What's Inside a Cluster of Software User Feedback: A Study of Characterisation Methods

Peter Devine*, James Tizard*, Hechen Wang*, Yun Sing Koh†, Kelly Blincoe*
*Department of Electrical, Computer, and Software Engineering
†School of Computer Science
University of Auckland
New Zealand
{pdev438, jtiz003, hwan531}@aucklanduni.ac.nz, {y.koh, k.blincoe}@auckland.ac.nz

Abstract—**Feedback from software users is vital for engineering better software requirements. One tool for extracting requirements from online user feedback is clustering, where the most mentioned topics are found by grouping similar feedback together. For these topics to be understood, clusters have been summarized in previous work using characterizing phrases or sentences. This work evaluates which method of characterization (unigrams, bigrams, trigrams, or sentences) is most effective for understanding the semantic meaning of a whole cluster using feedback from multiple feedback sources. We evaluate multiple characterization methods to determine the ability of each method to create distinct, descriptive characterizations. We further evaluate the amount of requirements relevant characterizations created by each characterization method. We find that unigrams, bigrams, trigrams, and full sentences all perform similarly in distinguishing clusters from each other. However, we find that fewer and more expressive characterizations, such as full sentences, contain more requirements relevant information from a feedback cluster compared to more numerous but less expressive unigrams, meaning a sentence will better summarize the important requirement relevant information from a cluster. Our findings inform the future development of user feedback clustering tools, with different cluster characterization methods being quantitatively measured for the first time.**

I. Introduction

Knowing what the users of a software product want is vital for the continual improvement of that product. One way to gain insight into what users want is by analysing the product feedback that is often left by users on online platforms such as App Stores, Twitter, and product forums. Feedback on these platforms have all been shown to contain information that is relevant to requirements engineering (RE) [30], [13], [37]. Manually reading all user opinions on a product may be impractical due to the sheer volume of feedback available online, and so many automated approaches to analysing this feedback have been proposed within the literature (e.g., [16], [31]).

Clustering, which is a technique that collects user feedback into semantically linked groups, has been used widely within the literature. Clustering has been shown to generate clusters that effectively group semantically similar user feedback together, both in Tweets [36] and in App Store Reviews [11], [35]. This includes grouping feedback into meaningful clusters of similar functional bug reports or feature requests [35]. Clustering results in a (potentially large) list of feedback in each cluster. Therefore, feedback contained within a cluster must still be somehow summarized to software developers so they do not need to manually read all of the feedback in a cluster to make sense of it.

Summarization of clusters has been done within the literature by identifying characterizations such as a set of individual words (unigrams) [35], pairs of words (bigrams) [11], sentences [11], or full Tweets [36] that can be used to describe the many pieces of feedback that are contained within a cluster. Some work has been done to understand which method of cluster characterization best summarizes feedback, with Gao et al. [11] finding that sentences describing user feedback have a higher word embedding similarity to subsequent changelogs than bigrams. However, this work compares the same number of phrases against sentences, meaning that sentences may only be better summaries due to their higher word count. There has been little work evaluating the direct effectiveness of different characterizations in summarizing feedback within a cluster given a fixed set of words.

An effective characterization of a cluster must be both **descriptive** of the cluster content and **distinct** from other clusters' characterizations [18]. Moreover, in the context of summarizing user feedback for the purpose of understanding software requirements, we also believe cluster characterizations should focus on the **requirements relevant** information within the cluster. While previous work has found that clustering tools are effective at clustering requirements relevant feedback together [35], it is unclear as to the requirements relevance of the subsequently generated characterizations. While a descriptive and distinct characterization may be generated for a cluster of user feedback, it may relate to the general semantic meaning of the feedback in the cluster (e.g. "thanks" or "problem") rather than specifically communicating the requirements relevant details of the cluster (e.g. "crashing" or "dark mode"). Thus, a question remains as to how effective these clusters would be at communicating their requirements relevant contents to a user of such a tool.

For this reason, this work seeks to evaluate which method of characterizing clusters is most suitable for creating descriptive, distinct, and requirements relevant characterizations. This informs how future clustering tools should be designed in order to most effectively summarize important requirements relevant information within user feedback.

This resulted in two main research questions:

- **RQ1:** What characterization method is best able to describe and distinguish a cluster compared to a random baseline?
- **RQ2:** What characterization method best communicates requirements relevant information from a cluster?

In this work, we evaluate four methods of characterizing clusters: unigrams, bigrams, trigrams, and sentences. This evaluation is performed on user feedback from five user feedback platforms (Google Play Store, Apple App Store, Reddit, Twitter, and product user forums) sourced from 6 literature datasets [4], [14], [5], [35], [37], [38], as well as additional user feedback collected for this study. The evaluation was done in two stages, firstly as a random characterization comparison task, in which labellers attempted to select the correct characterization for a cluster given both the correct characterization and a random characterization. Secondly, we also performed a requirements relevance labelling task, in which characterizations were manually labelled as requirements relevant or requirements irrelevant. Together, this evaluation gives insight into which methods of characterization work best for different platforms, and, thus, which methods of characterizing clusters can be most effective across a range of feedback.

II. Background

Prior work has shown that user feedback is valuable for product development insights, and much recent research has investigated user feedback from online sources [6], [16]. User feedback from Twitter [15], App Stores [30], and product forums [37] have all been shown to contain useful information for requirements engineering. Due to the large amount of user feedback available online, manually identifying requirements relevant information can be too time-consuming [12]. Thus, various methods and tools to analyze this online user feedback have been proposed. A systematic review by Lim et al. found that clustering methods were some of the most popular ways in analysing user feedback, alongside classification [21]. The classification methods typically aim to categorize online user feedback into predefined categories, such as bug reports or feature requests [23], [31], [28]. This still leaves significant work for the product development team to understand, for example, which user feedback is related to the same bug report or feature request. Clustering can help to reduce this effort by grouping together similar feedback. Lim et al. found that topic modelling, such as LDA [2] and BTM [39], and classical clustering methods, using algorithms such as k-means clustering [24], were the two most popular methods used to cluster together similar user feedback.

Scalabrino et al. proposed the CLAP model, which performs analysis of app reviews [35]. Firstly, all reviews are classified into classes such as "functional bug report", "suggestion for new feature", and "report of security issues". The reviews contained within a class are then clustered using unsupervised machine learning with the DBSCAN algorithm [10]. Each cluster is then classified as "high priority" or "low priority" with high priority clusters being sorted to the top of the list of cluster results. CLAP was found to have high accuracy in categorizing user reviews, an ability to create meaningful clusters of related reviews, accuracy in prioritising clusters, and positive qualitative feedback from industrial contexts, all of which show promise for the adoption of clustering user feedback in the RE field more broadly.

