Order in Chaos: Prioritizing Mobile App Reviews using Consensus Algorithms

Layan Etaiwi
Polytechnique Montréal
Montréal, Canada
mashael.etaiwi@polymtl.ca

Sylvie Hamel
Université de Montréal
Montréal, Canada
hamelsyl@iro.umontreal.ca

Yann-Gaël Guéhéneu
Concordia University
Montréal, Canada
yann-gael.gueheneuc@concordia.ca

William Flageol
Concordia University
Montréal, Canada
w_flage@encs.concordia.ca

Rodrigo Morales
Concordia University
Montréal, Canada
rodrigo.moralesalvarado@concordia.ca

***Abstract*—The continuous growth of the mobile apps industry creates a competition among apps developers. To succeed, app developers must attract and retain users. User reviews provide a wealth of information about bugs to fix and features to add and can help app developers offer high-quality apps. However, apps may receive hundreds of unstructured reviews, which makes transforming them into change requests a difficult task. Approaches exist for analyzing and extracting topics from mobile app reviews, however, prioritizing these reviews has not gained much attention. In this study, we introduce the use of a consensus algorithm to help developers prioritize user reviews for the purpose of app evolution. We evaluate the usefulness of our approach and meaningfulness of its consensus rankings on four Android apps. We compare the rankings against reviews ranked by app developers manually and show that there is a strong correlation between the two (average Kendall rank correlation coefficient = 0.516). Thus, our approach can prioritize user reviews and help developers focus their time/effort on improving their apps instead of on identifying reviews to address in the next release.**

***Index Terms*—Consensus algorithms, Rankings, Review prioritization, User reviews, Software evolution.**

I. Introduction

The mobile apps industry grew tremendously in the past few years. Apple App Store and Google Play, the two largest marketplaces for mobile apps, were launched in 2008 and host more than 5 millions apps. As of September 2019, Google Play reached a total of 3.3 million apps while Apple App Store 2.2 million apps [1].

In addition to the download service, these marketplaces allow users to rate and review the apps using a five-point Likert scale and unstructured text. These ratings and reviews are important. Beside guiding prospective users in the choice of apps to download, they provide a valuable source of information for app maintenance and development.

Harman *et al.* [2] showed that there is a high correlation between user ratings and reviews and numbers of downloads. Recent studies [3]–[5] showed that one third of the information in the user reviews could help developers maintain and develop their apps. Therefore, it is important for app developers to consider user reviews when updating their apps.

However, processing and analyzing these reviews present three challenges. First, the volume of user reviews received for some apps is extremely large and surpasses developers' ability to read them manually. For example, while WordPress app receives about 100 reviews every month [6], Facebook app gets $> 4,000$ reviews per day [7]. Second, reviews contain unstructured text that is difficult to parse and analyze [8]. Third, the portion of high-quality reviews is relatively small, only 35.1% is useful for app improvement [5].

To solve these challenges, many approaches exist to process, analyze, classify, and cluster user reviews [2], [4], [8]–[12]. For example, AR-Miner [5] tags reviews as informative and non-informative, extracts, groups, and ranks topics from reviews. CLAP [13] categorizes them into categories and clusters related reviews. URR [14] classifies reviews based on topics and sub-topics taxonomy and links reviews to source code.

These approaches could help developers identify important reviews, however their prioritization techniques are limited to either labeling them as high/normal, or ranking them based on a fixed formula. They are not flexible enough to consider every possible review attribute into the prioritization process.

Developers would benefit from a prioritization method that (1) takes into account multiple attributes and (2) finds a consensus among all reviews and their attributes to help developers plan the next releases of their apps.

In this study, we propose an approach of applying a consensus algorithm to aggregate a set of rankings of user reviews into one optimal ranking. The optimal ranking is a prioritized list of user reviews that help app developers deciding what improvements to implement in the next releases of their apps.

Figure 1 summarizes our approach. First, we collect the publicly-available user reviews used in [13]. Second, we pre-process, categorize, and cluster the reviews using CLAP [13]. Third, we define a set of attributes that are commonly used by developers to rank reviews (cardinality, oldest date, average rating, category). Fourth, we rank the clusters of reviews based on these four attributes. For example, the ranking by the oldest date is an ordered list of clusters in which the first cluster contains the review with the oldest date among all reviews. Fifth, we apply a consensus algorithm on the rankings to generate one consensus ranking of ordered clusters/reviews to address in the next release.

