expect a lower difference (especially negative ones) to indicate higher priority for the cluster (*e.g.,* if the average rating of the cluster is 2.5 and the average app rating is 4, then the difference is -1.5).

The number of different hardware devices in the cluster **($|devices|$).** One of the information available to app developers when exporting their reviews from Google Play is

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

the "Reviewer Hardware Model", reporting the name of the device used by the reviewer. We conjecture that the higher $|devices|$, the higher the priority of a cluster. For example, if a cluster of functional/non-functional bug report reviews contains reviews written by users exploiting several different devices, the bug object of the cluster is likely to affect a wider set of users with respect to a bug only reported by users working with a specific device. Similarly, this holds in the case of "desired features", since the same app can expose different features on different devices (*e.g.*, on the basis of the screen size).

Also in this case, historical data with known (and labeled) value of the dependent variable is used to build the Random Forest decision tree. Note that, given the different nature of reviews belonging to different types of clusters (*i.e.*, *functional bug report*, *suggestion for new feature*, *report of security issues*, *report of performance problems*, *report of excessive energy consumption*, *request for usability improvements*), the prioritization is performed separately for clusters containing the different types of reviews. A portion of a tree generated in this step can be found in Figure 3.

It is important to point out that CLAP prioritizes clusters within each review category. It does not use the category for prioritization purpose. In principle, one could be tempted to give high priority to some categories, *e.g.*, *report of security issues* or *functional bug report*. However, we decided not to do so as the intent of CLAP is to provide the project manager with all elements—in our case categorization and clustering first, within-category prioritization after—to make a decision, and not replacing such a decision.

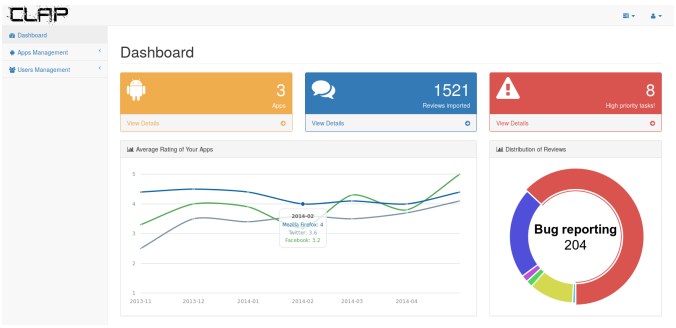

Fig. 4: Dashboard of CLAP presented to a developer after the login.

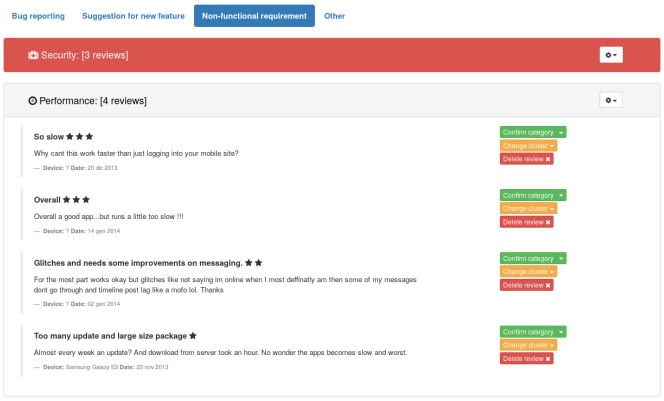

Fig. 5: Non-functional reviews from Facebook app. Non-functional categories are shown as if they were clusters.

2.4 CLAP Prototype

CLAP allows developers to manage users' reviews of their apps. When a registered user logs in, she is presented with a dashboard (Figure 4), showing (i) a chart presenting the evolution over time of the average rating of the apps she imported in CLAP, (ii) a pie chart depicting the number of the reviews loaded in CLAP for each of the seven categories and (iii) three boxes reporting key information, including the number of apps currently managed via CLAP, the total number of reviews imported from the Google Play store, and the number of high priority tasks (*i.e.*, high priority clusters). Each of these boxes represents a shortcut to the related information (*e.g.*, it allows to quickly access the high priority tasks). In the example shown in Figure 4 the user imported in CLAP user reviews from the MOZILLA FIREFOX, FACEBOOK, and TWITTER apps. By selecting one of the imported apps it is possible to visualize its reviews are shown in Figure 6.

CLAP groups the app's reviews into four tabs: One grouping reviews reporting functional bugs, one for requests for new features, one for reviews related to non-functional requirements, and one for the "*Other*" category. In the "Non-functional requirements" tab, each category of reviews (*i.e.*, *report of security issues*, *report of performance problems*, *report of excessive energy consumption* and *request for usability improvements*) is shown as a cluster of related reviews (see Figure 5). Clusters painted in red indicate "high priority" clusters, while those in grey are classified by CLAP as "low priority" (*e.g.*, in Figure 5 the cluster related to *report of security issues* is marked as high priority).

A similar organization is present in the tabs reporting reviews related to functional bugs and to recommendations for new features. The only difference is that here, each cluster, groups together reviews reporting the same functional bug or recommending the implementation of the same feature. Figure 6 shows the GUI related to the *functional bug reporting* tab. Each cluster has a label composed of (i) a simple identifier (*e.g.*, C1), and (ii) the four most frequent terms in the reviews belonging to it. Again, red clusters are those marked by CLAP as "high priority", while grey clusters have a "low priority". The tool also provides a feedback mechanism to allow the developer to indicate whether or not she is going to implement the reviews contained in a cluster (the gear button present on each cluster). Such a manual feedback can be used by the developer to expand/revise the automatic prioritization training set according to the reviews she actually implements. This allows to adapt CLAP's future recommendations to the specific user preferences (*e.g.*, some users could prefer to always prioritize clusters having high values of $|reviews|$, while others could focus more on those having very low $\Delta rating$).

Finally, by expanding a cluster, one can see the reviews it contains. As it can be seen in the example, the two reviews of cluster C56 report a similar problem: the users say that the TWITTER app has some kind of problem with notifications. Also in this case there is a feedback mechanism to change the review category (consequently modifying the training set used by CLAP to learn how to categorize reviews) or to assign it to a different cluster (see Figure 6). This option

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

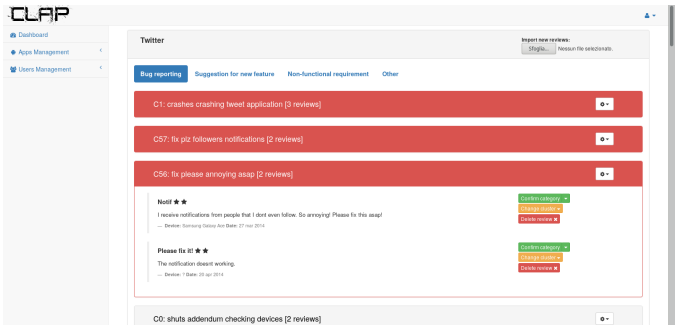

Fig. 6: Reviews of Twitter app imported in CLAP and automatically categorized, clustered and prioritized.

allows developers to manually correct clustering mistakes made by CLAP. Developers can also lock a cluster they manually defined so that, when new reviews are imported, CLAP will not modify such clusters.

3 Empirical Study Design

The *goal* of this study is to evaluate CLAP in terms of its (i) accuracy in categorizing user reviews in the seven categories of interest (*i.e., functional bug report*, *suggestion for new feature*, *report of performance problems*, *report of security issues*, *report of excessive energy consumption*, *request for usability improvements* and *other*), (ii) ability in clustering related user reviews belonging to the same category (*e.g.,* all reviews reporting the same functional bug), (iii) ability in proposing meaningful recommendations on how to prioritize the bugs to be fixed and new features to be implemented while planning the next release of the app, and (iv) its suitability in an industrial context. The *context* of the study consists of 4,025 reviews of Android mobile apps and three Italian software companies.

3.1 Research Questions

In the context of our study we formulated the following four research questions (RQ):

- **RQ**$_1$: *How accurate is* CLAP *in classifying user reviews in the considered categories?* This RQ assesses the accuracy of CLAP in classifying user reviews in the considered categories. It aims at evaluating the step "categorizing user reviews" described in Section 2.1.
- **RQ**$_2$: *How meaningful are the clusters of reviews generated by* CLAP*?* This RQ focuses on the meaningfulness of clusters of reviews extracted by CLAP in a specific category of reviews (*e.g.,* those reporting functional bugs). We are interested in assessing the differences between clusters of reviews automatically produced by CLAP with respect to those manually produced by developers. RQ$_2$ evaluates the step "clustering related reviews" described in Section 2.2.
- **RQ**$_3$: *How accurate is the reviews prioritization recommended by* CLAP*?* Our third RQ aims at evaluating the relevance of the priority assigned by CLAP to the reviews to be implemented in sight of the next release of the app. We assess the ability of CLAP in predicting which functional/non-functional bugs will be fixed (features will be implemented) by developers among those reported (requested) in user reviews of release r_i when working on release r_{i+1}. This RQ evaluates the prioritization step described in Section 2.3.
- **RQ**$_4$: *Would actual developers of mobile applications consider exploiting* CLAP *for their release planning activities?* For a tool like CLAP, a successful technological transfer is the main target objective. In RQ$_4$ we investigate the industrial applicability of CLAP with the help of three software companies developing Android apps. Thus, RQ$_4$ evaluates the CLAP prototype tool as a whole, as described in Section 2.4.