Similar to CLAP, other methods and tools have been proposed to cluster online user feedback. Stanik et al. presented a tool to cluster feedback from Twitter [36]. This work embedded Tweets to a large telecommunications company using SBERT [33], before clustering these embeddings using a HDBSCAN algorithm [26]. Like CLAP, this work showed high coherence within the clusters generated by this method, highlighting its potential usefulness for developers analysing their users' feedback. Gao et al. proposed MERIT [11], a clustering tool that uses an Adaptive Online Biterm Sentiment-Topic Model (inspired by the biterm topic model [39]) to cluster app reviews. MERIT was found to be accurate at identifying emerging issues in new app releases.

While these studies show that clustering is an effective way of grouping user feedback into semantically coherent groups, there is a lack of knowledge on how user feedback clusters such as the ones produced by these methods and tools should be communicated to the product development team. CLAP summarized clusters by showing the four most popular terms (unigrams) within the reviews of each cluster [35]. The tool created by Stanik et al. used a representative tweet to summarize each cluster [36]. MERIT used both the three most descriptive phrases (mainly consisting of bigrams) and the three most descriptive sentences extracted from the feedback text to characterize each cluster. The studies of CLAP and the tool created by Stanik et al. did examine the effectiveness of different characterization techniques. The study of MERIT did compare their two characterization techniques (phrases and sentences). However, the characterizations were evaluated against software changelogs instead of directly examining the effectiveness of the characterization to summarize the cluster. They found that sentences had higher coverage in the changelogs compared to phrases. However, as they report, this result could simply be due to the fact that sentences contain more words than phrases and there were an equal number (3) of sentences and phrases being compared. It remains unclear as to which summarization technique is most effective when the number of words is roughly equal, which is a pertinent question given that space within a UI layout can be limited when designing feedback summarization tools. In addition, none of the studies have examined whether the cluster summarizations in the existing methods and tools capture the requirements relevant content of a user feedback cluster. Thus, an open question remains as to how best these semantically coherent and requirements relevant clusters can be communicated to a requirements analyst given a finite amount of words in a requirements relevant way.

It is for these reasons that this work seeks to evaluate the different ways in which clusters have previously been

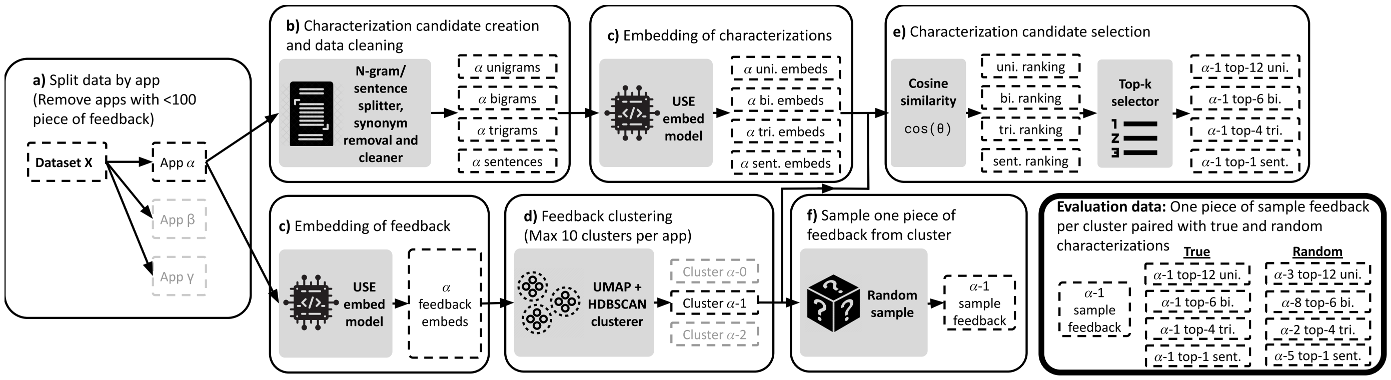


Fig. 1. Diagram depicting the cluster characterization data preparation process for one cluster of hypothetical app α from dataset X

characterized within the literature to assess which are most appropriate for use in a requirements engineering tool.

III. Method

In this work, we clustered user feedback from six literature datasets, supplemented by user feedback data we collected from five platforms. Feedback was split up by app, each piece of feedback for that app was embedded, and then these embeddings were clustered together. From these clusters, characterizations were generated for each, with separate characterizations created using unigrams, bigrams, trigrams, and sentences, all extracted from the corpus of feedback from that app. A pictorial description of our characterization creation process can be seen in Fig 1. These cluster and characterization pairs were then evaluated by human annotators through a random characterization comparison task and a requirements relevance classification task. This section details how each part of this process was carried out more thoroughly.

A. User feedback data

For the evaluation in this study, we selected data relating to multiple apps across different platforms from a variety of datasets such that our results cover a variety of different use cases and data distributions. We used user feedback from six datasets made available by prior studies in the literature [4], [14], [5], [35], [37], [38], which contain feedback from four different platforms: the Google Play Store, the Apple App Store, Twitter, and user product forums. These datasets contain user feedback relating to a wide range of apps in each. Since the majority of these datasets only averaged a few hundred pieces of feedback per app, and this is too low to generate a significant amount of clusters per app, more user feedback was collected for this study. We collected data relating to the Spotify music player[1] due to its popularity, as it has millions of reviews on both the Apple and Play stores, millions of followers on its dedicated support Twitter account, and hundreds of thousands of members in its dedicated subreddit. Spotify user feedback was collected between the 26th of August and the 17th of September 2020. This data was collected from five sources, the Google Play Store (24,346 reviews), the Apple App Store (5,670 reviews), r/Spotify subreddit on Reddit (1,536 posts), Twitter (6,115 Tweets), and the Spotify product user forum (1,061 posts).

All feedback that was not detected as being English by the `langdetect` Python module [2] was removed for each dataset. Text was then further cleaned by removing all URL links from feedback.

Requirements engineers are most often interested in generating requirements related to a single software application. Thus, when clustering tools are applied to user feedback, they usually create clusters separately for each app. For this reason, each dataset was split up into individual apps, such that only feedback that is from the same dataset and the same app can be embedded and clustered together. This splitting is detailed in a) of Fig 1. Table I describes the number of pieces of feedback in each dataset, the number of pieces of feedback after removing all non-English language posts, the number of apps within that dataset, the number of apps that were included in the clusters generated, and the number of clusters that were created from those apps.

B. Characterization candidate creation and data cleaning

In order to characterize our clusters, we first generate a list of candidate characterizations. We do this for 4 types of characterizations: unigrams (single words), bigrams (two consecutive words), trigrams (three consecutive words), and full sentences. Unigrams, bigrams, and sentences were chosen as characterization methods due to their previous use to characterize clusters within the literature [11], [35], [36]. We also sought to test longer n-grams than those explored in the established literature, but there is a trade-off between the number of n-gram lengths tested and the time it takes for the manual labelling tasks used in our analysis. For this reason, trigrams were also included in our evaluation, but we leave examining longer n-grams for future work. Candidate selection is illustrated in b) of Fig 1.

