2017 IEEE 25th International Requirements Engineering Conference

Mining Twitter Feeds for Software User Requirements

Grant Williams* and Anas Mahmoud†
Division of Computer Science and Engineering, Louisiana Sate University
Baton Rouge, LA, 70803
*gwill83@lsu.edu,†amahmo4@lsu.edu

***Abstract*—Twitter enables large populations of end-users of software to publicly share their experiences and concerns about software systems in the form of micro-blogs. Such data can be collected and classified to help software developers infer users' needs, detect bugs in their code, and plan for future releases of their systems. However, automatically capturing, classifying, and presenting useful tweets is not a trivial task. Challenges stem from the scale of the data available, its unique format, diverse nature, and high percentage of irrelevant information and spam. Motivated by these challenges, this paper reports on a three-fold study that is aimed at leveraging Twitter as a main source of software user requirements. The main objective is to enable a responsive, interactive, and adaptive data-driven requirements engineering process. Our analysis is conducted using 4,000 tweets collected from the Twitter feeds of 10 software systems sampled from a broad range of application domains. The results reveal that around 50% of collected tweets contain useful technical information. The results also show that text classifiers such as Support Vector Machines and Naive Bayes can be very effective in capturing and categorizing technically informative tweets. Additionally, the paper describes and evaluates multiple summarization strategies for generating meaningful summaries of informative software-relevant tweets.**

I. Introduction

In recent years, Twitter has become one of the most popular micro-blogging social media platforms, providing an outlet for millions of users around the world to share their daily activities through real-time status updates. As of the fourth quarter of 2015, Twitter has averaged around 305 million monthly active users. The sheer volume of real-time and highly-diverse data provided by Twitter has revolutionized research in many data science areas. Twitter data has been leveraged to predict the daily ups and downs of the stock market [1], predict the political affiliation of the masses [2], and uncover and explain temporal variations in social happiness [3].

From a software engineering perspective, Twitter has created an unprecedented opportunity for software providers to monitor the opinions of large populations of end-users of their systems [4]. Using Twitter, end-users of software can publicly express their needs and concerns in the form of micro-blogs. In fact, it has become a social media tradition that, with the release of each new mobile app, operating system, or web service, people resort to Twitter to describe their experiences and problems and recommend software to their friends, causing these systems to be *trending* worldwide. Such data can be leveraged to extract rich and timely information about newly-released systems, enabling developers to get instant technical and social feedback about their software.

Motivated by these observations, this paper reports on a three-fold study that is aimed at leveraging Twitter as a main source of useful software user feedback. In particular, we evaluate the performance of multiple data classification and summarization techniques in automatically detecting and summarizing technical user concerns raised in software-relevant tweets. Automation is necessary to deal with the massive scale of Twitter data available, its unique format, and diverse nature [2]. Generated feedback can be used as an input for a well-informed release-planning process by pointing out critical bugs and helping developers prioritize the most desired features that need to be addressed in forthcoming releases [5]. This presents an advantage over classical user feedback collection methods that rely on face-to-face, user reviews, bug tracking, or survey communication. Ultimately, our main objective is to support an adaptive data-driven requirements engineering process that can detect users' needs in an effective and timely manner. In particular, in this paper, we document the following contributions:

- We collect and manually classify 4,000 tweets sampled from the Twitter feeds of 10 different software systems. These systems extends over a broad range of application domains. Our objective is to qualitatively assess the technical value of software-relevant tweets.
- We employ two text classification techniques to accurately capture and categorize the various types of actionable software maintenance requests present in software systems' Twitter feeds.
- We investigate the performance of various text summarization techniques in generating compact summaries of the common concerns raised in technically informative tweets.

The remainder of this paper is organized as follows. Section II motivates our work and describes our research questions. Section III describes our data collection and qualitative analysis process. Section IV and V evaluate the performance of various text classification and summarization strategies for capturing and summarizing technically informative tweets. Section VI discusses the threats to the study's validity. Section VII reviews related work. Finally, Section VIII concludes the paper and discusses prospects for future work.

2332-6441/17 $31.00 © 2017 IEEE
DOI 10.1109/RE.2017.14

II. Motivation and Research Questions

The explosive growth of the computational capabilities of computing devices has led to a drastic increase in the demand for software [6]. This in turn has led to a huge spike in the number of software systems released on a daily basis. For instance, as of March 2015, the Apple App Store alone has reported around 2.25 million active apps, growing by over 1000 apps per day. Software systems, such as operating systems, video games, and social networks have expanded in use to reach vastly broad and diverse populations of users.