TABLE 2: Objects used in our research questions.

RQ	#Apps	#Reviews	Origin
RQ$_1$	705	3,000	Randomly selected from [14]
RQ$_2$	5	200	Reviews from popular apps referring to the same app release
RQ$_3$	14	725	Selected on the basis of specific criteria from [14] and F-Droid
RQ$_4$	2	100	Reviews from two very popular apps (Facebook and Twitter)

3.2 Context Selection and Data Analysis

Table 2 summarizes the objects (*i.e.,* apps and user reviews) used in each of our research questions. To address **RQ**$_1$ we manually classified a set of 3,000 users reviews randomly selected from 705 different Android apps extracted from the dataset by Chen *et al.* [14]. In particular, two of the authors independently analyzed the 3,000 reviews by assigning each of them to a category among the seven considered in our tool. Then, they performed an open discussion to resolve any conflict and reach a consensus on the assigned category. This was needed for 268 (9%) out of the 3,000 reviews. In total, of the considered 3,000 reviews we labeled 764 as *functional bug report*, 333 as *suggestion for new feature*, 50 as *report of security issues*, 135 as *report of performance problems*, 107 as *request for usability improvements*, 106 as *report of excessive energy consumption* and 1505 as *other*. Then, we used this dataset to compute the overall average accuracy of the model by using a 10-fold cross validation. We used the WEKA's default configuration for the Random Forest classifier, *i.e.,* we set the number of trees to 100, the number of randomly chosen attributes to 0 and the maximum depth of the trees to "unlimited". We assess the overall performance of the model with its recall, precision, and F-Measure. Also, we dig into the results by presenting (i) the obtained confusion matrix, (ii) the model accuracy for each of the seven considered categories, and (iii) the Area Under the ROC curve (AUROC) [10] obtained for each category as well as for the overall model. An AUROC of 0.5 indicates a model having the same prediction accuracy in identifying true positives as a random classifier. A prefect model (*i.e.,* zero false positives and zero false negatives) has instead AUROC=1.0. Thus, the closer the AUROC to 1.0, the higher the model performances.

We also compare our approach to AR-Miner[10] [15], able to classify reviews into *informative* and *not informative*. Since the categories taken into account by the two tools are different, we performed two different comparisons: (i) we generalized CLAP, so that it categorizes reviews as *informative* and *not informative* and (ii) we specialized AR-Miner to classify reviews into the same seven categories considered by CLAP. As for the CLAP generalization, we

10. Since AR-Miner is not publicly available, we re-implemented such an approach.

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

considered the reviews belonging to the category *other* as *not informative*, and all the other reviews as *informative*. We specialized AR-MINER using all the seven categories used by CLAP in the Expectation Maximization for Naive Bayes (EMNB) algorithm, exploited by AR-MINER to filter reviews. Indeed, such an algorithm can be used to classify documents in an arbitrary number of categories.

For $\mathbf{RQ}_2$ we manually collected a second set of 200 user reviews among five Android apps, *i.e.,*FACEBOOK, TWITTER, YAHOO MOBILE CLIENT, VIBER, and WHATSAPP. For this research question we have selected very popular apps since we needed to collect from each app a good number of reviews (i) related to the same app's release, and (ii) belonging to the *functional bug report* or to the *suggestion for new feature* category. We randomly selected from each of these apps 40 reviews, 20 *bug reports* and 20 *suggestions for new features*, referring to the same app's release[11]. Then, we asked three industrial developers having over five years of experience each to manually clustering together the set of reviews belonging to the same category (*e.g., functional bug report*) in each app. We clearly explained to the developers that the goal was to obtain clusters of reviews referring to the same bugs to be fixed or feature to be implemented.

The three developers independently analyzed each of the 200 reviews to cluster them. After that, they reviewed together their individual clustering results and provided us a single "oracle" reflecting their overall point of view of the existing clusters of reviews. Once obtained the oracle, we used CLAP, and in particular the process detailed in Section 2.2, to cluster together the same sets of reviews.

As previously explained, to apply the DBSCAN clustering algorithm we need to tune its ϵ parameter. We performed such a tuning by running the DBSCAN algorithm on the YAHOO app varying ϵ between 0.1 and 0.9 at steps of 0.1 (*i.e.,* nine different configurations). Note that it does not make sense to run DBSCAN with $\epsilon = 0.0$ or $\epsilon = 1.0$ since the output would trivially be a set of singleton clusters in the former case and a single cluster with all reviews in the second case. To define the best configuration among the nine tested ones, we measured the similarity between the two partitions of reviews (*i.e.,* the oracle and the one produced by CLAP) by using the MoJo eFfectiveness Measure (MoJoFM) [41], a normalized variant of the MoJo distance computed as follows:

$$MoJoFM(A,B) = 100 - (\frac{mno(A,B)}{max(mno(\forall E_A, B))} \times 100)$$

where $mno(A,B)$ is based on the minimum number of *Move* or *Join* operations one needs to perform in order to transform a partition A into a partition B, and $max(mno(\forall\ E_A, B)$ is the maximum possible distance of any partition A from the partition B. Thus, $MoJoFM$ returns 0 if partition A is the farthest partition away from B; it returns 100 if A is exactly equal to B. The results of this tuning are shown in Figure 7. In general, values between 0.5 and 0.7 allows to achieve good performances, with the highest MoJoFM reached at 0.6. This is the default value in CLAP, and thus the one we will use in the evaluation.

11. As previously said, in CLAP we are not interested in clustering reviews belonging to the *other* category, as well as the ones belonging to the non-functional categories.

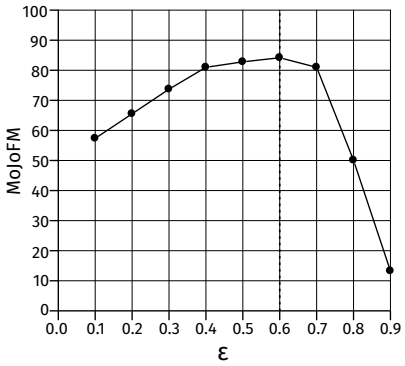

Fig. 7: Tuning of the ϵ DBSCAN parameter.

In order to evaluate the CLAP's clustering step, we measured the MoJoFM distance on reviews of the remaining apps (*i.e.,* excluding YAHOO). Also in this case, we compare the results achieved by CLAP to the results achieved by AR-MINER. Such a tool uses LDA to divide the reviews into topics. Such an approach requires to specify the number of topics (k) to extract from the given set of reviews. To experiment with this approach in its most favorable scenario, we set k equals to the number of clusters manually defined by the developers for each set of reviews we considered in our study. For example, when we clustered reviews from Twitter (category *functional bug report*), we set $k = 13$, because the developers clustered such reviews into 13 groups; on the other hand, we used $k = 11$ for the category *suggestion for new feature* of the app Facebook, because such reviews were clustered into 11 groups. Thus, the approach behind AR-MINER has been experimented with an optimal setting of the k parameter.

To answer $\mathbf{RQ}_3$ we exploited the Android user reviews dataset made available by Chen *et al.* [14]. This dataset reports user reviews for multiple releases of ∼21K apps, showing for each review: (i) the date in which it has been posted, (ii) the app's release it refers to, (iii) the user who posted it, (iv) the hardware device exploited by the user, (v) the rating, and (vi) the textual content of the review itself. In addition, each app in the dataset is associated to a metadata file containing its basic information, including the "updated" optional field that app's developers can use to describe the changes they made to the different app's releases (*i.e.,* a sort of release note shown in the Google Play store). We exploited such a dataset to build, for a given app, an oracle reporting which of the reviews left by its users for the release r_i have been implemented by the developers in the release r_{i+1} (*i.e., high priority* reviews) and which, instead, have been ignored/postponed (*i.e., low priority* reviews). To reach such an objective, firstly we identified the apps in the dataset having all the information/characteristics required to build the oracle:

1. A non-empty "updated" field containing at least one English word. As said before, this is an optional field where the app developers can report the changes they applied in a specific app's review. This first filtering was automated by looking for the "updated" fields matching at least one term (excluding articles) in the Mac OS X English dictionary. This left us with ∼11K apps.

2. Explicitly reporting the app's version to which the "updated"

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

field refers. Often developers simply put in the "updated" field the changes applied to the last release of the app without specifying the "release number" (*e.g.,* release 2.1). This is an information needed to build our oracle. Indeed, starting from the release note (*i.e.,* the content of the updated field) of a specific release r_{i+1}, we have to look at the reviews left by users of the release r_i to verify which of them have been actually implemented by the developers. We adopt regular expressions (*e.g.,* `version | release | v | r` followed by at least two numbers separated by a dot) to automatically identify apps reporting the release number in the "updated" field. This left us with ∼1.4K apps.