A list of example characterizations related to a randomly sampled piece of feedback from a cluster can be seen in

[1]https://www.spotify.com/

[2]https://pypi.org/project/langdetect/

TABLE I
TABLE DESCRIBING THE NUMBER OF PIECES OF FEEDBACK CONTAINED WITH EACH DATASET USED FOR EVALUATION, THE NUMBER OF PIECES OF FEEDBACK ONCE NON-ENGLISH LANGUAGE FEEDBACK HAD BEEN REMOVED, THE NUMBER OF APPS WITHIN EACH DATASET, AND THE NUMBER OF CLUSTERS THAT THEY ENDED UP GENERATING.

Dataset ID	**Citation**	**Feedback source**	**Downloaded data size**	**Post-cleaning size**	**Number of apps**	**Number of apps with clusters**	**Number of clusters**
A	[4]	App reviews	181,081	141,877	4	4	40
B	[5]	App reviews	2,325	1,415	27	3	8
C	[14]	App reviews	6,159	3,863	7	7	28
D	[35]	App reviews	3,000	2,867	694	1	2
E	[37]	Forums	3,806	3,293	3	3	16
F	[38]	Twitter	4,000	3,795	10	10	24
G	Supplementary data for Spotify Twitter	Twitter	6,115	1,653	1	1	10
H	Supplementary data for Spotify Apple App Store	App reviews	5,670	2,889	1	1	10
I	Supplementary data for Spotify Play Store	App reviews	24,346	16,394	1	1	10
J	Supplementary data for Spotify Reddit	Reddit	1,536	840	1	1	4
K	Supplementary data for Spotify Forum	Forums	1,061	1,032	1	1	8

TABLE II
EXAMPLE SAMPLE FEEDBACK FROM A CLUSTER, AND ITS ASSOCIATED CHARACTERIZATIONS USING DIFFERENT METHODS.

Sample feedback	I had use app 6 month ... but now i cant open my app... when i try it is indicate offline conection ... but internet is conected....
Unigram characterization	login, offline, authentication, unable, connect, apk, password, reconnect, unavailable, unusable, unresponsive, connectivity
Bigram characterization	unable sign, app unresponsive, offline issue, account error, bug login, unable access
Trigram characterization	unable login app, unable login offline, unable login facebook, app bug login
Sentence characterization	Won't let me log in just keeps saying I have no Internet connection when every other app works fine?

Table II. Note that the characterizations are extracted from all feedback in the cluster, not only the sample feedback presented in the table.

Unigrams In order to generate unigrams, we first split sentences into individual words. This was done by applying the word splitter from the PyICU package[3] to the entire corpus of lowercased feedback for a given app. Any words longer than 15 characters were also excluded, as was done by Man et al. because such long strings are seldom valid words [25]. All unigrams that only contained numeric characters were also excluded due to their lack of semantic expression. We removed all misspellings or otherwise noisy non-standard words by excluding any unigrams that were not recognized as valid words by the Autocorrect library's[4] correctly spelled word list. Conjugations and forms of the same word were all unified by using lemmatization, which changes words into their common base form (E.g. "crashes", "crashing", "crashed" to "crash"). All words were lemmatized using the NLTK lemmatizer [1] to reduce the number of highly similar words within our unigram list. We used a list of stopwords (words which carry little semantic meaning by themselves, such as "a", "the", and "do") from the stopwordsiso list[5] and removed any words on this list from our candidate word list. This resultant list became our unigram candidate list.

Bigrams and trigrams Using the cleaned unigram candidate list, we applied the NLTK n-gram creator[6] to each piece of feedback, which joins each unigram to its neighbours

[3]https://pypi.org/project/PyICU/
[4]https://github.com/filyp/autocorrect
[5]https://github.com/stopwords-iso/stopwords-iso
[6]https://www.nltk.org/api/nltk.util.html#nltk.util.ngrams

to form all possible bigrams and trigrams of the sequence. Any n-grams that contained repeated words were removed from these candidate list to avoid repetitive characterizations of the clusters. The maximum number of unique trigrams was 100,110 in the Facebook dataset of the Chen et al. dataset [4]. Due to computational constraints when comparing embeddings, we kept the 10,000 most popular n-grams for each app, with only 7 out of the 33 apps studied having a unique bigram or trigram list longer than this limit.

Sentences Similarly to generating unigrams, in order to generate characterization candidate sentences, we first split up our feedback into individual sentences. To do this, we use the PyICU[3] sentence splitter, which breaks sentences as defined by the break rules of the Unicode Organisation[7]. In order to reduce the amount of single or few word sentences that contained little information, short sentences (those with less than 5 words) were removed from our list of candidate sentences. This step was validated by manually inspecting 100 sentences with less than 5 words, with only a small minority (4 out of 100) identified as potentially containing useful information (for example "everything all crashed"). This validation was done by two of the authors who agreed on all cases. For this reason, these sentences were excluded from our candidate list.

Synonym removal Initial experiments found that characterizations using unigrams, bigrams, and trigrams contained many synonymous words and phrases. For example, "delete" and "deletion", or "app crash" and "crash app". Some of these were handled by the lemmitization performed when cleaning the data. However, some still remained after lemmitization. Therefore, pairs of characterizations that were highly similar were removed in the following way. Embeddings of all candidates were generated, and compared pairwise with each other. Then, one of any pair of characterizations that had a similarity greater than 0.8 were excluded from the candidate list. This ensured diversity within the candidate list. Experiments were done to determine an appropriate threshold, with a sample of characterizations being manually evaluated for their presence of synonyms. We experimented with thresholds of 0.7, 0.8, and 0.9, finding that 0.8 was the best threshold. An example is a cluster which contained feedback such as *"After the last update one third of my PDF files fail to open in the built in reader They worked fine under the old built in pdf reader"* and *"Cannot open 225mb PDF Excellent for everything except for opening larger PDFs. Please fix!!!'*, a threshold of 0.7 yielded trigrams of "annoy pdfs crash", "ipad select pdf", "pdfs finally figure", "ipad update install", a threhold of 0.8 yielded "unable pdf file", "issue bad pdfs", "pdfs screen black", "pdf annoy constantly", and a threhold of 0.9 yielded "update unable pdf", "app pdfs update", "bad pdfs app", "pdf viewer update". Thus a threshold of 0.8 was chosen as it represents a middle ground between representing a cluster and removing synonyms.

[7]https://unicode-org.github.io/icu/userguide/boundaryanalysis/break-rules.html

C. Feedback embedding and characterization embedding

A text embedding is a numerical vector representation of natural language text. These representations are designed to encode semantic meaning into natural language text which can be quantified, with two sentences that are semantically similar also producing vectors or embeddings that are similar. In order to cluster text, we first convert each piece of feedback into such an embedding such that it can then be quantitatively compared to every other piece of feedback, and thus eventually clustered. This is done in the same manner as previous work such as that by Stanik et al. [36]. We used the Universal Sentence Encoder (USE) [3] to embed the feedback due to its proven efficacy in embedding requirements related data together [8]. Each piece of user feedback and every cluster characterization candidate was encoded using the official latest implementation of the USE model hosted on TFHub[8]. Embedding of feedback and characterizations is detailed in both c) steps of Fig 1.