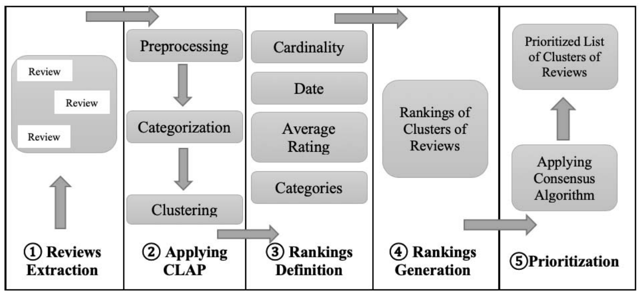

Figure 1: Overview of the Approach

We validate our approach by prioritizing user reviews of four Android apps released on Google Play. First, we compare the consensus rankings generated by our approach with manually-prioritized rankings by four app developers and statistically measure the correlation between them by computing the kendall rank correlation coefficient. Second, we perform a qualitative evaluation by inviting app developers to evaluate the results. Finally, we survey apps developers whether they would consider our approach in their next release planning. Results show that our approach can produce useful rankings and, thus, help successfully apps developers prioritizing their users reviews when working on a new release.

The rest of this paper is as follows. Section II presents related works. Section III describes a motivating example. Section IV introduces the consensus algorithms. Section V details our approach. Section VI discusses validation methods and results. Finally, we discuss threats to validity in Section VII and conclude in Section VIII.

II. Related Work

Many studies pertain to users feedback to extract features for different purposes. Harman *et al.* [2] extracted features from user feedback via data mining and reported a strong correlation between user rating and number of app downloads. Machine learning approaches were used [4], [8] to process user reviews and extract features to classify these reviews into maintenance and evolution categories. Iacob *et al.* [9] analyzed hundreds of reviews to define a set of the most common topics discussed in reviews. In addition to extracting features for classifying reviews, Ciurumelea *et al.* [14] linked classified reviews with source code to help developers find code to modify.

Studying user feedback for information retrieval has gained much attention, *e.g.,* automated information retrieval systems to query specific features or keywords in reviews [12]. Pagano and Maalej [3] performed an exploratory study and concluded that (1) numbers of reviews decrease in time, (2) most reviews focus on three main topics, and (3) positive reviews lead to high download. Fu *et al.* [10] identified reasons for users to like/dislike apps at three levels of granularity. Gebauer *et al.* [15] used structural equations to identify factors impacting user reviews.

Our approach is different than the previous ones. We focus on studying user reviews to prioritize them in a consensual way to help developers plan their apps' future releases.

Laurent *et al.* [16] proposed a semi-automated technique to prioritize software requirements using a probabilistic traceability model. Avesani *et al.* [17] introduced a case-based ranking framework for software requirements prioritization, which uses a pairwise comparison technique to identify/explain preferred requirements. Beg *et al.* [18] used B-trees to prioritize software requirements. These approaches do not apply to mobile app reviews because they (1) do not work well on a large number of reviews, (2) assume that clients participate in the prioritization process, and (3) do not perform preprocessing, categorizing, and clustering reviews activities.

Keertipati *et al.* [19] defined three user review prioritization approaches on four review attributes (frequency, rating, negative emotions, and deontics). Another approach proposed by Gao *et al.* [20] is PAID to find issues in reviews at phrase level rather than sentence level. These approaches do not take into account all possible review attributes in the prioritization process, and do not perform any review preprocessing.

AR-Miner by Chen *et al.* [5] was the first automated approach to preprocess, classify, and rank user reviews. CLAP by Scalabrino *et al.* [13] performs better than AR-Miner when categorizing and clustering reviews. In addition, CLAP prioritizes the clusters of reviews by labeling them as high/normal. Our approach differs from these two. It aggregates the results of many rankings and provides developers with one consensus ranking of ordered reviews.

III. Motivating/Running Example

To motivate our work, we use this running example: Matthew is an app developer for WordPress [6]: an Android app on Google Play, with an average rating of 4.5, over 135,435 submitted reviews, and over 10 million downloads. Matthew must regularly update his app with new features and bug fixes to maintain its high rating and user satisfaction. He needs to know what is the general opinion of users about his app, expected features, and reported issues. Therefore, he frequently reads the many user-submitted reviews on Google Play.

However, he finds himself spending too much time analyzing, and extracting information from the reviews due to the unstructured nature of these reviews. Table I shows some reviews for his app on Google Play on different dates. Clearly, the first and second reviews are somewhat useful and can help Matthew improve his app; the other two contain no useful development information.

To reduce effort, Matthew looks for an automated mobile app review analysis tool that can do the job for him. He chooses and applies CLAP on reviews of his app, which

ID	Date	Rating	Review
0	4/4/2014	5	I need the justify post feature
2	4/5/2014	4	Stats broken in last update
5	4/6/2014	2	It does not work
6	4/8/2014	3	It does not go through my self hosted blog

Table I: A few user reviews from WordPress App. v2.7.1

preprocesses, categorizes, and clusters reviews for him as illustrated in Table II.

ID	Review	Category	Cluster
0	I need the justify post feature	Feature	C1
1	More options, like text coloring	Feature	C1
2	Stats broken in last update	Bug	C10
3	Facebook shared posts show other photos	Bug	C11
4	Cannot log in	Bug	C12
7	slow and buggy	Perf.	C17
8	Unacceptably poor UI	Usability	C18
9	Wish it is easier to format text and images	Usability	C18

Table II: CLAP categories and clusters for some reviews

Matthew successfully produced preprocessed, categorized, and clustered reviews but now he needs an approach to prioritize the clusters of reviews for an effective release planning. When it comes to ranking reviews, there are different attributes to consider. For example, Matthew could rank the clusters based on the average rating, oldest review's date, or categories as illusrated in Table III.