3. Having a non-ambiguous release note (update field). Release notes only containing sentences like "fixed several bugs" or "this release brings several improvements and new features" are not detailed enough to understand which of the user reviews have been implemented by developers. For this reason, one of the authors manually looked into each of these 1.4K apps for those containing a non-ambiguous release note. This selection led to only 73 apps remaining.

4. Having available at least 30 reviews for the release r_i preceding the r_{i+1} described in the release note. The dataset by Chen *et al.* does not report reviews for all releases of an app. Thus, it is possible that the reviews for r_i are not available or are too few for observing something interesting. This further selection process led to the five apps considered in our study: `barebones 3.1.0`, `hmbtned 4.0.0`, `timeriffic 1.11`, `ebay 2.6.1`, and `viber 4.3.1`.

The five selected app releases received a total of 18,591 user reviews from the r_i releases to be labeled as "implemented" or "not implemented" in r_{i+1}. Since manually labeling all of them would not be feasible in a reasonable time, a a statistically significant random sample of reviews with 95% confidence level and 5% confidence interval was extracted. This resulted in 463 reviews that were manually analyzed, and labeled as implemented/not implemented on the basis of the information contained in the related release note. Also in this case, conflicts (raised for 37 reviews) were solved with an open discussion.

Given the low number of apps suitable for our study that we were able to identify in the dataset by Chen *et al.* [14], we looked on F-Droid[12], a forge of open source apps, for additional apps suitable for our study. Overall, we analyzed 30 apps: We mined their Google Play page as well as the commit history from their repository selecting the apps satisfying the following requirements:

1. Having a repository with tagged releases. We excluded all the apps do not tagging commits with the app's release they refer to. This was necessary to identify the sequence of commits (changes) performed by developers while working on a release r_{i+1}. Indeed, as detailed later, we used this information to verify which of the users' reviews posted for r_i was actually implemented by developers in r_{i+1}.

2. Having at least one release r_i having at least 30 reviews on the Google Play store. This was needed to have enough reviews on which perform clustering and prioritization. Versions of apps having less than 30 reviews were automatically discarded.

TABLE 3: Total number of clusters and high/low-priority clusters for each category used to answer $\mathbf{RQ}_3$.

Category	Total number	High-priority	Low priority
functional bug report	113	13	97
suggestion for new feature	74	11	60
report of security issues	5	0	4
report of performance problems	6	1	5
report of excessive energy consumption	2	0	2
request for usability improvements	7	2	5
Total	200	27	173

With these two selection criteria, we collected the following nine apps: `ifixit 1.3.1`, `duckduckgo 2.1.1`, `zxing 4.6.2`, `ringdroid 2.5`, `boinc 7.2.7`, `dolphin 0.11`, `wordpress 2.7.1`, `2048 1.8` and `linphone 1.0.16`. Then, we cloned the *git* repository of each app, and one of the authors manually analyzed the sequence of commits going from the tagged version r_i to the tagged version r_{i+1} to check which of the requests from r_i's users' reviews were implemented in r_{i+1}. In this way, each of the reviews belonging to the ten considered app's releases was tagged as "implemented" or "not implemented" (as previously done for the five apps from the dataset by Chen *et al.* [14]).

The study is conducted in a scenario in which the categorization of the reviews into *functional bug report*, *suggestion for new feature*, *report of performance problems*, *report of security issues*, *report of excessive energy consumption*, *request for usability improvements* and *other*, as well as the result of the clustering step has been manually checked (and fixed, when needed) by the developer. To support such a scenario, also in this case two authors (i) categorized the reviews manually labeled as implemented/not implemented in one of the six informative categories (reviews belonging to the category *other* were excluded), and (ii) manually clustered them. Table 3 shows the number of clusters for each category, and how many clusters were classified as high-priority or low-priority.

Once built the oracle for the 14 apps, we assess the ability of CLAP, and in particular of the prioritization step described in Section 2.3, to correctly identify the clusters of reviews that should be prioritized in sight of the next app's release (*i.e.,* those that have been actually implemented by developers in the r_{i+1} reviews). To achieve this goal, we perform a ten-fold validation on the set of clusters manually defined. Given the unbalanced distribution of *high priority* and *low priority* clusters (*i.e.,* 27 *vs* 173 totally), at each iteration of the ten-fold validation the training-set was balanced using the SMOTE filter [13], which produced artificial instances of *low priority* clusters. To avoid any bias, no changes were applied to the test-set. We use the same metrics employed in RQ_1 to assess the CLAP prioritization accuracy.

Finally, to answer $\mathbf{RQ}_4$ we conducted semi-structured interviews with the project managers of three software companies developing Android apps[13]. Before the interviews, one of the authors showed a demo of CLAP, and let the participant interact with the tool. To avoid biases and evaluate the tool with a consistent set of reviews, all project managers

12. https://f-droid.org

13. RQ_4's participants were not the same involved in RQ_2.

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

worked with a version of CLAP having the reviews for TWITTER and FACEBOOK imported. Note that, using the reviews of the apps developed by the three companies was not an option, since most of the reviews they receive are not in English and the current implementation of CLAP only supports such a language. The interviews lasted for two hours with each company. Each interview was based on the think-aloud strategy. Specifically, we showed all the tool features to the managers to get qualitative feedback on both the tool and the underlying approach. In addition, we explicitly asked the following questions:

Usefulness of reviews. Do you analyze user reviews when planning a new release of your apps?

Factors considered for the prioritization phase. Are the factors considered by CLAP reasonable and sufficient for the prioritization of bugs and new features?

Review categories. Is the categorization of reviews into the seven considered categories sufficient for release planning or there are other categories that should be taken into account?

Tool usefulness. Would you use the tool for planning new releases of your apps?

Participants answered each question using a score on a four-point Likert scale: 1=absolutely no, 2=no, 3=yes, 4=absolutely yes. The interviews were conducted by one of the authors, who annotated the provided answers as well as additional insights about the CLAP's strengths and weaknesses that emerged during the interviews[14].

3.2.1 Replication Package

The material used in our studies along with its working data set is publicly available in our replication package [36]. In particular, we include:

- RQ_1: The training sets and the test sets for the 3,000 user reviews. The sets are provided in ARFF format, to be run with the WEKA implementation of the Random Forest to replicate our results.
- RQ_1: The categorization accuracy obtained by CLAP using different machine learning algorithms (*i.e.,* Random Forest, Rotation Forest, J48, Bayesian Network, Simple Cart, and SMO, a support vector classifier)).
- RQ_2: The 160 user reviews classified as functional bug reporting/suggestion for new features used to evaluate the CLAP clustering feature.
- RQ_2: The clustering oracles manually build by the three industrial developers (includes a README file explaining how to interpret them).
- RQ_3: The list of implemented/not implemented reviews for the 14 considered apps.
- RQ_3: The ten training sets and test sets for the 207 prioritized clusters. The sets are provided in `arff` format, to be run with the WEKA implementation of the Random Forest to replicate our results.
- A `csv` file reporting the mapping from the apps' categories in Google Play to our 10 macro-categories (see Section 2.1).

14. Note that the interviews reported in this paper are updated versions of the ones from our ICSE paper [40]: We contacted again the project managers, showed them the new CLAP version, and collected their new thoughts.

4 STUDY RESULTS

This section reports the analysis of the results for the four research questions formulated in Section 3.1.

4.1 RQ_1: How accurate is CLAP in classifying user reviews in the considered categories?

Table 4 reports the Recall, Precision, F-Measure and AUROC achieved by CLAP and AR-MINER when categorizing the reviews as *informative* and *not informative*. In particular, we measured such metrics for each category as well as for the overall model for both the approaches. It is clear that both approaches perform very well, with CLAP achieving slightly better performance than AR-MINER for all the considered metrics (+3% in terms of Precision, Recall and F-Measure and +0.02 in terms of AUROC). It is worth noting that, even if the improvement is minimal in absolute terms, it is very hard to obtain any type of improvement starting from a such high accuracy (∼90%). This confirms that state-of-the-art categorization techniques are very good in recognizing informative and not informative user reviews.

Table 5 reports the Recall, Precision, F-Measure and AUROC achieved by CLAP and AR-MINER when classifying user reviews *in the seven categories considered by our approach*. Also in this case, we measured such metrics for each category as well as for the overall model for both the compared approaches.

The average F-Measure achieved by CLAP across the seven categories is 86%, with an AUROC of 0.96. The results show that the automatic classification of reviews belonging to some categories, such as *functional bug report*, *report of excessive energy consumption* and *other*, can be very accurate, with an F-Measure higher than 80%. On the other hand, for some categories, such as *request for usability improvements*, *report of security issues* and *report of performance problems* the F-Measure achieved by the classifier is lower than 70%. This shows that the model experiences more difficulties in correctly discriminate these types of reviews, likely because it is difficult to identify a set of words/n-grams characterizing them (as a sort of fingerprint). As an example, reviews indicating a *request for usability improvements* often contain words like *unusable* or n-grams like *difficult to use*. However, some of these reviews only contain a generic description of the problem, which can be attributed to a usability issue. Let us take into account the following example: *With the latest update I'm now able to sign into the app on my Galaxy S4 and it is considerably more stable and responsive on my Nook HD+, but unfortunately now when I try to read comics, the pages only take up about 2/3 of the screen available [...]*. This review does not contain any word or n-gram that could help CLAP in discerning that it belongs to the *request for usability improvements* category. Despite these errors, it is worth noticing that the AUROC value is always higher or equal than 0.90, showing a prediction accuracy quite far from a random classifier.