D. Feedback clustering

The embeddings of user feedback for each app were clustered using HDBSCAN [26], which has been shown by Stanik et al. to create coherent feedback clusters [36]. In order to achieve a diverse mix of clusters across apps, we limited our selection of clusters to a maximum of 10 clusters per app per platform, which were randomly sampled from all clusters. This meant that our Spotify dataset clusters were limited to 10 clusters per platform due to them only representing one app.

Clustering algorithms such as HDBSCAN require calculating the distance of individual points to others to determine their relative closeness to assign them to clusters. This distance calculation can be computationally expensive at higher dimensions. Due to the fact that the USE embedder outputs a 512 dimension embedding of a piece of text [3], clustering each point based on its position within this 512 dimension space was beyond our computational limitations. For this reason, a dimensionality reduction technique was applied to all embeddings before being clustered. The Uniform Manifold Approximation and Projection (UMAP) manifold learning technique [27] creates a graph representation of the data and then optimizes a low-dimensional version of the graph to be as structurally similar to the original graph as possible. This technique was used due to its proven performance in creating semantically coherent user feedback clusters, as shown in work by Stanik et al. [36]. As was done by Stanik et al., we used the following parameters: a UMAP output dimensionality of 20, the number of neighbors considered in the UMAP algorithm to be 100, the UMAP minimum distance to be 0, and the HDBSCAN minimum cluster size at 30. All other hyperparameters used were the default for the HDBSCAN Python package[9] (epsilon = 0.0, distance metric = euclidean, alpha = 1.0) and Python implementation of UMAP[10].

Due to the requirement by the UMAP to have at least as many embedding points as number of neighbors considered in

[8]https://tfhub.dev/google/universal-sentence-encoder-large/5
[9]https://github.com/scikit-learn-contrib/hdbscan
[10]https://github.com/lmcinnes/umap

the dimensionality reduction step, only apps with more than 100 pieces of feedback were clustered. This resulted in the creation of 160 clusters taken from 33 apps across all datasets. This clustering step is described in d) of Fig 1.

E. Characterization candidate selection

For each cluster, characterizations were selected for each of the four characterization techniques (unigrams, bigrams, trigrams, and sentences). To select the characterization for each of these types, the encoded list of characterization candidates described in section III-B were compared against all encoded pieces of feedback for a given cluster using cosine similarity. While cosine similarity was used here, early experiments showed little difference in candidate ranking when using other distance metrics. The overall similarity score for each candidate characterization was calculated by summing the cosine similarity for that characterization with each piece of feedback in the cluster. This calculation is described in equation 1.

$$s_c = \sum_{i=F} sim(e_c, e_i) \tag{1}$$

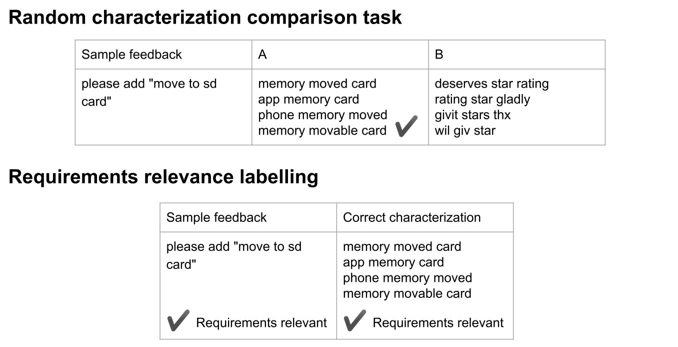
Where e_c is the embedding of the candidate characterization, e_i is the embedding of the i-th piece of feedback, F is the set of all pieces of feedback within a cluster, sim is the cosine similarity function, and s_c is the overall similarity score for candidate characterization c.

Candidates were ranked from most similar to least, and the top k were selected for inclusion in the characterization. Values of k=12 was used for unigrams, k=6 for bigrams, k=4 for trigrams, and k=1 for sentences. These values were chosen so as to balance the number of words included within each characterization (i.e., there are 12 words for each characterization for the unigrams, bigrams, and trigram characterizations). This number of 12 words was chosen so that these characterizations would be at least as long, and thus not be disadvantaged in their characterization ability by a fewer number of words compared to the sentence characterizations. The average number of words in the sentence characterizations chosen was 9.84 and 81% of sentences contained less than or equal to 12 words.

Each cluster was then characterized in this way for each characterization type. This resulted in four characterizations (one for each of unigrams, bigrams, trigrams, and sentences) for each of the 130 clusters created. Selection of characteristic n-grams and sentences from the list of candidates is described in e) of Fig 1.

F. Evaluation criteria

In order to evaluate the characterizations of each cluster, manual labelling was employed. A sample piece of feedback of each cluster was randomly sampled from all feedback within that cluster, and this sample feedback was stored with its respective characterizations. The sampling of feedback can be seen described in f) of Fig 1. Two separate evaluation tasks were then undertaken - a **random characterization comparison task** and a **requirements relevance evaluation task**. These evaluation tasks were carried out by three software engineering PhD students, all of whom have experience working as a software engineer. Each individual labelling was done by two separate people for both tasks in order to allow for the calculation of inter-coder reliability across our dataset.


Random characterization comparison task

Sample feedback	A	B
please add "move to sd card"	memory moved card app memory card phone memory moved memory movable card ✓	deserves star rating rating star gladly givit stars thx wil giv star

Requirements relevance labelling

Sample feedback	Correct characterization
please add "move to sd card" ✓ Requirements relevant	memory moved card app memory card phone memory moved memory movable card ✓ Requirements relevant

Fig. 2. Diagram depicting the format of the data used in the labelling process of both evaluation tasks

A diagram visually depicting these two tasks can be seen in Fig. 2.

1) Random characterization comparison task: In line with previous similar cluster evaluation tasks in the literature, such as the intruder detection tasks used by Guzman et al. and Stanik et al. [15], [36], a random characterization and a true characterization were selected for each cluster and paired with the sample feedback for that cluster. The generation of true and random cluster characterizations is shown in the final output of Fig 1. Evaluators were instructed to read the sample feedback and choose the most appropriate characterization for this feedback from the two presented options (the randomly selected characterization and the most similar characterization selected using the method described in Section III-E). This random characterization comparison was repeated for all characterization types, with only one characterization type being considered at a time (e.g. only unigram characterizations were compared against other unigram characterizations and so forth). This evaluation of comparing a true characterization to a randomly sampled characterization helps to determine how distinguishable an intentionally created characterization is compared to a random one. Higher accuracy on this task signifies that a cluster characterization is better able to describe the contents of the cluster and distinguish it from other clusters based on human judgement.