Attribute	Ranking
Average Rating: order by average rating of each cluster ascendingly	[[C18],[C11,C17],[C10],[12],[C1]]
Oldest Date: order by the oldest submission date of the reviews in each cluster	[[C18,C1],[C10],[C12],[C11],[C17]]
Category: order by cluster categories: first bug, then performance, usability, and finally feature	[[C10,C11,C12],[C17],[C18],[C1]]

Table III: Rankings of the clusters presented in Table II

Applying different attributes, prioritizes the clusters differently, making it problematic for Matthew to identify the most pressing reviews as none of the rankings might optimally provide the best order. Thus, we propose to apply a consensus algorithm on these rankings to aggregate them into one optimal ranking that will help Matthew identify the most important reviews to address in his next releases.

In summary, the main contributions of our work are:

1) A novel approach to prioritize mobile app user reviews considering a set of review attributes.
2) An evaluation of the approach on four Android apps with four apps developers involved to prove the effectiveness of our approach in providing a prioritized list of user reviews.
3) Feature responses from an online questionnaire that investigates if app developers prioritize user reviews when working on new updates, how, and if they would consider our approach.

IV. Consensus Algorithms

A. Context

Aggregating multiple rankings into one consensus ranking is a two-century old problem [21] that gained an increasing research interest in the last two decades. The problem was first investigated in the context of voting [22]. Then, it has been addressed in many domains like bioinformatics [23], Web engines querying [24], AI [25] and others. To the best of our knowledge, aggregating rankings into one consensus ranking has not been applied in software engineering in general and mobile app evolution in particular.

B. Definitions and Measures

Finding a consensus ranking is aggregating multiple rankings, *i.e.,* ranked lists of elements, into one ranking that minimizes the pairwise disagreement between elements [26]. Rankings can be permutations, *i.e.,* strictly ordered complete rankings: the same set of elements are included in each ranking and those elements are strictly ordered. However, rankings from real-world applications might not be complete rankings, *i.e.,* might contain ties.

Many studies [23], [27] suggested the use of a generalized version of the Kendall-τ distance [28] as a dissimilarity measure when comparing two rankings with ties. $G(r, s)$ [27] computes the distance between two rankings with ties, r and s, on n elements as follows:

$$\begin{gathered} G(r,s) = \#\{(i,j) : i < j \wedge \\ ((r[i] < r[j] \wedge s[i] > s[j]) \vee (r[i] > r[j] \wedge s[i] < s[j]) \vee \textbf{(1)} \\ (r[i] \neq r[j] \wedge s[i] = s[j]) \vee (r[i] = r[j] \wedge s[i] \neq s[j]))\} \textbf{ (2)} \end{gathered}$$

which is the sum of (1) the number of times element i and j disagree in their order in the two rankings, or (2) the number of times the two elements are tied in one ranking, and not tied in the other one.

Generating a consensus ranking from a given set of rankings with ties R requires finding the ranking that has the smallest generalized Kemeny score. This score, K, is defined as:

$$K(r, \mathcal{R}) = \sum_{s \in \mathcal{R}} G(r, s).$$

An optimal consensus ranking of a set of rankings with ties $\mathcal{R}$ with the generalized Kemeny score is r^* such that:

$$K(r^*, \mathcal{R}) \leq K(r, \mathcal{R}),$$

for all $r \in \mathcal{R}_n$, where $\mathcal{R}_n$ is the set of all possible rankings with ties over n elements.

C. Implementations

In [27], 14 different consensus algorithms are studied and categorized into three approaches: algorithms that (1) handle ties and focus on the disagreements of the order of elements; (2) focus on disagreements of the order of elements, cannot handle ties, but can be modified to handle ties, (3) focus on the position of elements in the rankings, cannot deal with ties but could be modified to deal with ties.

Brancotte *et al.* [27] studied several consensus ranking algorithms, with experiments on real and synthetic, small and large, datasets. They concluded that BioConcert [23] and ExactAlgorithm [27] provide the best quality results. KwikSort algorithm [29] came second after BioConcert when the dataset is extremely large ($n > 30,000$). If execution time is a concern and the rankings do not contain many ties, then BordaCount [30] and MEDRank [31] are usable. In our datasets, the number of elements and rankings are < 100, therefore execution time is not a concern and we used BioConcert, ExactAlgorithm, and KwikSort.

V. Approach

Our approach generates a consensus ranking of a set of user reviews to help apps maintenance and evolution. We now discuss (1) the input data of our approach and the steps to prepare and process user reviews; (2) the possible attributes that can be used for ranking user reviews; and, (3) how we apply a consensus algorithm to a set of rankings to produce one consensus ranking.