As compared to AR-MINER, CLAP achieves a higher AUROC for all the categories. Also, our tool always achieves a higher F-Measure, with the only exception of the *report of security issues* category. Overall, CLAP achieves a +4% in terms of F-Measure as compared to AR-MINER.

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

TABLE 4: RQ_1: Classification Accuracy of Reviews (generalized CLAP).

Category	Precision		Recall		F-Measure		AUROC	
	CLAP	AR-MINER	CLAP	AR-MINER	CLAP	AR-MINER	CLAP	AR-MINER
Informative	90%	87%	90%	92%	92%	89%	0.97	0.95
Not informative	93%	92%	93%	86%	92%	89%	0.97	0.95
Overall	**92%**	**89%**	**92%**	**89%**	**92%**	**89%**	**0.97**	**0.95**

TABLE 5: RQ_1: Classification Accuracy of Reviews (specialized AR-MINER).

Category	Precision		Recall		F-Measure		AUROC	
	CLAP	AR-MINER	CLAP	AR-MINER	CLAP	AR-MINER	CLAP	AR-MINER
functional bug report	91%	80%	90%	91%	90%	85%	0.97	0.95
suggestion for new feature	63%	61%	84%	77%	72%	68%	0.94	0.92
report of security issues	65%	100%	66%	52%	65%	68%	0.93	0.87
report of performance problems	79%	65%	52%	36%	63%	47%	0.90	0.78
report of excessive energy consumption	76%	82%	86%	75%	81%	78%	0.97	0.94
request for usability improvements	85%	59%	47%	37%	60%	46%	0.90	0.79
other	92%	91%	91%	87%	91%	89%	0.97	0.94
Overall	**87%**	**82%**	**86%**	**82%**	**86%**	**82%**	**0.96**	**0.93**

TABLE 6: RQ_1: Confusion Matrix for CLAP.

	functional bug report	*sugg. new feature*	*security related*	*performance related*	*energy related*	*usability related*	*other*
functional bug report	**684**	33	1	7	6	3	30
sugg. new feature	9	**280**	0	3	3	2	36
security related	6	2	**33**	0	0	0	9
performance related	24	16	2	**70**	4	1	18
energy related	1	3	1	0	**91**	0	10
usability related	9	25	2	0	1	**50**	20
other	19	85	12	9	14	3	**1363**

4.1.1 *Studying Factors Influencing* CLAP*'s Classification Accuracy*

The results discussed above for CLAP are the ones achieved using the full categorization approach described in Section 2. When designing the approach, we conjectured that the low frequency of some types of reviews, such as *report of security issues*, could result in a low automatic categorization accuracy for such categories. Therefore, we use the SMOTE filter to balance the training sets in the 10-fold cross-validation. To check the real effectiveness of such a step, we performed a comparison between the categorization approach with and without SMOTE. The overall F-Measure achieved by the two versions of the approach is very similar (85% without SMOTE and 86% with SMOTE). However, if we consider the single categories, SMOTE is helpful for improving the categorization accuracy. Specifically, with SMOTE we improve the F-Measure for *report of performance problems* and *request for usability improvements* by 8% each. For the other categories, the achieved F-Measures is almost the same ($\pm 1\%$). Surprisingly, SMOTE is not helpful for *report of security issues*, which is the least represented category (only 50 review).

We also investigated to what extent the quantity of available training data influences the categorization step. To do this, we tested our approach training the classifier with an increasing number of training instances. Specifically, we randomly split the 3,000 reviews considered in our dataset into ten folds (300 reviews per fold). We used each fold, in turn, as test set and the nine remaining fold as the training set. Thus, the training set is composed at each iteration by 2,700 reviews. Then, at each iteration, we trained the model only on a subset of reviews from the training set, going from 10% to 100% at steps of 10%. This means that we experiment our approach with a training set size going from 270 to 2,700 at steps of 270 (*i.e.*, 270, 540, 810, *etc.*). We computed the overall F-Measure achieved by the classifier when using the training sets of different size. Figure 8 shows the results of this experiment (note that the y-axis starts at 80% for the sake of legibility). Using just 10% of the training instances (*i.e.*, 270 reviews) it is possible to achieve 80% of F-Measure; also, with 60% of the training instances (*i.e.*, 1,620 reviews) it is possible to achieve an F-Measure not far from the one achieved using the whole training set ($\sim 86\%$).

Table 6 reports the confusion matrix obtained by CLAP. The three most frequent cases of misclassification occurred in our evaluation are:

- Classifying *other* as *suggestion for new feature* (86 instances). These errors are often due to specific corner cases in which there are misleading words (*e.g., "Excellent set of features added in this release"*) or in which negation handling is not sufficient (*e.g., "The app lacked of [...] but now it is ok"*).
- Classifying *suggestion for new feature* and *functional bug report* as *other* (30 and 36 instances, respectively). These errors occur when the review does not contain any word/n-gram symptomatic of its category (*e.g., "Time wasted trying to send a new message"*).
- Classifying *functional bug report* as *suggestion for new feature* (33 instances). These errors are often caused by reviews particularly misleading for CLAP (*e.g., "[...] Please add a check to the date field"*). The "check to the date field" is seen as a feature to (Please) add by CLAP, thus leading to a misclassification.

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

TABLE 7: RQ_2: MoJoFM achieved by CLAP and AR-MINER.

		Facebook	Twitter	Viber	Whatsapp	**Average**
CLAP	func. bug report	78%	63%	71%	89%	**75%**
	sugg. new feature	78%	78%	82%	94%	**83%**
AR-MINER	func. bug report	59%	47%	43%	72%	**55%**
	sugg. new feature	31%	41%	52%	47%	**43%**

TABLE 8: RQ_3: Prioritization accuracy.

	correctly classified	false positive	false negative
functional bug report	90%	6%	5%
sugg. new feature	91%	3%	6%
non-functional	79%	10%	10%

4.2 RQ_2: How meaningful are the clusters of reviews generated by CLAP?

Table 7 shows the MoJoFM between the clusters of reviews manually defined by developers and those resulting from the clustering step of CLAP and AR-MINER. The average MoJoFM of CLAP is 75% when clustering reviews reporting functional bugs, and 83% for what concerns reviews recommending new features, suggesting a high similarity between manually- and automatically-created clusters.

Moving to AR-MINER, it clusters the reviews in groups substantially different from the ones manually created, achieving an average MoJoFM of 55% when clustering reviews reporting functional bugs and 43% for reviews recommending new features. While in the categorization step the two approaches achieved pretty similar results, CLAP performs significantly better in clustering reviews.

In order to give a better idea of the meaning of such MoJoFM values, Figure 9 shows the two partitions of reviews manually-created by the developers involved in the study (left side) and automatically generated by CLAP (right side) for the 20 Whatsapp reviews reporting functional bugs. The points in light grey represent the 13 reviews considered both by developers and by CLAP as singleton clusters (*i.e.*, each of these reviews recommended a different feature). The points in black represent, instead, the reviews clustered by developers into two non-singleton clusters, depicted with different colors in Figure 9. The first cluster (the green one) is exactly the same in the oracle and in the clusters generated by CLAP. The yellow cluster is similar between the two partitions. However, in the automatically generated partition it does not include the review *R5*, isolated as a singleton cluster by CLAP. This difference causes the MojoFM to be valued 89% (100% is the value for two identical partitions).

The example reported in Figure 9 provides an idea about the errors made by CLAP in clustering related reviews. We observed as it tends to be more conservative in clustering the reviews with respect to the manually produced oracle (*i.e.*, it generates more singleton clusters). While this could suggest a wrong calibration of the ϵ parameter, we also replicated this study with $\epsilon = 0.7$, $\epsilon = 0.8$, and $\epsilon = 0.9$ since higher values of ϵ should promote the merging of related reviews. However, these settings resulted in lower values of the MoJoFM across all experimented systems, due to a too aggressive merging of reviews.

4.3 RQ_3: How accurate is the reviews prioritization recommended by CLAP?

Table 8 reports the accuracy achieved by CLAP in classifying as *high priority* or *low priority* clusters of *functional bug report*, *suggestion for new feature* and *non-functional* reviews (*i.e., report of security issues*, *report of performance problems*, *report of excessive energy consumption* and *request for usability improvements*). False positives are clusters wrongly classified by CLAP as *high priority*, while false negatives are clusters wrongly classified as *low priority*.

CLAP correctly prioritizes 90% of clusters containing *functional bug report* reviews (97 out of 108), producing six ($\sim$ 6%) false positives and five ($\sim$ 5%) false negatives. The accuracy is slightly higher when prioritizing new features to be implemented, with 91% of correctly classified clusters (63 out of 70), three false positives ($\sim$ 3%) and four false negatives ($\sim$ 6%). Finally, the classification of clusters having reviews reporting issues with *non-functional* requirements is also high (79%). The overall AUROC achieved by CLAP as for prioritization is 0.775.