Three accuracies are reported: **per-coder accuracy** is the accuracy of an individual labelling of a characterization of a cluster, **both-coder accuracy** is the accuracy of both labellers choosing the same correct characterization of a cluster, and **either-coder accuracy** is the accuracy of either one of the coders choosing the correct characterization. These accuracies are then reported across all characterization types, in order to compare the comparative strength in characterization ability between unigrams, bigrams, trigrams, and sentences.

2) Requirements relevance labelling: Once each labeller had labelled a cluster for the random characterization comparison task, they also labelled each sample feedback and each matching characterization as requirements relevant or not relevant. The sample feedback was labelled as relevant or not relevant in every cluster. For clusters whose sample feedback was labelled as requirements relevant, its matching characterization was also labelled as requirements relevant or not. Labelling was done using manual content analysis [20], in which a definition of requirements relevance was first established amongst all labellers, a small amount of example sentences were labelled together as practice, before individually labelling the dataset. This ensured consistency of labelling standards between evaluators. The definition of "requirements relevant" used by the labellers was derived from Santos et al. [34] and centered around mentioning a specific feature or behaviour of the product in relation to a bug report, feature request, or other information pertinent to a software developer. Before labelling began, the three labellers first examined a small sample of characterizations for their requirements relevance to their associated sample feedback. Once an agreement was reached as to which characterizations best related to the requirements of the feedback within the cluster, these characterizations were included as part of a labelling guideline for future labelling. In order to ensure that both labellers were labelling the same characterization (in relation to the given sample feedback) for requirements relevance, only clusters which had characterizations that been correctly selected by both labellers in the previous random characterization comparison task were labelled for requirements relevance.

The inter-coder agreement score of this labelling process was calculated using Cohen's Kappa [7], and resulted in a κ of 0.677 for the sample feedback labelling (with 84.2% of sample feedback having the same label from both evaluators) and 0.555 for the characterization labelling (with 78.3% of characterization labels having the same label from both evaluators). These scores respectively indicate substantial and moderate inter-coder agreement according to Landis and Koch [19].

A sample piece of feedback or characterization was then evaluated to be requirements relevant only if it was labelled as such by both labellers, so as to ensure a high degree of precision in identifying truly requirements relevant feedback from our labelling process.

IV. Results

A. Random characterization comparison task

RQ1: What characterization method is best able to describe and distinguish a cluster compared to a random baseline?

The results of the random characterization comparison task can be seen in Table III.

These results describe the percentage of characterizations that are correctly selected compared to a random characterization in the manual analysis of the most appropriate characterization. A high percentage means that the characterizations of the clusters are descriptive of the cluster feedback contents and distinct from other clusters' characterizations. Conversely, a low percentage signifies that a characterization does not represent the content of a cluster well or is too similar to other characterizations of clusters.

TABLE III
Label accuracy for an individual coder, accuracy of either coder labelling a characterization correctly, and accuracy of both coders labelling it correctly. There is little difference in the accuracy scores between characterization types.

	Per-coder	Either coder	Both coder
Unigrams	0.838	0.900	0.775
Bigrams	0.853	0.906	0.800
Trigrams	0.847	0.944	0.750
Sentences	0.888	0.956	0.819

Averaged over all characterization types, per-coder accuracy is 85.6%, both coder accuracy is 78.6%, and either coder accuracy between 92.7%. For all characterization types, we can see that per-coder accuracy ranges between 83-89%, both coder accuracy between 75-82%, and either coder accuracy between 90-96%. These results all exceed the random baselines of 50%, 25%, and 75% significantly.

We find cluster characterization accuracy to be similar across different characterization types, with accuracy not differing by greater than 7% for any characterization type. With this finding, we do not detect any characterization type (unigrams, bigrams, trigrams, and sentences) to significantly characterize the contents of its cluster more descriptively or distinctly than any other.

In 14.1% (90 total) cases, evaluators disagreed on which characterization was most appropriate for a given sample feedback. An example of this can be seen with the sample feedback *"Music, podcasts and everything mine!"* which had candidate bigram characterizations of *"spotify worth, apps music, music stream, music android, basically spotify, spotify offline"* and *"podcast apps, podcast absolutely, favourite podcast, nice podcast, enjoy podcast, library podcast"*, with the latter being the true generated characterization for the cluster. One characterization focuses on music and another on podcasts, while the sample feedback contains references to both, and so both characterizations can be seen to be relevant to the sample feedback. In this case, the generated characterization is descriptive of the feedback that it contains, but is not successfully distinct from other characterizations, and therefore is not an effective characterization.

In 7.3% (47 total) of cases, both evaluators chose the incorrect, random characterization. An example of this can be seen in the case where the sample feedback was *"otherwise, amazing game with awesome graphics!"*, and the two choices for a sentence characterization were *"otherwise a great game !"* and *"the graphics are fantastic !"*. In this case, both evaluators chose the latter as the matching feedback when the former was the actual generated characterization for this cluster. While the former does have some semantic overlap with the sample feedback (i.e. the use of the word "otherwise" and praising the game), the latter also semantically matches the sample feedback but in a way that is more relevant to the software product (i.e. praising the graphics of an app). As with the previous example, this clustering has resulted in an indistinct characterization in that characterizations of other clusters would also be appropriate for this cluster. Beyond a lack of distinction, this characterization also does not describe the elements of feedback specifically related to the software that a requirements engineer would be more interested in, in that it does not describe the graphics mentioned within the feedback. This characterization is also ineffective as it is both indistinct from other characterizations and does not fully describe the feedback contained within a cluster.

Overall, we observed that the majority (78.6%) of characterizations were chosen correctly by both evaluators. This demonstrates the relative effectiveness of our characterization method overall. When compared across characterization types, we observed no notable differences between unigrams, bigrams, trigrams, and sentences (see details in Table III). We also found that in the minority of cases where both or either evaluator chose an incorrect characterization, characterizations had been shown to be indistinct and to not fully describe the information from the cluster feedback which would be relevant to requirements engineers.

Answer to RQ1:
All characterization methods tested (unigrams, bigrams, trigrams, and sentences) describe and distinguish clusters compared to a random baseline to a similar high degree.

B. Requirements relevance labelling

RQ2: What characterization method best communicates requirements relevant information from a cluster?

The sample feedback labelling evaluation resulted in 34.1% of sample feedback (218 sample feedback relating to 640 characterizations) being labelled as requirements relevant. This reflects previous work, such as that by Chen et al. [4], that found that informative user feedback ranges between 24.6% and 55.4% of all app reviews for different apps. Of the 34.1% of sample feedback that were labelled as requirements relevant, 82.6% (180 characterizations out of 218 total) were correctly selected and agreed upon by both labellers in the prior random characterization comparison task.