A. Input

Our approach takes as input a set of user reviews. Reviews usually contain the following information: (1) title, (2) free unstructured text, (3) star rating (between 1 and 5), (4) reviewer's username, (5) date of the review, and (6) version of the reviewed app. Our approach can be applied to any size of dataset, reviews from any year, and other platforms (such as Apple App Store or BlackBerry World).

Running Example. We use the publicly-available dataset of user reviews by Scalabrino *et al.* [13], which contains 725 user reviews from Google Play for 14 Android apps. The reviews roughly cover the period from October 2010 to May 2014 and covers different app categories. The diversity of the categories ensures the variety of the reviews that are submitted by different users with different interests, which helps achieve a higher level of generalizability when evaluating our approach in the following section. Table IV provides information about the apps: (1) name, (2) category, (3) number of reviews. We decided to eliminate the data of Harvest Moon BTN app from the dataset because the app does not exist anymore.

App Name	Category	Number of Reviews
BOINC	Eduaction	30
Lightning Web Browser	Communication	27
~~Harvest moon BTN~~	~~Game~~	~~34~~
Timeriffic	Tools	25
iFixit: Repair Manual	Lifestyle	5
DuckDuckGo	Tools	17
eBay	Shopping	260
Barcode Scanner	Shopping	21
Ringdroid	Music	23
2048	Puzzle	11
Viber	Communication	108
Dolphin Emulator	Arcade	107
LinePhone	Communication	28
WordPress	Productivity	29

Table IV: List of Android apps contained in dataset

B. Preprocessing

User reviews contain useful information for app developers to maintain and evolve their apps. Yet, they come with challenges: (1) some apps receive high volume of reviews daily, (2) a single review can report more than one issue, (3) reviews include unstructured text [8], and (4) low quality, non-informative reviews are numerous [5]. As a result, preprocessing the reviews before prioritization becomes mandatory.

We performed a comparison study of five automated review analysis tools: (1) ARdoc (App Reviews Development Oriented Classier) [7], (2) URR (User Request Referencer) [14], (3) SUR-Miner (Software User Review Miner) [2], (4) AR-Miner (App Review Mining) [5], and (5) CLAP (Crowd Listener for releAse Planning) [13]. To help decide which tool we can employ to preprocess our dataset, we defined a set of questions: (1) is the tool automated? (2) does the tool apply NLP techniques (*e.g.,* tokenizing and stemming)?, (3) are non-informative reviews filtered out?, (4) does the tool categorize reviews? (5) does the tool cluster related reviews? and, (6) how accurate are the results of the tool?

We applied each tool on the dataset by Scalabrino *et al.* [13], which we use as running example. Two of the authors compared the tools results and concluded that CLAP is the only tool that answers positively all the questions. CLAP preprocesses, categorizes the reviews into eight categories (bug, feature request, performance, security, energy, usability, and other), and clusters related ones together with high accuracy.

Running Example. Table V presents a sample of the results of applying CLAP to the WordPress app reviews. Full results are available in our replication package [32].

Review	Category	Cluster
I need the justify post feature	Feature	C1
Stats broken in last update	Bug	C10
It is slow and buggy.	Performance	C17
Wish it is easier to format text and images.	Usability	C18
Some error is coming and it does not install.	Bug	C15

Table V: Results of applying CLAP to WordPress reviews

C. Rankings

In this step, we rank the clusters of reviews that are produced by CLAP according to a set of defined review attributes independently. We performed an iterative reviews analysis to identify attributes that app developers could use for ranking reviews when deciding what change requests to address in the next release. We identified four attributes: cardinality, oldest date, average rating, and category.

- **Cardinality**: clusters of reviews are ranked according to the number of reviews in each cluster decreasingly. Clusters with higher number of reviews indicate that the same issue has been reported by many users.
- **Oldest Date**: clusters of reviews are ordered chronologically (from oldest to most recent) based on the oldest date of review included in the cluster.

- **Average Rating**: we compute the average rating of all reviews in a cluster and rank clusters with lower average rating first.
- **Category**: we sent a survey to some app developers and Ph.D. students with app development experience to understand how they would rank categories. 17 developers answered as follows: bug, security, performance, energy, usability, and feature. We rank the clusters according to their ordering of the categories.

Running Example. We order the clusters of reviews from CLAP based on the four chosen attributes to generate a set of rankings. Table VI shows the results of ranking WordPress clusters of reviews and Table VII summaries the reviews in each clusters. Ties in rankings indicate that tied clusters have the same value, hence, they have an equal rank.