For example, a *functional bug report* cluster correctly *highly* prioritized by CLAP is the one from the `ebay` app, in which 141 different users using a total of nine different hardware devices were pointing out a bug present in the release 2.6.0 that prevented the app user to visualize the seller's feedbacks. This cluster also had a very low average rating ($\overline{rating}$ = 2.2), much lower that the average app rating ($\Delta rating_{app}$ = -1.9). The `ebay` developers fixed this bug in the release 2.6.1, reporting in the release note: "*Fixed bug where seller feedback would not load*".

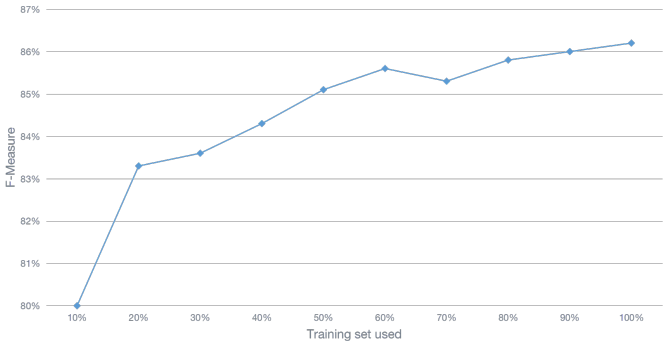

Fig. 8: RQ_2: Categorization F-Measure with an increasing percentage of training data.

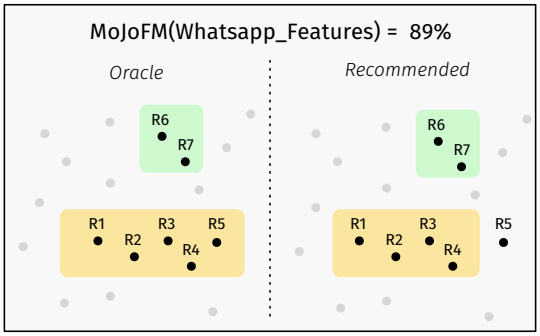

Fig. 9: RQ_2: CLAP *vs* oracle when clustering suggestions for new features on Whatsapp.

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

A false negative generated by CLAP when prioritizing clusters reporting suggestions for new features is a singleton cluster from the `barebones` app, a lightweight mobile browser. One of the users reviewing the release 3.0 assigned five stars to the app and asked for the implementation of search suggestions ("*I wish it can have search suggestions in the search bar*"). Despite the single user requiring such a feature and the high rating she assigned to the app, the `barebones` developers implemented search suggestions in the release 3.1: "*Added Google Search Suggestions*". The characteristics of this cluster led CLAP to a misclassification, since the decision trees generated in the prioritization step tend to assign a *high priority* to clusters having high values for $|reviews|$ and $|devices|$, and low values for $\overline{rating}$ and $\Delta rating_{app}$ (see the example in Figure 3). Note that these classification trees are the results of the training performed on the 14 considered apps. In a real scenario, the CLAP user can explicitly indicate which clusters she is going to implement, allowing the machine learner to adapt the classification rules on the basis of the user feedback.

To further assess the prioritization performance of CLAP, we compared it with the prioritization performed by AR-MINER [15]. Both techniques aim at prioritizing groups of reviews based on their "importance" for developers (*i.e.,* their relevance when working on a subsequent release).

The prioritization applied by AR-MINER focuses on the reviews classified as *informative* and it is based on a weighted sum of three factors: (i) the number of reviews in the group (cluster), (ii) the average rating of the reviews in the group, and (iii) the temporal distribution of reviews (more recent reviews are considered more important). Since AR-MINER is not available, we reimplemented its prioritization feature, and tuned the weighting parameters as reported in [15]. Then, we applied AR-MINER on the same set of clusters prioritized by CLAP. In particular, we considered as *informative*, the reviews not falling in the *other* category and as clusters to prioritize those manually defined (*i.e.,* exactly the same clusters prioritized in this evaluation by CLAP). Then, we compare the Area Under the Curve (AUC) for both techniques. We use AUC as AR-MINER produces a ranked list whereas CLAP produces a classification, hence it is not possible to directly compare precision and recall values.

Both CLAP and AR-MINER obtain an average AUC of 0.87 when prioritizing reviews. While the prioritization performance of the two approaches is comparable, it is worth noticing that CLAP (i) separately prioritizes reviews belonging to different categories, thus providing additional information to the developer (this was not done in this comparison in order to compute the AUC for the two techniques on exactly the same set of reviews' clusters/groups), and (ii) exploits a machine learner to prioritize clusters of reviews. This latter characteristic, combined with the feedback mechanism implemented in our tool (see Section 2.4), allows CLAP to learn from the developer her prioritization preferences. AR-MINER, instead, prioritizes groups of reviews on the basis of a fixed weighted sum of prioritization factors.

4.3.1 Studying Factors Influencing CLAP's Prioritization Accuracy

As we did for the CLAP categorization step, we studied the impact of the SMOTE filter also on the clusters' prioritization. In this case, we conjectured that the low frequency of high priority clusters could result in a suboptimal prioritization. Thus, we compared the prioritization approach with and without the use of the SMOTE filter. The overall accuracy achieved by the two versions of the approach is similar (+1% without SMOTE), but the recall achieved by our approach (*i.e.,* its ability to correctly identify high-priority clusters) is much higher when using the SMOTE filter ($+8\%$ for *functional bug report*, $+18\%$ for *suggestion for new feature*, no changes for non-functional clusters). In this prioritization step, given the similar accuracy obtained by the configuration with/without SMOTE, we believe that the configuration using SMOTE should be preferred, since it misses less important clusters of reviews that must be considered by the developer when working on the next release of her app. In other words, we prefer to have more false positives (*i.e.,* low-priority clusters classified as high-priority) than true negatives (*i.e.,* high-priority clusters classified as low-priority), to minimize the number of high-priority clusters missed by the approach.

Note that, differently from what done for the categorization step, we do not analyze the influence of the training set size on the prioritization accuracy. This is due to the limited size of data we have available in this case (207 clusters of reviews to prioritize across three different categories, *i.e.*, *functional bug report*, *sugg. new feature*, and *non-functional*). For example, we only have 17 clusters of reviews related to *non-functional* requirements. We plan to run such an analysis as part of our future work and by exploiting the clusters of reviews manually prioritized by developers using our tool.

4.4 RQ$_4$: Would actual developers of mobile applications consider exploiting CLAP for their release planning activities?

In order to answer our last research question, we qualitatively discuss the outcomes of the semi-structured interviews we conducted with project managers of three Italian companies aimed at analyzing the practical applicability of CLAP in a real development context.

Nicola Noviello, Project manager @ Next [4]. Nicola answered our first question (*i.e.,* usefulness of user reviews) with "absolutely yes", specifying that before planning a new release of an app, the developers of his company manually analyze the app reviews to identify critical bugs or feature recommendations. Nicola also confirmed that such a task is time consuming: when planning the release 2.0 of the app UNLIKELY QUOTES [6] "*A developer spent two days in analyzing more than 1,000 reviews. While the need to fix some bugs and to implement some features was easily spotted due to the fact that they were reported (required) by several users, there were also interesting features and critical bugs hidden in a large amount of non informative reviews. I strongly believe that* CLAP *would have sensibly reduced the effort we spent to identify such reviews.*" Nicola also positively answered to our questions related to the completeness of the categories of the reviews considered by CLAP and the factors it uses for prioritization ("absolutely yes" to both of them). Concerning the review categories considered by CLAP, Nicola suggested an additional category that could be considered: "*reviews referring to the app sales plan*". Nicola considers these reviews

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

"*very important*" and he explained that "*the version 2.0 of the app Unlikely quotes was released both in a free and non-free versions, with the latter introducing some features for which the users explicitly claimed (in their reviews) that they will to pay for having such functionalities.*" Thus, user reviews could not only be useful to plan *what* to implement in the next release of an app, but also to define the sale strategies. Finally, Nicola was really enthusiastic about CLAP and he will be happy to use it in his company. Indeed, he considers the tool highly usable and ready to the market. He also pointed out two possible improvements for CLAP. First, it would be useful to make the tool able to store and analyze user reviews coming from different stores (*e.g.*,, Google Play and Apple App Store): "*putting together reviews posted by users running the app on different platforms could be important to discriminate between bugs strongly related to the app from those lying in the server-side. For example, if the bug is reported by both Android and iOS users, it is very likely that the bug is in the services exploited by the app, rather than in the app itself.*" Also, Nicola suggested to integrate in CLAP a mechanism that allows to read and analyze "*the reviews of competitive apps in order to identify features for my app that are not explicitly required by my users, but that have been suggested by users of competitive apps. In other words, I do not want to listen only to my users but also the users of competitive apps!*" Clearly, this would require the implementation of techniques to automatically identify similar apps. We consider this point as part of our future work agenda.