From these 180 cluster characterizations, the percentage labelled as requirements relevant can be seen in Table IV. These numbers range dramatically between characterization types. Only 3 unigram characterizations (nearly 7% of all unigram characterizations) were labelled as requirements relevant. This stands in contrast to 21 sentence characterizations (45.7% of sentence characterizations correctly selected and associated with a requirements relevant sample feedback) that were labelled as requirements relevant. Between these two, bigrams

TABLE IV
PERCENTAGE OF CHARACTERIZATIONS THAT ARE ASSOCIATED WITH REQUIREMENTS RELEVANT SAMPLE FEEDBACK THAT WAS ALSO LABELLED AS REQUIREMENTS RELEVANT.

	% Characterizations requirements relevant
Unigrams	6.98%
Bigrams	31.91%
Trigrams	29.55%
Sentences	45.65%

and trigrams have an intermediate requirements relevance rates of 31.9% and 29.5%, respectively.

While unigrams had the most individual characterizations (12 unigrams per cluster), we can see that a small set of longer n-grams is more effective at capturing the requirements relevant content of a cluster compared to a large amount of shorter n-grams. We can see this trend somewhat borne out with bigrams and trigrams having lower requirements relevance than sentences but higher than unigrams. Within the 21 requirements relevant sentence characterizations, the average length of a sentence is 13.7 words with 62.0% of these sentences below the 12 words. This average is longer than the 9.84 average sentence length of all sentence characterizations in our dataset. This suggests that adding more words to a single characterization has a positive effect on the ability of the characterization to summarize requirements relevant information.

An example of a requirements relevant characterization can be seen in the sentence characterization associated with the sample feedback *"Spotify all of a sudden not playing certain local files. I have a few local files that I got from soundcloud that used to play but now do not, even if I re download them, they appear in the local files library but they are greyed out. They used to play and still play on my touch pad that has not connected to wifi in a few weeks."*, which is *"Spotify Local Files arent syncing with mobile device correctly."*. This characterization succinctly summarizes the main points of the requirements relevant information contained within the feedback representing the cluster. A similar example can be seen with the trigram characterization for the sample feedback *"pics don't load after the new update."*, which is *rarely load update, photo mobile update, picture feature update, picture refuse load*. Again, the characterization covers both the problem (the picture not loading) and the context (after an update) which would inform requirements engineers that a bug needs to be addressed from this cluster of user feedback.

Conversely, examples of related but requirements irrelevant characterizations for requirements relevant sample feedback are also informative to our understanding of the effectiveness of these characterizations. The sample feedback *"2/3 of my Home Screen is Podcast suggestions. It's really annoying. Considering going back to Apple Music. And use separate app for podcasts again"* has a trigram characterization of *podcast play unavailable, podcast video issue, podcast feature disable, podcast spotify offer*, which does mention an aspect of the software product (podcasts), but fails to summarize the requirement itself (changing the UI to reduce the number of podcasts on it). Similarly, sample feedback *"Good. But this would probably be my go-to lockscreen app if you guys integrated a pass code and/or PIN feature."* has a unigram characterization of *wallpaper, tweak, widget, fingerprint, unlock, lock, launcher, setup, minimalist, customizable, android, moto*, which does not mention the pass code, password, or PIN number that was requested in the original feedback.

Therefore, we can see that while some characterizations can effectively summarize requirements, most do not and are instead related to some requirements irrelevant aspect of the set of feedback it is summarizing.

Answer to RQ2:
While requirements relevance of evaluated characterizations is somewhat low, longer characterization types (such as sentences) communicate requirements relevant information more effectively than shorter types (such as unigrams).

V. Discussion

A. Implications

Overall, we find that all characterization types evaluated had a fairly similar ability to both describe and distinguish a cluster of user feedback. This suggests that there is not necessarily a benefit in using one method of characterizing clusters over another purely for describing the whole semantic content of a cluster or distinguishing clusters from each other. However, there is a clear advantage to using full sentences as opposed to shorter characterization types, such as unigrams, when describing specifically requirements relevant information from the cluster. It is also notable that the sentences that were labelled as requirements relevant were longer than the average of all characterization sentences. This indicates that a longer single sentence can contain more requirements relevant information than a shorter one. However, even when the number of words is kept constant (as in our evaluation of unigrams, bigrams, and trigrams), we see that unigrams are considerably worse at describing requirements compared to more expressive bigrams and trigrams.

These results dovetail with those of Gao et al., who showed that a fixed number of sentences are more likely to explain an emerging issue in feedback compared to the same number of phrases [11]. We show that even when the number of phrases (in our case unigrams, bigrams, and trigrams) is increased to include enough words to roughly match that of sentences, sentences are still more effective at summarizing bug reports, feature requests, and other requirements relevant information.

This suggests that using sentences or higher n-grams to summarize a cluster of user feedback will be more useful for conveying the requirements relevant information of that cluster compared to using unigrams. This informs the future development of user feedback summarization tools, as some within the literature thus far have used unigrams or bigrams as their method of characterizing clusters [11], [35]. User feedback analysis tools will benefit from displaying what requirements a cluster is mentioning rather than another characterization that may be semantically linked to the cluster but is not relevant to software development. This will increase the requirements engineering possibilities of these tools, and thus their utility.

The higher requirements relevance of sentence characterizations above other characterization types may be related to the higher expressivity of individual sentences compared to individual unigrams or bigrams. We can see that a small number of more expressive characterizations ultimately communicate requirements relevant information more effectively than a larger number of less expressive characterizations. This is a somewhat intuitive result, but an important one given the widespread use of unigrams and bigrams as characterization methods throughout the literature.

Another learning from these results is the fact that many characterizations do not match the requirements relevant aspects of the feedback that they are summarizing, but instead match another aspect of the feedback. The selecting of characterizations was done using the cosine similarity of feedback to characterization candidates based on a general text embedder (USE) which has been trained on a broad range of natural language processing tasks. This means that these comparisons were not done within the context of software or requirements engineering, but rather in the broader semantic space of language more generally. Therefore, improvement upon these characterizations could possibly be made by using an embedding model which has been specifically trained on data relevant to requirements engineers.

B. Future work

In this work, we compare our characterizations against an identically generated characterization from another cluster. This random baseline is quite a low barrier against which to select a "best" characterizations. Future work could explore how the characterizations generated in this work compare to other, potentially more sophisticated methods of generating characterizations. An experiment could include selecting the best characterization from a selection of multiple characterizations for a specific cluster.

Given that this work has shown the ability of a few expressive characterizations to communicate requirements contained within user feedback, we can look to new ways of generating different types of expressive characterizations for future use in user feedback clustering tools. Extractive summarization relies on using similar methods to those in this paper, in which a selection of candidate characterizations is extracted from a document or corpus of text, from which a few are selected as the summary of that document or corpus. Examples of this include HAHSum [17] and MatchSum [41], and could be used to extract better sentence based characterizations of a set of user feedback which effectively summarize the requirements detailed from a potentially large number of users. Abstractive summarization is also a potentially useful tool for creating user feedback cluster summaries. This method of summarizing a set of texts uses a model which has been trained to generate summaries using text that is not necessarily contained within the original text [9]. Examples of this include PEGASUS [40], and potentially allow for more natural, concise, or comprehensive summary of user feedback than simply extracting the most representative existing sentences from a cluster. This could result in the output of a short report of user feedback covering feedback over a certain time span that is generated periodically by automatic clustering and characterization. Such a report would allow requirements engineers to understand what users want from a piece of software in a quick and comprehensive way without having to read large amounts of feedback manually. It remains for future work to evaluate these methods of generating characterizations compared to those used in this study.