Attribute	Ranked Clusters
CD	[[9],[8,12],[1,4,14,18],[2,5,6,7,10,11,17,3,13,15,16]]
D	[[2],[9,1,4,14,18,5],[6,10],[12],[8,15],[11,3,13,16],[7,17]]
AR	[[16],[12],[9],[18,15,3,13],[14,11,17],[4],[5,6,10,8,7],[2,1]]
C	[[16,12,9,15,13,14,11,10,8],[17],[18],[3,4,5,6,7,2,1]]

Table VI: Ranked clusters of WordPress reviews (CD=Cardinality; D=Date; AV=Average Rating, C=Category)

Cluster	Summary	Cluster	Summary
C1	Text justify feature	C10	Stats is broken
C2	Ability to add featured image to posts	C11	Posts tab not showing posts
C3	Ability to modify multiple posts	C12	App log in issue
C4	Ability to format text	C13	Update are removing websites
C5	Ability to edit profile	C14	html tags appear when editing post
C6	Unable to upload media to posts	C15	Posts getting published
C7	Black background theme option	C16	Installing app issue
C8	Website log in issue	C17	Slow and buggy
C9	Image upload error	C18	Poor UI and navigation difficulty

Table VII: Summary of reviews in each cluster of WordPress

D. Consensus Ranking

We finally apply the consensus algorithms on the four different rankings of reviews from the previous step to produce one ranking that help developers choosing the reviews to consider for their next release. We produce one optimal consensus ranking (CR) that best agrees with all the four rankings.

We use the Rank Aggregation with Ties Web site [27] to generate the consensus rankings. As discussed in section IV, we selected the top three consensus algorithms (ExactAlgorithm, BioConcert, and Kwiksort) to generate the CR.

Table VIII presents the consensus rankings produced by each algorithm when applying them to the four rankings in Table VI for WordPress app along with the generalized Kemeny score of this consensus. The consensus rankings obtained by these methods are quite close to each other, partly due to the small number of clusters in each ranking. Consequently, we decided to adapt the results of the ExactAlgorithm in this study, since it guarantees an optimal CR, despite its complexity. We would choose BioConsert and Kwiksort for larger datasets.

Running Example. For each of the 13 Android app, we apply the ExactAlgorithm to the set of four rankings. The consensus rankings of the remaining apps are available in the replication package [32].

Consensus Algorithm	Consensus Ranking
ExactAlgorithm	Distance = 240 [[9],[12],[14,18],[8],[1,4], [10,11,3,13,15,16],[17],[2,5,6,7]]
BioConcert	Distance = 240 [[9],[12],[14,18],[8],[1,4], [10,11,3,13,15,16],[17],[2,5,6,7]]
Kwiksort	Distance = 241 [[9],[12,8],[1,4,14,18], [16,10,11,3,13,15],[17],[5,2,6,7]]

Table VIII: Consensus Rankings for WordPress, where distance represents the generalized Kemeny score between each consensus ranking and the set of 4 rankings of Table VI

VI. Validation and Results

We now evaluate the usefulness of our approach and the efficiency of the consensus ranking in prioritizing bugs and new features. We perform the evaluation on the user reviews of four android apps to answer the following research questions:

RQ1: (Performance) How effective is the consensus algorithm in prioritizing user reviews? We evaluate the performance of the consensus algorithm in prioritizing review clusters (1) quantitatively by computing the correlation between the consensus ranking and a gold ranking (GR) *i.e.,* a set of clusters manually-ranked by app developers and (2) qualitatively by asking app developers to evaluate the results of our approach.

RQ2: (Interest) Do mobile app developers prioritize user reviews and would they consider our approach when planning new releases? We address this question by conducting an online questionnaire of apps developers to understand how developers perform prioritization activity and how interested they are in our approach.

A. Quantitative Evaluation

We compare the CR with a gold ranking, *i.e.,* a ranking manually defined by app developers, quantitatively by computing their generalized Kendall-τ distance and the Kendall rank correlation coefficient; a correlation that statistically measures the association between two rankings [33].

We communicated with developers of the 13 apps to create the GR. We manually extracted developers' email addresses from the app's Google Play Web pages. We sent each developer a description of our research, a link to a Web site hosting the clusters of reviews of their apps, and a guide on how to rank the clusters as they see fit. We did not provide them with the CR to avoid any bias. Developers of only four apps responded to our invitation (BOINC, DuckDuckGo, Viber, and WordPress).

Running Example. The GR of WordPress is as follows while Table VII define the clusters. The gold rankings of the other apps are available in the replication package [32].

$$GR_{WP} = [[12],[16],[13],[8],[14],[15],[11],[10], [9],[18],[19],[6],[4],[1],[2],[3],[5],[7]].$$

We answer RQ1 through two methods. In the first method, we compute the worst possible distance between any ranking and the gold ranking by computing the generalized Kendall-τ distance between the GR and its reversed ranking. Then, we compute the distance between the CR and GR using the same measure. Finally, we compare those two distances.

Running Example. Table IX shows the worst possible distances and the generalized Kendall-τ distances between the CRs and the GRs for each app. The results suggest that the distance between the CR and GR is fairly small and far from reaching the worst distance. Despite that WordPress reports a relatively large distance, it is still 61.4% away from the worst distance.