This unprecedented level of competition has encouraged software providers to look beyond traditional software engineering practices into methods that enable them to connect with their end-users in a more effective and instant way. An underlying tenet is that user involvement in the software process is a major contributing factor to software success [7]. User feedback contains important information that helps developers to understand user requirements and expectations and identify missing features [8], [9], [10]. Considerable work in this domain has been focused on mining user reviews in mobile app stores (e.g., Apple App Store, Google Play, and Nokia Ovi) [11], [9]. Recent analysis of large datasets of app store user reviews has revealed that almost one third of these reviews were technically informative to app developers [9].

Motivated by these observations, in this paper we exploit the online micro-blogging service Twitter as a more open, more widespread, and more instant source of technically-informative software information. Unlike user reviews in mobile application stores, Twitter feedback is not limited to mobile applications. Rather, it extends to any software system with a sizable user base. Prior research on leveraging micro-blogging services in software engineering has been focused on the developer side, or how communities of software engineers use such services to support their daily development activities (Sec. VII). Analysis of sample software-relevant tweets has revealed that developers frequently make use of social media as a means to facilitate team coordination, learn about new technologies, and stay in touch with the interests and opinions of all stakeholders [12], [13], [14], [15].

In our analysis, we examine the value of Twitter as a source of user feedback that can be translated into actionable software engineering requests. The main objective is to help software developers to instantly and directly connect with their users, and ultimately, survive in a highly-competitive and volatile market. Based on these assumptions, we formulate the following research questions:

- ***RQ$_1$: How informative is Twitter data for software engineers?*** Twitter is a public service. Millions of tweets are generated every day by millions of users all over the world about a vast spectrum of subjects. The assumption that all these tweets carry useful technical information is unrealistic. For instance, users might tweet about software to share their experience with others, ask people to follow them on social media platforms, or recommend a video game to their friends. Therefore, the first objective of our analysis is to determine how technically informative, or useful, software users' tweets are. Technically informative tweets can be described as any user concern that can be translated into an actionable software maintenance request, such as a bug report or a user requirement. Uninformative tweets, on the other hand, can be simply spam or messages that provide no immediate technical feedback to the developer.
- ***RQ$_2$: To what extent can informative software-relevant tweets be automatically classified?*** Assuming software-relevant tweets contain sufficient user technical feedback, can such information be automatically captured? Manually filtering through massive amounts of Twitter data can be a laborious and error-prone task. Therefore, for any solution to be practical, automated support is needed to facilitate a more effective data filtering process that can separate, with a decent level of accuracy, technically informative from uninformative tweets.
- ***RQ$_3$: How can informative tweets be accurately summarized?*** Twitter posts are lexically and semantically restricted. Furthermore, several tweets might raise similar concerns. Presenting such large and heavily redundant amounts of raw tweets to developers can cause confusion. This emphasizes the need for automated methods to summarize informative tweets in such a way that enables a more effective data exploration process.

III. Data Collection and Qualitative Analysis

In this section, we answer our first research question regarding the potential value of tweets for software developers. In particular, we describe our data collection process along with the main findings of our manual qualitative analysis.

A. Data Collection

In our analysis, we used Twitter's Search API to collect our dataset [16]. This API can be customized to search for a specific word or hashtag (#) in Twitter feeds. Twitter has integrated hashtags into the core architecture of the service, allowing users to search for these terms explicitly to retrieve a list of recent tweets about a specific topic. Searching through hashtags can be effective when Twitter is mined at a massive scale to infer the opinions of the masses towards a certain topic, such as learning peoples' views of a certain public figure (search for *#obama*) or certain recent events (*#election*) [3], [17]. However, one of the main drawbacks of hashtag, or word, search is the very high noise-to-signal ratio. More specifically, such queries can return millions of tweets. For example, a search for *#snapchat* returns millions of tweets of the nature *"please follow me on #snapchat"*. While such vast amounts of data can be very useful for inferring public trends, classifying such data manually can be a tedious task. To overcome these limitations, in our analysis, we limit our data collection process to tweets addressed directly to the Twitter account of a given software product (e.g., tweets beginning with *@Windows10*). This strategy ensures that only tweets that are meant to be a direct interaction with the software provider are included.