Another possibility for future work would be to fine tune an existing model to specifically create requirements relevant summaries using labelled data. This could be done using a siamese network (as was used to train SBERT [33]), in which a deep neural network (e.g. a pretrained transformer based model such as BERT) is trained to embed semantically related sentences into similar positions within an embedding space. If such a model was trained on pairs of user feedback data, it may be better suited to generating meaningful clusters of feedback compared to general text embedders such as SBERT or USE. Another type of model that could be applied to the problem of summarizing user feedback clusters is a text generation model (e.g. GPT-2 [32] or GPT-3 **??**). These transformer decoder based models are trained to model language by predicting the next word in a document, and can then be used to generate text based on this internal modelling of language. A text generation model could be trained on multiple pieces of user feedback and their summaries (such as multiple bug reports and the resultant issue ticket) such that the model can summarise feedback in a natural and holistic manner. Using models which have been fine-tuned on user feedback may result in more requirements-aware characterizations than using generalised text embedding models as has been done in this work.

Along with using multiple methods to generate characterizations, future work could also consider the optimal length of this characterization. In our work, we have kept the length of the characterization roughly constant across the four characterization methods that we looked at (unigrams, bigrams, trigrams, and sentences). However, future work could explore whether there may be an optimal length of characterization which balances brevity and expressivity best.

VI. Threats to Validity and Limitations

A. *Method for creating clusters and characterizations*

One potential threat to validity is that our results regarding clustering characterization are specific to the methods we used for embedding, clustering, and characterizing feedback in our evaluation. We used a state of the art embedding method, USE [3], for embedding our feedback, which has previously been shown to be the most effective at grouping alike user feedback together [8]. We also used the state of the art methods available in the user feedback clustering literature, UMAP and HDBSCAN, which has previously shown to produce highly coherent clusters of user feedback in Tweets [36]. Future work can replicate this study with other embedding, clustering, and characterization techniques to validate our findings.

Our use of the Autocorrect library also presents a threat to this work. This library may dispose of words that are not officially recognised, but are useful to the developer. The decision to use the Autocorrect library was made to remove noisy or misspelled words that would make our characterizations less meaningful. This tradeoff could be mitigated in future work with a more sophisticated correction technique.

Finally, our characterization method was chosen as a simple summed similarity between the embeddings of feedback within a cluster and the embeddings of the characterization candidates. This method was chosen for its simplicity such that our results did not reflect the choice of a more convoluted or complex characterization method that was specific to our work. However, since this is the first study evaluating the ability of different cluster characterizations to communicate user feedback, it remains for future work to compare other methods for choosing characterizations from candidate lists.

B. *Data threats*

A threat exists in that our results may not generalize to user feedback more broadly. This threat was mitigated by using multiple datasets from the literature that contain feedback from a wide variety of sources in our evaluation. However, it remains for future work to replicate this study on a wider array of user feedback, particularly on sources that were not represented in this study, such as Steam reviews [22] or Facebook posts [29].

There is also a threat of a labeller biasing results as there were only three labellers used in the labelling of data within this study. This threat was mitigated by using three PhD students, all of whom had previous experience working as software engineers, as knowledgeable labellers for this task. Grounding of the definition of requirements relevance in the literature (as described in section III-F) was done to reduce bias in the labellers, and a preparatory joint labelling session done before our evaluation ensured that each labeller knew the standard at which to label. The use of multiple, trained labellers and a consistent, settled labelling schema helped to mitigate the biases of manually labelling data.

Our decision to characterise a cluster using one piece of feedback is a potential threat to the validity of our findings. It could be the case that our random sample of one piece of feedback from a cluster does not represent the rest of the feedback within a cluster. We try to mitigate this threat by using a similar clustering process to previous work, which has shown that clustering feedback using a large, pre-trained language model, UMAP embeddings, and HDBSCAN clustering algorithms results in coherent clusters [36]. It is for future work to assess cluster characterizations while comparing to multiple pieces of feedback from a cluster.

A related threat to the validity of our work is the fact that a piece of feedback or characterization was only considered requirements relevant if both labellers labelled it as requirements relevant, and thus may be under-counting the true number of requirements relevant pieces of feedback and characterizations. This was done to avoid including requirements relevance false positives within our dataset, and the moderate inter-coder agreement scores show that the majority of feedback was agreed upon by labellers. Moreover, we focused on evaluating the differences in requirements relevant rates for each characterization type, and so the gross value requirements relevance is not as impactful on our work. However, while some form of consensus can be achieved for what feedback is "requirements relevant", it remains a somewhat subjective characteristic. It remains for future work to replicate our evaluation with more evaluators to determine the true requirements relevance rate of cluster characterizations.

This threat relates to another threat, that our final evaluation dataset of 180 cluster characterizations was somewhat small. This was due to the fact that we only selected characterizations that were both correctly selected as being the representation of the sample feedback from the cluster and this sample feedback being labelled as requirements relevant by both evaluators. This was done to ensure that both evaluators were labelling the requirements relevance of the correct characterizations that were actually related to the piece of feedback. Moreover, this was done to ensure that there was a potential for the characterization to refer to requirements relevant information, as it would otherwise be impossible if the sample feedback was not requirements relevant. This was done to facilitate both the random characterization comparison and requirements relevance labelling tasks being done on the same data with manual analysis, which had not previously been done within the literature. However, this did result in a smaller evaluation dataset, and so future work could complement our results by using automated tools, such as a requirements relevance or informativeness classifier such as AR-Miner [4], to determine the RE content of various characterizations on a much larger dataset.

Another threat of this work is that our evaluation dataset is heavily biased towards app reviews, so we do not know how our results will generalize to other feedback platforms. The majority of clusters created were generated using app review datasets, which are the most plentiful within the user feedback analysis literature. Therefore, while the initial intention of this work was to also offer a comparison of characterization methods across individual user feedback platforms, we had too few clusters from lesser represented data sources (e.g. forums, Twitter, Reddit) to make any comparison between these meaningful. It remains for future work to replicate this work on a larger set of user feedback from more feedback sources.