App	Worst Distance	Actual Distance	Correlation Coefficient
WordPress	153	59	0.411
Viber	28	8	0.557
DuckDuckGo	21	7	0.514
BOINC	55	14	0.585

Table IX: Worst VS. actual distance and correlation coefficient between CR and GR.

In the second method, we compute the Kendall rank correlation coefficient for tied ranks between the two rankings of each app to measure if there is a statistically significant association between the CR and GR. The correlation is defined as [33]:

$$\frac{C-D}{\sqrt{(\frac{1}{2}n(n-1)-U)(\frac{1}{2}n(n-1)-V)}}$$

where C = number of concordant pairs, D = number of discordant pairs, n = number of elements, U = number of tied pairs in the first ranking, and V = number of tied pairs in the other ranking. The correlation coefficient is in the interval $[-1,+1]$ in which, according to Cohen *et al.* [34], values of 0.5 or greater indicate a large correlation, values between 0.5 and 0.3 a medium one, values between 0.3 and 0.1 a small one, and values below 0.1 indicate no correlation.

Running Example. Table IX shows a positively strong coefficient for BOINC, DuckDuckGo, and Viber (0.557, 0,514, 0.585, respectively). WordPress achieves a medium correlation coefficient (0.411).

RQ1: Quantitative Validation

There is a strong, positive correlation between the consensus rankings generated by our approach and the gold rankings from developers for the four apps.

B. Phase 2: Qualitative Evaluation

We consulted the same developers of the four apps to assess the performance of the CR generated by the consensus algorithm to answer RQ1 qualitatively. We shared with each developer the CR of the clusters of reviews of their apps and asked them "if your app reviews were prioritized in such a consensus ranking, do you believe this ranking would help you plan a successful release and would you plan your next release according to this ranking?" We provided developers with five options from which to choose (strongly agree, agree, neutral, disagree, and strongly disagree).

Except for WordPress, developers reported that they "strongly agree" with the question. WordPress developer answered "neither", and he stated that he has his own approach of prioritizing new update requests. However, he fully agreed to follow the consensus ranking with minor changes to adapt to his approach. Thus, the answer to RQ1 is that applying the consensus algorithm can generate effective user reviews prioritization.

RQ1: Qualitative Validation

Developers of the four apps agree that our approach generates consensus rankings that they would follow in planning their next releases.

C. Phase 3: Developer Questionnaire

We collect and analyze answers from Android apps developers via an online questionnaire to answer RQ2. We mined 300 apps from Google Play to collect developers' contact information. The mined apps varied between open source, free, and paid apps. We removed apps from the dataset if they either did not have an email address in their Developer field or the email address was a generic one, *e.g.,* `support@...`, `help@...`. We obtained 248 email addresses.

We used Google Form to design and administer the questionnaire. We sent invitations to the 248 email addresses explaining our project, the purpose of the questionnaire, instructions to participate, and a link to the Google Form. We gave developers two months to answer the questionnaire. We received a few bounce-back emails from very popular apps, *e.g.,* Facebook and Google. After two months, we received seven complete questionnaires.

The questionnaire consisted of four main parts. The first part gathered developers' background information (*i.e.,* app name, role, years of experience, education, gender, and age). The second part investigated the importance of user reviews. The third part asked developers if they categorize and cluster their apps' reviews and, if so, what techniques they used. The fourth part focused on any review prioritization approaches as well as important attributes (*e.g.,* frequency, average rating, number of devices, severity, category, date). (We used these answers to choose relevant attributes in Section V-C). We further explain our approach, and ask explicitly if developers would be interested in considering the approach when improving their apps.

Developers had mobile development experiences from 2 to 8 years and ages between 20 and 39 from different geographical locations (Europe, Mexico, and USA). 85% (6 out of 7) said that they strongly rely on user-submitted reviews for their next release. 57% (4/7) reported doing review categorization, clustering, and prioritization before planning their next release. 42% (3/7) reported to consider urgent and most frequent reviews by determining them through a manual analysis. 57% (4/7) mentioned using the attributes: cardinality, oldest date, average rating, and category. 86% (6/7) reported great interest in using our approach.

RQ2: Qualitative Validation

Developers' feedback positively answers RQ2 and supports the importance of our approach.

VII. Discussion

A. Discussion

We explain the results of the study in answer to the research questions described in Section VI and discuss their implications.

The quantitative results reported by the two validation methods are promising. When using the Kendall-τ distance to measure how close or far is the CR from the GR, there is no fixed threshold that the distance between the two rankings should reach. However, we could compute the worst possible distance between two rankings. The further and smaller the distance between the CR and GR from the worst possible distance, the closer and similar are the rankings. As demonstrated in our results, the distance between the CR and GR are far from reaching the worst distance, which indicates that the approach is able to produce reliable results close to the manual prioritization. It is also worth noting that it is expected not to have a very small generalized Kendall-τ distance between the CR and GR. That is due to the fact that we are measuring the distance between two different rankings, where the CR contains ties, while the GR contains no ties. Besides, we assume that the greater the statistical correlation coefficient between the rankings, the better the result of the consensus algorithm. Our results showed a very strong correlation coefficient between the CR and GR for three of the apps.