Giuseppe Socci, Project manager @ Genialapps [2]. As well as Nicola, Giuseppe answered "absolutely yes" to our first question related to the usefulness of user reviews: "*Very often reviews are informative and useful to understand which are the directions for new releases. I usually analyze the reviews manually and such a task is really time consuming. In the first year of life of our app Credit for 3 [1], I analyzed more than 11,000 reviews, dedicating six or seven hours per week to this specific task. However, keep up with the reviews helps a lot in making the app more attractive.*" Giuseppe also answered "absolutely yes" to our second question related to the completeness of the review categories, claiming that he generally considers less fine-grained categories when analyzing the reviews (only bug reporting and suggestions for new features), but finds very interesting the ones related to non-functional requirements. Instead, Giuseppe answered "yes" to the question related to the completeness of the factors used to prioritize the bugs and the new features: "*While the exploited factors are reasonable, in my experience I also implemented several features and fixed some bugs that require few hours of work even if they were reported by just one person who is also already happy about the app. For instance, a user of the app Happy Birthday Show [5] rated the app with five stars, and requested to change the color of some buttons. Such a request required just a couple of hours of work. Thus, I decided to implement it. Considering the change impact of a new request or a bug fix might make the prioritization even more useful*". In addition, Giuseppe highlighted that the prioritization of the new features should take into account the kind of revision to perform, *i.e.*, minor or major revision: *If a major revision is planned, I tend to include as many feature requests as possible. Instead, if I am working on a minor revision, I really look for the most important feature requests to include (those having the highest payoff). In this case, the factors considered by* CLAP *in the prioritization are certainly valid.* Finally, Giuseppe answered positively ("absolutely yes") to our last question and he is willing to use CLAP as a support for the release planning of his future apps. The only showstopper for the application of CLAP in Genialapps is that most of the user reviews are written in languages different from English (*e.g.*, Spanish, Italian, French). Giuseppe already provide us this feedback when we interviewed him for our ICSE paper. While it was our plan to implement this feature, we explain in Section 6 why we did not make CLAP a multi-language tool.

Luciano Cutone, Project manager @ IdeaSoftware [3]. While Luciano considers the user reviews useful for planning new releases, in his company, in general, user reviews are not analyzed. The reason is simple. IdeaSoftware usually develops apps on commission. Thus, instead of considering user reviews, the developers of IdeaSoftware implement the features and fix the bugs required by their customers. Despite this, Luciano claimed that "*some of the features and bug fixes required by the customers of our apps were derived from the (in)formal analysis of user reviews.*". Luciano answered "absolutely yes" to the questions related to the completeness of the review category and "yes" to the factors used to prioritize the bugs and the new features. However, he also noticed that the tool could be more usable if a criticality index is provided for each feature and bug. Specifically, "*instead of having clusters classified as high and low priority, I would like to see a list of features/bugs to be implemented ranked according to a criticality index ranging between 0 (low priority) and 1 (high priority). This would provide a better support for release planning especially when the number of features/bugs classified as high priority is quite high and I do not have enough resources to implement all of them.*" Finally, Luciano claimed that the tool seems to be useful "*especially when a high number of reviews needs to be analyzed*" and he is willing to use the tool in his company for analyzing the user reviews of the apps they plan to develop for the mass market (as opposed to those they currently implemented on commission for specific customers). Luciano also suggested to capture more information on how and when feature requests and bug fixes clusters have been implemented: "*For each cluster I would like to store the version of the app in which I implemented it. In this way I can maintain in* CLAP *the revision history of my apps and I could automatically generate release notes for each version*".

5 Threats to Validity

Threats to *construct validity* are mainly related to imprecisions made when building the oracles used in the first three research questions. As explained in Section 3, the manual classifications performed for $\mathbf{RQ}_1$ and $\mathbf{RQ}_3$, as well as the golden set clusters for $\mathbf{RQ}_2$ have been performed by multiple evaluators independently, and their results discussed to converge when discrepancies occurred. Two of the authors analyzed the 3,000 reviews used in the evaluation of the CLAP categorization step with the goal of assigning each of them to a category among the seven considered in our tool. Such a classification could suffer of subjectivity bias. At least, (i) the manual analysis was performed independently by the two authors for all 3,000 reviews, and (ii) the authors performed an open discussion to resolve conflicts arisen for 268 (9%) of the 3,000 reviews.

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

TABLE 9: Accuracy achieved by different Machine Learning techniques

ML Technique	Accuracy
Random Forest	**86%**
Rotation Forest	84%
J48	83%
Simple Cart	83%
SMO	82%
Bayesan Network	81%

The three experts involved in the evaluation reported in $\mathbf{RQ}_4$, were asked to use a version of CLAP with reviews from the Facebook and Twitter apps imported. This choice was dictated by (i) the will of using a homogeneous set of reviews among all three developers; and (ii) the impossibility of using the reviews of their own apps, since most of these reviews are in Italian and in Spanish. As shown in Section 6, our attempt to implement multilingual support into CLAP failed and thus, we asked participants to analyze reviews of very well-know apps for which, at least, we are sure they have a good knowledge of the application domain.

Threats to *internal validity* concern factors internal to our study that could have influenced our findings. One threat is related to the choice of the machine learning algorithm. As explained in Section 2 we have experimented various approaches and chosen the one exhibiting the best performance. We report in Table 9 the comparison among the six machine learning techniques we experimented: Random Forest achieves the best accuracy, but other machine learning techniques, *e.g.,* Rotation Forest, also have good results. We cannot exclude that machine learners we did not consider (or different settings of the algorithm) could produce better accuracy. Similar considerations apply for the clustering algorithm. The ϵ parameter of the DBSCAN algorithm has been chosen using the tuning explained in Section 2.2.

Finally, we are aware that planning the next release is a very complex process which involve different factors. Therefore, the prioritization simply based on the factors we considered in CLAP is only a recommendation that need to be complemented by factors related to the expertise and experience of software engineers.

Threats to *external validity* concern the generalization of our findings. In the context of $\mathbf{RQ}_1$ we chose to select the 3,000 reviews from a high number of apps (705) instead that from just one or two apps to obtain a more general model. Indeed, training the machine learner on reviews of a specific app_i would likely result in a model effectively working on app_i's reviews, but exhibiting low performances when applied on other apps. To analyze to what extent the quantity of training data is important, we studied how the size of the training set influences the classification accuracy. We saw that using just 60% of the training set is sufficient to achieve an F-Measure similar to the one achieved using the complete training set. Still, while we tried to assess our approach on a relatively large and diversified set of apps, it is possible that results would not generalize to other apps, *e.g.,* those developed for other platforms such as iOS or Windows Phone. Also, as it will be further discussed in Section 6, the approach adaptation to reviews written in languages different from English does not exhibit the same performances we obtained.

6 What Did Not Work

Developing and experimenting a tool like CLAP clearly means facing a number of research challenges not always easy to overcome. What has been presented in this paper, both in terms of features implemented in CLAP as well as in the design choices made while experimenting it, does not necessarily reflect the plan we had in mind when we started the CLAP project: It is the result of compromises we had to accept on our original plan due to ideas and solutions that did not work as expected.

In this section we discuss solutions and ideas that **did not work** in the CLAP project. Our aim is to share as much as possible of this experience with the research community, thus also discussing negative results we obtained.

6.1 CLAP as a Multi-language Tool

As previously mentioned, one of the project managers we interviewed (*i.e.,* Giuseppe) while evaluating CLAP for our ICSE paper [40] expressed his interest for multi-language support in our tool. The reason for such a request was that most of the user reviews his apps receive are written in languages different from English (the only language supported by CLAP at date).

We thought that implementing multi-language support was a great idea, likely doable by relying on automatic translation tools. Indeed, we concluded the interview with Giuseppe in our ICSE paper by writing: *"We are currently adapting the tool aiming at making it multi-languages by exploiting automatic translation tools"* [40]. Indeed, such a path has been previously followed by Hayes *et al.* in the area of traceability link recovery between artifacts written in multiple languages [25].

The first author of this paper started working on this feature by using the YANDEX[15] open source library. The reason for its choice was mainly opportunistic, since with YANDEX it is possible to translate for free a good number of sentences. YANDEX was integrated in CLAP to automatically translate non-English reviews before any text-preprocessing was performed on them. This was needed in order to reuse the CLAP infrastructure aimed at manage negations, extract n-grams, *etc.* We built a new oracle to experiment the accuracy of the CLAP's reviews categorization feature: Two of the authors manually analyzed 500 reviews written in three different languages (*i.e.,* Spanish, Italian, and French) and classified them into the seven categories considered by our tool (*i.e., functional bug report*, *suggestion for new feature*, *etc.*).

When experimenting on these 500 reviews, the categorization accuracy of our tool strongly dropped down, with an F-Measure value close to 50% (as compared to the 86% obtained for English reviews). While 50% really sounds like a "random classifier", let us note that 50%, with seven possible categories considered in the classification, is much better than a random classifier (that would obtain on average an F-Measure of 14%). However, a tool with such performances is unlikely to be useful in supporting software developers, since it fails in classifying $\sim$50% of the reviews.

We thought that maybe the problem was the automatic translator we used. However, from our manual inspection

15. https://tech.yandex.com/translate/

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

of some reviews, YANDEX seems to work fairly well, by only introducing minor mistakes. However, these mistakes are likely responsible for invalidating one of the many steps behind the CLAP's text processing (*e.g.,* the removal of negated terms).