VII. Conclusion

In this work, we present a novel evaluation of the summarization of user feedback for a requirements engineer. We carried out a state-of-the-art clustering of user feedback from multiple literature datasets covering a diverse range of feedback platforms. These clusters were summarized using four different characterization methods (unigrams, bigrams, trigrams, and sentences). These characterizations were then manually evaluated for how well they generate descriptive and distinct summaries of the feedback contained within a cluster. Further manual evaluation was done to determine the relevance of these characterizations to requirements included within feedback. We found our characterizations to create highly distinguishing summaries of feedback in general and that all characterization methods evaluated performed similarly in their ability to distinguish a cluster of feedback compared to a random characterization. We also found that a smaller number of longer, more expressive characterizations (such as sentences) were more effective at summarizing the requirements relevant content of the user feedback they were characterizing compared to a larger number of shorter characterizations (such as unigrams). Our results allow insight into how requirements are summarized to requirements engineers, and open many questions as to how this can be better done in the future. Due to the demonstrated effectiveness of expressive characterizations, extractive or abstractive summarization techniques could be used to generate few sentence summaries of feedback clusters in the future. This will allow requirements engineers to quickly understand what their users want from their software, ultimately benefiting the engineer, the users, and the software.

Our replication package for generating the clusters and characterizations in this work, as well as the code for analysing our labelled data has been made available at https://zenodo.org/record/6585857.

References

[1] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[2] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[3] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

[4] Ning Chen, Jialiu Lin, Steven CH Hoi, Xiaokui Xiao, and Boshen Zhang. Ar-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th international conference on software engineering*, pages 767–778, 2014.

[5] Adelina Ciurumelea, Andreas Schaufelbühl, Sebastiano Panichella, and Harald 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)*, pages 91–102. IEEE, 2017.

[6] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. Automated classification of non-functional requirements. *Requirements engineering*, 12(2):103–120, 2007.

[7] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[8] Peter Devine, Yun Sing Koh, and Kelly Blincoe. Evaluating unsupervised text embeddings on software user feedback. In *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pages 87–95. IEEE, 2021.

[9] Wafaa S El-Kassas, Cherif R Salama, Ahmed A Rafea, and Hoda K Mohamed. Automatic text summarization: A comprehensive survey. *Expert Systems with Applications*, 165:113679, 2021.

[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[11] Cuiyun Gao, Jichuan Zeng, Zhiyuan Wen, David Lo, Xin Xia, Irwin King, and Michael R Lyu. Emerging app issue identification via online joint sentiment-topic tracing. *IEEE Transactions on Software Engineering*, 2021.

[12] Eduard C Groen, Norbert Seyff, Raian Ali, Fabiano Dalpiaz, Joerg Doerr, Emitza Guzman, Mahmood Hosseini, Jordi Marco, Marc Oriol, Anna Perini, et al. The crowd in requirements engineering: The landscape and challenges. *IEEE software*, 34(2):44–52, 2017.

[13] Emitza Guzman, Rana Alkadhi, and Norbert Seyff. A needle in a haystack: What do twitter users say about software? In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 96–105. IEEE, 2016.

[14] Emitza Guzman, Muhammad El-Haliby, and Bernd Bruegge. Ensemble methods for app review classification: An approach for software evolution (n). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 771–776. IEEE, 2015.

[15] Emitza Guzman, Mohamed Ibrahim, and Martin Glinz. A little bird told me: Mining tweets for requirements and software evolution. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 11–20. IEEE, 2017.

[16] Claudia Iacob, Rachel Harrison, and Shamal Faily. Online reviews as first class artifacts in mobile app development. In *International Conference on Mobile Computing, Applications, and Services*, pages 47–53. Springer, 2013.

[17] Ruipeng Jia, Yanan Cao, Hengzhu Tang, Fang Fang, Cong Cao, and Shi Wang. Neural extractive summarization with hierarchical attentive heterogeneous graph network. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3622–3631, 2020.

[18] Krista Lagus and Samuel Kaski. Keyword selection method for characterizing text document maps. 1999.

[19] J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

[20] Hartmut Lang. Content analysis for the social sciences and humanities, 1971.

[21] Sachiko Lim, Aron Henriksson, and Jelena Zdravkovic. Data-driven requirements elicitation: A systematic literature review. *SN Computer Science*, 2(1):1–35, 2021.

[22] Dayi Lin, Cor-Paul Bezemer, Ying Zou, and Ahmed E Hassan. An empirical study of game reviews on the steam platform. *Empirical Software Engineering*, 24(1):170–207, 2019.

[23] Walid Maalej and Hadeer Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *2015 IEEE 23rd international requirements engineering conference (RE)*, pages 116–125. IEEE, 2015.

[24] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[25] Yichuan Man, Cuiyun Gao, Michael R Lyu, and Jiuchun Jiang. Experience report: Understanding cross-platform app issues from user reviews. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*, pages 138–149. IEEE, 2016.

[26] Leland McInnes and John Healy. Accelerated hierarchical density based clustering. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 33–42. IEEE, 2017.

[27] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018.

[28] Maleknaz Nayebi, Henry Cho, and Guenther Ruhe. App store mining is not enough for app improvement. *Empirical Software Engineering*, 23(5):2764–2794, 2018.

[29] Emanuel Oehri and Emitza Guzman. Same same but different: Finding similar user feedback across multiple platforms and languages. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 44–54. IEEE, 2020.

[30] Dennis Pagano and Walid Maalej. User feedback in the appstore: An empirical study. In *2013 21st IEEE international requirements engineering conference (RE)*, pages 125–134. IEEE, 2013.

[31] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. Ardoc: App reviews development oriented classifier. In *Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering*, pages 1023–1027, 2016.

[32] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[33] Nils Reimers, Iryna Gurevych, Nils Reimers, Iryna Gurevych, Nandan Thakur, Nils Reimers, Johannes Daxenberger, Iryna Gurevych, Nils Reimers, Iryna Gurevych, et al. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 671–688. Association for Computational Linguistics, 2019.

[34] Rubens Santos, Eduard C Groen, and Karina Villela. A taxonomy for user feedback classifications. In *REFSQ Workshops*, 2019.

[35] Simone Scalabrino, Gabriele Bavota, Barbara Russo, Massimiliano Di Penta, and Rocco Oliveto. Listening to the crowd for the release planning of mobile apps. *IEEE Transactions on Software Engineering*, 45(1):68–86, 2017.

[36] Christoph Stanik, Tim Pietz, and Walid Maalej. Unsupervised topic discovery in user comments. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*, pages 150–161. IEEE, 2021.

[37] James Tizard, Hechen Wang, Lydia Yohannes, and Kelly Blincoe. Can a conversation paint a picture? mining requirements in software forums. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 17–27. IEEE, 2019.

[38] Grant Williams and Anas Mahmoud. Mining twitter feeds for software user requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 1–10. IEEE, 2017.

[39] Xiaohui Yan, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. A biterm topic model for short texts. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1445–1456, 2013.

[40] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR, 2020.

[41] Ming Zhong, Pengfei Liu, Yiran Chen, Danqing Wang, Xipeng Qiu, and Xuan-Jing Huang. Extractive summarization as text matching. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6197–6208, 2020.