To further explain the medium correlation achieved by WordPress, we compared the CR of the ExactAlgorithm in Table VIII and the GR. We can notice a medium variation in the order of the clusters. To further analyze this variation, we contacted the app developer who generated the GR asking him if he believes that his approach of prioritizing the clusters of reviews is the best approach. He stated in his email that whenever there is a new update planning, there is a continuous discussion between the app developers on what should be included in the new update. Each developer has a different opinion and in some situations they never reach a consensual decision. As a result, the medium coefficient between the CR and GR does not doubt the performance of our approach but rather the personal judgment of the developer.

When addressing the qualitative validation, we tried in our study to contact more than a developers for each app to obtain a larger evaluation from different software experience perspectives. For WordPress app, we reached out to many developers, however, we received responses from only two. One of the developers had less than a year of development experience, and he was not fully knowledgeable about the prioritization process, therefore, his answer was eliminated. We received responses from only one developer for each BOINC and Viber. DuckDuckGo on the other hand, is an app owned by only one developer. The consensus agreement of all developers on the quality of the consensus rankings establishes a good evidence that our approach is useful for prioritizing user reviews, and could help developers achieve app maintenance and evolution tasks.

Answers from the questionnaire support the importance of considering user reviews during day to day app's improvement activities. More interestingly, most developers confirmed the necessity of analyzing, categorizing, clustering and prioritizing for a more efficient release planning. In addition, they expressed how having an automated tool that can perform these tasks would reduce the amount of manual work. The high interest shown in our approach through the questionnaire answers, highlights the potential of the success of the approach in real-world environment.

B. Threats to Validity

Threats to construct validity: Building gold rankings to be compared with the consensus rankings requires high level of accuracy and app development knowledge. To mitigate this threat, we involved app developers in the creation of the gold rankings. These app developers certainly provided the most accurate gold rankings, however, there is a level of subjectivity in deciding what cluster to address first. More developers for each app are to be involved in building the GR in future work.

The authors are not mobile app developers and thus the defined ranking attributes might not cover all attributes used during review prioritization. To alleviate this threat, we involved app developers in the selection of attributes through an online questionnaire. Although these attributes are inclusive, other attributes would provide other rankings that could increase the quality of our results. We will systematically study all possible attributes and their impact on our results in future work.

We favored the use of ExactAlgorithm over BioConcert and Kwiksort algorithms since it is the only exact algorithm that provides the most optimal ranking on smaller dataset. In future work, we plan to use BioConcert and Kwiksort on larger datasets since they were proven to provide high quality results in such cases [27].

Threats to internal validity: This could involve the tool selection to process, categorize and cluster the reviews as there could be a risk of producing low accuracy results. To mitigate this threat, we compared the results of the most outperforming review analysis tools, and adapted CLAP as it was proven to provide the most accurate results. Despite that, CLAP could have its own threat in the employed machine learning and clustering techniques. Therefore, we plan to implement a complete review analysis tool that merges the use of machine learning along with deep learning for higher accuracy.

Threats to external validity: We are confident that our approach can be generalized and applied to user reviews of any app store and different size of dataset. However, the approach could provide different results when the dataset is extremely large. In addition, we mitigate external validity of our work by providing a replication package [32] that provide all the data used in our approach and its study. From the original user reviews to the consensus rankings as well as the clustered reviews, attribute rankings, and gold rankings. Thus, others can confirm and reproduce our results.

VIII. Conclusion and Future Work

Mobile apps became a fundamental part of everyone's daily lives. They must adapt to their users' needs and, thus, considering user reviews during app evolution is essential. However, prioritizing user reviews to decide what to address in the next release is a complex task.

We presented an approach to provide apps developers with a set of user reviews prioritized based on a consensus ranking. Our approach first clusters related reviews together using a previous work, CLAP. Then, it ranks these clusters of reviews using four developer-approved attributes: cardinality, date, average rating, and category. Finally, it applies a consensus algorithm to the set of rankings to generate a consensus ranking that can be followed by developers.

We evaluated our approach on user reviews of four Android apps by comparing the results of our approach to gold rankings defined by apps developers. Our results showed that there is a strong correlation (average Kendall rank correlation coefficient of 0.516) between the consensus rankings and the manually-prioritized rankings. Also, apps developers confirmed the efficiency of our approach and showed interest in using it.

While the results are promising, we will explore the following ideas. The generalized Kendall-τ distance is used as a distance measure for finding a consensus of rankings with ties. We intend to experiment with other distance measures to investigate whether they are a better fit to our kind of data. We also plan to study the effect of breaking ties to obtain completely-ordered solutions. We also want to enhance the accuracy of the categorization and clustering approach using other approach than CLAP. We will also perform a validation on a larger number of apps from different marketplaces.