Our take away from this experience is that approaches strongly relying on natural language analysis like CLAP are not suitable to integrate automated translators to deal with multiple languages.

6.2 Cluster ranking vs. high-low priority classification

Another suggestion we got from one of the interviewed project managers (*i.e.,* Luciano) while working on our ICSE paper was to implement a prioritization index going from 0 (low priority) to 1 (high priority) instead than a "black or white" prioritization like the one implemented in CLAP (*i.e., high* or *low* priority). We considered such an option since we firstly designed the CLAP's prioritization feature. However, we were not able to define a sound and reasonable strategy to assign a so fine-grained prioritization index for the clusters of reviews of an app. Consequently, we were not able to define a way to create a training set (with fine-grained prioritization information) on which CLAP could have learned the (fine-grained) priority of clusters.

We thought to many solutions, like for example considering the number of days between the review posting and the feature/bug fixing implementation as a proxy for the cluster priority. However, we discarded this solution since temporal indicators like the number of days needed to implement a review's request are influenced by too may factors (*e.g.,* the availability of software developers to work on the app, the complexity of the implementation task, *etc.*). Thus, we preferred to rely on a simpler cluster prioritization (high *vs* low) allowing to design a more robust way to learn what a high and a low priority cluster is.

Still, we believe that Luciano's suggestion is very good and thus we are further looking into alternative solutions to this problem.

6.3 On Changes Needed due to Missing Information

In our ICSE paper [40], the cluster prioritization was based on five predictor features, as compared to the four considered in this paper. In particular, we decided to remove the feature related to *the average difference of the ratings assigned by users in the cluster who reviewed older releases of the app* ($\Delta rating_u$). A Google Play user can review multiple releases of an app over time. Clearly, her rating can change over time as a consequence of her satisfaction in using the different releases. Given a cluster C containing a set of reviews R referring to the release r_i, we compute the average difference of the ratings assigned by authors of R with respect to last rating (if any) they assigned to the releases r_x, with $x < i$. If the authors of R did not review the app before r_i, she is not considered in the computation of $\Delta rating_u$. If none of the authors of R evaluated the app in the past, $\Delta rating_u = 0$.

The reason for removing such a feature was the impossibility to compute such a difference for new reviews imported from the Google Play store. Indeed, the information about the user who posted the review is not present among the information that the developer can export from Google Play. While it is possible to find this information in reviews' datasets available online (like the ones used in this paper), thus making possible the execution of empirical studies including such a feature, this is not an option in a real usage scenario, in which the developer imports the reviews of her app. Since our goal is to make CLAP a great tool to work with, we decided to simply discard this feature from the prioritization model. Despite this removal, the overall prioritization accuracy was still very high.

7 Related Work

Several works have focused the attention on the mining of app reviews with the goal of analyzing their topics and content [20], [26], [27], [31], the correlation between rating, price, and downloads [23], and the correlation between reviews and ratings [31]. Also, crowdsourcing mechanisms have been used outside the context of mobile development for requirements engineering, for example to suggest product features for a specific domain by mining product descriptions [18], to identify problematic APIs by mining forum discussions Zhang and Hou [42], and to summarize positive and negative aspects described in user reviews [24].

Given the goal of the approach proposed in our paper, we mainly focus our discussion on approaches aimed at automatically mining requirements from app reviews.

Galvis and Winbladh [12] extract the main topics in app store reviews and the sentences representative of those topics. While such topics could certainly help app developers in capturing the mood and feelings of their users, the support provide by CLAP is wider, thanks to the automatic classification, clustering, and prioritization of reviews.

Iacob and Harrison [26] provided empirical evidence of the extent users of mobile apps rely on reviews to describe feature requests, and the topics that represent the requests. Among 3,279 reviews manually analyzed, 763 (23%) expressed feature requests. Then, linguistic rules were exploited to defined an approach, coined as MARA to automatically identify feature requests. Linguistic rules have also been recently exploited by Panichella *et al.* [33] to classify sentences in app reviews into four categories: Feature Request, Problem Discovery, Information Seeking, and Information Giving. Di Sorbo *et al.* [17] introduced SURF (Summarizer of User Reviews Feedback), a tool able to generate interactive app reviews summary highlighting the main requests made by the app's users. CLAP, differently from these techniques, also provides prioritization functionalities to help the developers in planning the new release of their app. In our classification, we only consider reviews' categories relevant to the subsequent clustering and prioritization. Gu and Kim [21] presented a review summarization framework, named SUR-MINER. Such a framework classifies reviews in five categories: (i) aspect evaluation, (ii) praises, (iii) function requests, (iv) bug reports and (v) others. To achieve this goal, the authors propose to use character n-grams and syntactical features, such as trunk words and POS tags, instead of information retrieval techniques commonly used in the literature, such as Vector Space Model [22] and bag-of-words [29] [28]. The main goal of the proposed approach is to extract information about the users' sentiment towards specific aspects of an app. The

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

final output is the visualization of such information. SUR-MINER have a different purpose than CLAP; the first, indeed, provides detailed information about the sentiment of the users, while the second automatically prioritizes the reviews to help developers planning the next release of an app.

Maalej *et al.* [29] [28] devised an approach to automatically classify reviews into bug reports, new features requests, user experiences and text ratings. In their approach, different elements are used to classify reviews, such as (i) review metadata, (ii) keyword frequency, (iii) bag of words and (iv) sentiment analysis. The main difference between such an approach and the one used in CLAP for categorization is the kind of categories taken into account. Both Maalej *et al.* [28] approach and CLAP are able to classify reviews as *functional bug report* and *suggestion for new feature*, however CLAP also provides non-functional categories, while the approach proposed by Maalej *et al.* [28] is able to identify user experiences. Furthermore, CLAP provides, in addition, features for user review clustering and prioritization.

Guzman *et al.* [22] propose a classification taxonomy for app reviews. The authors identified 7 categories, *i.e.,* (i) bug report, (ii) feature strength, (iii) feature shortcoming, (iv) user request, (v) praise, (vi) complaint, and (vii) usage scenario. The authors also propose an approach based on Vector Space Model to automatically categorize reviews. They compared the accuracy achieved by four machine learning techniques, *i.e.,* Naive Bayes, SVM, Logistic Regression and Neural Networks and they found that Neural Networks achieves the best results. Differently from CLAP, this approach does not provide clustering and prioritization of reviews.

Ciurumelea *et al.* [16] recently defined another taxonomy of 5 high-level and 12 low-level user review categories and they developed a prototype tool, named URR (User Request Reference), which has the same overall goal of CLAP, *i.e.,* improving release planning of mobile apps. URR classifies reviews in an higher number of categories, with different levels of detail, in order to return smaller sets of reviews, which are easier to handle; on the other hand, CLAP provides a clustering feature in order to reduce the complexity of analyzing many reviews about the same topic. In addition, and differently from URR, CLAP provides a prioritization of user reviews, which is a key element for release planning.

Chen *et al.* [15] pioneered the prioritization of user reviews with AR-MINER, the closest existing approach to CLAP. AR-MINER automatically filters and ranks informative reviews. Informative reviews are identified by using a semi supervised learning-based approach exploiting textual features. Once discriminated informative from non-informative reviews, AR-MINER groups them into topics and ranks the groups of reviews by priority. The main differences between AR-MINER and CLAP are:

1. Fine grained categorization vs. informative/non-informative reviews. CLAP explicitly indicates to developers the category to which each review belongs (*e.g.,* "functional bug report", "suggestion for new feature", *etc.*), while AR-MINER only discriminates between "informative" and "non-informative" reviews. Clearly, this different treatment also affects the grouping step. Indeed, while in AR-MINER a specific topic (*e.g.,* a topic referred to a specific app's feature) could indicate both suggestions on how to improve the feature as well as bugs reports, in CLAP the review clustering is performed separately between the different review categories.

2. Recommending next release features/fixes vs. ranking reviews. CLAP exploits a machine learner to prioritize the clusters to be implemented in the next app release. This allows our approach to learn from the actual decisions made by developers over the change history of their app. On the opposite, AR-MINER ranks the importance of reviews based on a prioritization score, *i.e.,* a weighted sum of "prioritization factors". Since CLAP recommends reviews to be addressed in the next release based on the past history, it would be able to weigh different features of the prediction model differently for different apps and in general for different contexts. Finally, the reviews classification performed in the previous step permits the use of different prioritization models for different kinds of change requests.

In summary, CLAP represents, to the best of our knowledge, the first approach and available tool to provide, at the same time, (i) a fine-grained categorization of mobile apps' user reviews, (ii) the clustering of related reviews, and (iii) the prioritization of suggestions with respect to future releases.

8 Conclusion and Future Work

This paper described CLAP, a tool supporting the release planning activity of mobile apps by mining information from user reviews. The evaluation of CLAP highlighted its (i) high accuracy (86%) in categorizing user reviews on the basis of the contained information, (ii) ability to create meaningful clusters of related reviews (*e.g.,* those reporting the same functional bug)—∼80% of MoJoFM, (iii) accuracy (∼83%) in recommending the changes to implement in sight of the next app release, and (iv) suitability in industrial contexts, where we gathered very positive qualitative feedbacks about CLAP.