References

[1] A. Dogtiev, "App stores list 2019," https://www.businessofapps.com/guide/app-stores-list/, September 2019.

[2] A. Alsubaihin, A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang, "App store mining and analysis," 08 2015, pp. 1–2.

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

[4] L. Carreño and K. Winbladh, "Analysis of user comments: An approach for software requirements evolution," in *Proceedings - International Conference on Software Engineering*, 05 2013, pp. 582–591.

[5] 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 *ICSE 2014*, 2014.

[6] "Wordpress app on google play store," https://play.google.com/store/apps/details?id=org.wordpress.android.

[7] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. Gall, "Ardoc: App reviews development oriented classifier," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 11 2016.

[8] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. Gall, "How can i improve my app? classifying user reviews for software maintenance and evolution," in *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference*, 09 2015.

[9] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," 05 2013.

[10] 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 *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013, p. 9.

[11] F. Palomba, M. Linares-Vasquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "User reviews matter! tracking crowdsourced reviews to support evolution of successful apps," in *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 2015, pp. 291–300.

[12] P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. Nguyen, "Mining user opinions in mobile app reviews: A keyword-based approach (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 749–759.

[13] S. Scalabrino, G. Bavota, B. Russo, M. Di Penta, and R. Oliveto, "Listening to the crowd for the release planning of mobile apps," pp. 68–86, 2017.

[14] A. Ciurumelea, A. Schaufelbühl, S. Panichella, and H. C. Gall, "Analyzing reviews and code of mobile apps for better release planning," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 91–102.

[15] J. Gebauer, Y. Tang, and C. Baimai, "User requirements of mobile technology: results from a content analysis of user reviews," *Information Systems and e-Business Management*, vol. 6, no. 4, pp. 361–384, 2008.

[16] P. Laurent, J. Cleland-Huang, and C. Duan, "Towards automated requirements triage," in *15th IEEE International Requirements Engineering Conference (RE 2007)*. IEEE, 2007, pp. 131–140.

[17] P. Avesani, C. Bazzanella, A. Perini, and A. Susi, "Facing scalability issues in requirements prioritization with machine learning techniques," in *13th IEEE International Conference on Requirements Engineering (RE'05)*. IEEE, 2005, pp. 297–305.

[18] R. Beg, Q. Abbas, and R. P. Verma, "An approach for requirement prioritization using b-tree," in *2008 First International Conference on Emerging Trends in Engineering and Technology*. IEEE, 2008, pp. 1216–1221.

[19] S. Keertipati, B. T. R. Savarimuthu, and S. A. Licorish, "Approaches for prioritizing feature improvements extracted from app reviews," in *Proceedings of the 20th international conference on evaluation and assessment in software engineering*. ACM, 2016, p. 33.

[20] C. Gao, B. Wang, P. He, J. Zhu, Y. Zhou, and M. R. Lyu, "Paid: Prioritizing app issues for developers by tracking user reviews over versions," in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2015, pp. 35–45.

[21] M. J. A. N. de Caritat and M. De Condorcet, *Essay on the application of analysis 'a the probability 'e of decisions made à the plural é of voices*, 1785.

[22] J. G. Kemeny, "Mathematics without numbers," *Daedalus*, vol. 88, no. 4, pp. 577–591, 1959.

[23] S. Cohen-Boulakia, A. Denise, and S. Hamel, "Using medians to generate consensus rankings for biological data," in *International Conference on Scientific and Statistical Database Management*. Springer, 2011, pp. 73–90.

[24] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, "Rank aggregation methods for the web," in *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 613–622.

[25] D. M. Pennock, E. Horvitz, C. L. Giles *et al.*, "Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering," in *AAAI/IAAI*, 2000, pp. 729–734.

[26] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee, "Comparing and aggregating rankings with ties," in *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2004, pp. 47–58.

[27] B. Brancotte, B. Yang, G. Blin, S. Cohen-Boulakia, A. Denise, and S. Hamel, "Rank aggregation with ties: Experiments and analysis," *Proceedings of the VLDB Endowment*, vol. 8, no. 11, pp. 1202–1213, 2015.

[28] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.

[29] N. Ailon, M. Charikar, and A. Newman, "Aggregating inconsistent information: ranking and clustering," *Journal of the ACM (JACM)*, vol. 55, no. 5, p. 23, 2008.

[30] J. C. de Borda, "Memoir on the elections in the poll, 1781," *History of the Royal Academy of Sciences, Paris*, 1953.

[31] R. Fagin, R. Kumar, and D. Sivakumar, "Efficient similarity search and classification via rank aggregation," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 2003, pp. 301–312.

[32] "Replication package," https://doi.org/10.5281/zenodo.3748768.

[33] M. G. Kendall, *Rank correlation methods*. Griffin, 1948.

[34] J. Cohen, *Statistical power analysis for the behavioral sciences*. Routledge, 2013.