Such qualitative feedbacks will drive our future work agenda, aimed at improving CLAP with novel features and in particular: (i) the identification of similar apps in the store with the goal of mining user reviews from competitive apps; and (ii) the multi-store support. Also, we will continue our investigation for identifying solutions aimed at making CLAP a multi-language tool.

References

[1] "Credit for 3. https://itunes.apple.com/it/app/credito-per-tre-soglie-in/id376583617?mt=8."
[2] "Genial apps website. http://www.genialapps.eu/portale/."
[3] "Ideasoftware website. http://lnx.space-service.it."
[4] "Next website. http://www.nextopenspace.it/."
[5] "Sing happy birthday songs. http://happybirthdayshow.net/en/."
[6] "Unlikely quotes. https://itunes.apple.com/it/app/citazioni-improbabili-2.0/id555656654?mt=8."
[7] "Weka. http://www.cs.waikato.ac.nz/ml/weka/."
[8] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.
[9] G. Bavota, M. L. Vásquez, C. E. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "The impact of API change- and fault-proneness on the user ratings of Android Apps," *IEEE Trans. Software Eng.*, vol. 41, no. 4, pp. 384–407, 2015.
[10] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145 – 1159, 1997.
[11] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

[12] 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.
[13] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, pp. 321–357, 2002.
[14] N. Chen, S. C. Hoi, S. Li, and X. Xiao, "Simapp: A framework for detecting similar mobile applications by online kernel learning," in *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, ser. WSDM '15. ACM, 2015, pp. 305–314.
[15] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang, "AR-miner: Mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014, 2014, pp. 767–778.
[16] A. Ciurumelea, A. Schaufelbühl, S. Panichella, and H. Gall, "Analyzing reviews and code of mobile apps for better release planning," in *24th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER'17)*, 2017, p. to appear.
[17] A. Di Sorbo, S. Panichella, C. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in my apps? summarizing app reviews for recommending software changes," in *Proceedings of the 24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, ser. FSE 2016, 2016, p. To appear.
[18] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhordi, "On-demand feature recommendations derived from mining public product descriptions," in *33rd IEEE/ACM International Conference on Software Engineering (ICSE'11)*, 2011, pp. 181–190.
[19] M. Ester, H. Kriegel, J. S, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 1996, pp. 226–231.
[20] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh, "Why people hate your app: Making sense of user feedback in a mobile app store," in *19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013, pp. 1276–1284.
[21] X. Gu and S. Kim, ""what parts of your apps are loved by users?" (t)," in *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ser. ASE '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 760–770. [Online]. Available: http://dx.doi.org/10.1109/ASE.2015.57
[22] E. Guzman, M. El-Haliby, and B. Bruegge, "Ensemble methods for app review classification: An approach for software evolution (n)," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015, pp. 771–776.
[23] 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.
[24] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pp. 168–177.
[25] J. Huffman Hayes, H. Sultanov, W. Kong, and W. Li, "Software verification and validation research laboratory (SVVRL) of the university of kentucky: traceability challenge 2011: language translation," in *TEFSE'11, Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering, May 23, 2011, Waikiki, Honolulu, HI, USA*, 2011, pp. 50–53.
[26] 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.
[27] 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.
[28] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, "On the automatic classification of app reviews," *Requirements Engineering*, vol. 21, no. 3, pp. 311–331, 2016. [Online]. Available: http://dx.doi.org/10.1007/s00766-016-0251-9
[29] 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*. IEEE, 2015, pp. 116–125.
[30] G. A. Miller, "WordNet: A lexical database for English," vol. 38, no. 11, pp. 39–41, 1995.
[31] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *21st IEEE International Requirements Engineering Conference*, 2013, pp. 125–134.
[32] F. Palomba, M. Linares-Vásquez, 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 *Proceedings of the 31st International Conference on Software Maintenance and Evolution*, ser. ICSME 2015, 2015, p. To appear.
[33] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "How can i improve my app? classifying user reviews for software maintenance and evolution," in *Proceedings of the 31st International Conference on Software Maintenance and Evolution*, ser. ICSME 2015, 2015, pp. 281–290.
[34] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.
[35] I. J. M. Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. E. Hassan, "Impact of ad libraries on ratings of android mobile apps," *IEEE Software*, vol. 31, no. 6, pp. 86–92, 2014.
[36] S. Scalabrino, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta, "Replication package. http://dibt.unimol.it/reports/clap."
[37] R. Socher, J. Bauer, C. D. Manning, and A. Y. Ng, "Parsing With Compositional Vector Grammars," in *ACL*, 2013.
[38] 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 *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, ser. ICSME '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 301–310.
[39] M. L. Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "API change and fault proneness: a threat to the success of Android apps," in *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*, 2013, pp. 477–487.
[40] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. D. Penta, "Release planning of mobile apps based on user reviews," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, 2016, pp. 14–24.
[41] Z. Wen and V. Tzerpos, "An effectiveness measure for software clustering algorithms," in *Proceedings of the 12th IEEE International Workshop on Program Comprehension*, 2004, pp. 194–203.
[42] Y. Zhang and D. Hou, "Extracting problematic API features from forum discussions," in *21st International Conference on Program Comprehension (ICPC'13)*, 2013, pp. 141–151.

Simone Scalabrino Simone Scalabrino received the Master's Degree in Computer Science from the University of Salerno in 2015 defending a thesis on Search Based Software Testing, advised by Prof. Andrea De Lucia. He received the Bachelor's Degree from the University of Molise in 2013, defending a thesis about source code readability, advised by Rocco Oliveto and Denys Poshyvanyk. He is currently a Ph.D. student at University of Molise. His research interests include software security, testing and quality.

Gabriele Bavota Gabriele Bavota is an Assistant Professor at the Università della Svizzera italiana (USI), Switzerland. He received the PhD degree in computer science from the University of Salerno, Italy, in 2013. His research interests include software maintenance, empirical software engineering, and mining software repository. He is author of over 70 papers appeared in international journals, conferences and workshops. He serves as a Program Co-Chair for ICPC'16, SCAM'16, and SANER'17. He also serves and has served as organizing and program committee member of international conferences in the field of software engineering, such as ICSE, ICSME, MSR, ICPC, SANER, SCAM, and others.

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TSE.2017.2759112

Barbara Russo Barbara Russo is associate professor at the Faculty of Computer Science of the Free University of Bozen-Bolzano, Italy and member of the International Software Engineering Research Network. She was visiting researcher at the Max-Planck Institut fur Mathematik in Bonn, Germany and at the University of Liverpool, United Kingdom. Since 2014, she is coordinating the research area in Software and Systems Engineering (SwSE) at the Faculty Computer Science. Since 2006, she is a member of ISERN - International Software Engineering Research Network. She has been program co-chair and PC member of international conferences and journals (EMSE, TSE, JSS, ESEJ, IS, IST, etc.). She is interested in modelling and predicting software reliability and data mining in software engineering for software quality.

Massimiliano Di Penta Massimiliano Di Penta is an associate professor at the University of Sannio, Italy. His research interests include software maintenance and evolution, mining software repositories, empirical software engineering, search-based software engineering, and service-centric software engineering. He is an author of more than 230 papers appeared in international journals, conferences, and workshops. He serves and has served in the organizing and program committees of more than 100 conferences such as ICSE, FSE, ASE, ICSM, ICPC, GECCO, MSR WCRE, and others. He has been a general co-chair of various events, including the 10th IEEE Working Conference on Source Code Analysis and Manipulation (SCAM 2010), the second International Symposium on Search-Based Software Engineering (SSBSE 2010), and the 15th Working Conference on Reverse Engineering (WCRE 2008). Also, he has been a program chair of events such as the 28th IEEE International Conference on Software Maintenance (ICSM 2012), the 21st IEEE International Conference on Program Comprehension (ICPC 2013), the ninth and 10th Working Conference on Mining Software Repository (MSR 2013 and 2012), the 13th and 14th Working Conference on Reverse Engineering (WCRE 2006 and 2007), the first International Symposium on Search-Based Software Engineering (SSBSE 2009), and other workshops. He is currently a member of the steering committee of ICSME, MSR, SSBSE, and PROMISE. Previously, he has been a steering committee member of other conferences, including ICPC, SCAM, and WCRE. He is in the editorial board of the IEEE Transactions on Software Engineering, the Empirical Software Engineering Journal edited by Springer, and of the Journal of Software: Evolution and Processes edited by Wiley.

Rocco Oliveto Rocco Oliveto is Associate Professor in the Department of Bioscience and Territory at University of Molise (Italy). He is the Chair of the Computer Science program and the Director of the Laboratory of Computer Science and Scientific Computation of the University of Molise. He received the PhD in Computer Science from University of Salerno (Italy) in 2008. His research interests include traceability management, information retrieval, software maintenance and evolution, search-based software engineering, and empirical software engineering. He is author of about 100 papers appeared in international journals, conferences and workshops. He serves and has served as organizing and program committee member of international conferences in the field of software engineering. He is a member of IEEE Computer Society and ACM.

Copyright (c) 2017 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.