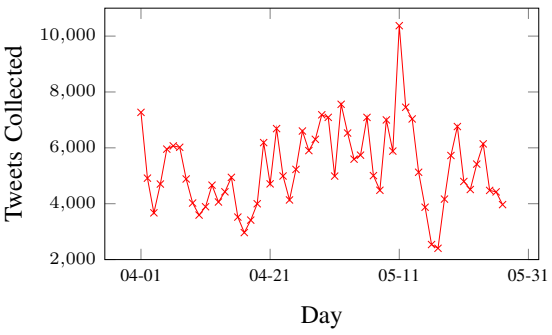

Fig. 1. Number of tweets collected per day (April 1st - May 27th, 2016)

TABLE I
OUR SAMPLE SYSTEMS AND NUMBER OF BUG, REQUIREMENT, AND MISCELLANEOUS TWEETS COLLECTED FOR EACH SYSTEM

System	domain	Bugs	Req.	Other
Android	Operating Systems	69	66	265
Apple Support	Customer service	179	47	174
Call of Duty	Gaming	44	121	235
Google Chrome	Browsing	150	61	189
Instagram	Social Media	87	98	215
Minecraft	Gaming	61	86	253
Snapchat	Social Media	80	172	148
Visual Studio	Development	136	86	178
WhatsApp	Messaging	112	118	170
Windows 10	Operating Systems	143	94	163
Total		1061	949	1990

We select 10 software products from a broad range of application domains to conduct our analysis (Table I). The data collection process was repeated on a daily basis from April 1st to May 27th of 2016. The resulting dataset contained 188, 737 unique tweets. Fig. 1 shows the number of tweets collected per day over the course of our data collection process. Random sampling is used to prepare our dataset. In particular, a Ruby script is used to randomly select 400 tweets from the set of tweets collected for each of our software systems.

B. What's in a tweet?

To get a sense of the information value of our data (i.e., to answer ***RQ1***), the sampled data is manually analyzed. To conduct our analysis, we adopt the standard categorization typically used to classify user reviews in app store mining research [8], [11]. More specifically, we assume that tweets that are addressed to the account of a software system can be classified into two main categories, including technically informative and uninformative messages.

Our manual classification process was performed by three independent industry professional experts with an average of 5 years of experience in software development. Each expert examined each of the sampled tweets in our dataset. A majority vote was taken in cases of a conflict. Conflicts arise in cases that carry a double meaning. For example, the tweet *"@android run app and it dies, any way to know why it dies rather than looking through logs?"* could be classified as a bug report (i.e., *it dies*) or a user requirement (i.e., *any way to*). In total, 121 conflicts were detected in our data ($\approx$ 3%). Table I summarizes our findings[1]. The outcome of our manual classification process can be described as follows:

- **Bug reports:** These tweets report a potential problem with the software. For example, *"@googlechrome I have never ever seen the auto-update function of chrome work on any of all my computers."* and *"@Photoshop When will the biggest Photoshop issue of: lag, freezing, unresponsive Marquee Tool be fixed?"*.
- **User requirements:** These tweets mainly include requests for new features, or alternatively express that a recently added feature is undesirable. For example *"@Snapchat pls make it to where I can see an individual score with someone so I know how many snaps we've sent back & forth !"*. Some requests tend to be less obvious, especially requests for a removed feature to be added back, for example *"@googlechrome any chance I could get my bookmark folders back now please?"*. Such requests can play a crucial rule in release planning as they help developers to decide what features to include/omit in the new release.
- **Miscellaneous and spam:** A considerable part of software-relevant tweets do not provide any useful technical information to the developer. Such tweets might include praise (e.g. *"@VisualStudio is quickly becoming my #goto #dev environment"*), insults (e.g. *"mediocre as usual #meh"*), general information or news (e.g. *"@WhatsApp announced it's started full end-to-end #encryption across its messaging app."*), and spam. Spammers take advantage of the openness and popularity of Twitter to spread unsolicited messages to legitimate users [18]. For instance, spammers tend to post tweets containing typical words of trending topics along with URLs that lead users to completely unrelated websites (e.g. *"#imgur #fix #problem bit.ly/1xTYs"*).

In summary, our manual analysis shows that, out of the 4000 tweets examined, 51% of these tweets were technically informative (27% bug reports and 24% user requirements), while the other 49% were basically spam and miscellaneous information. These findings answer ***RQ1*** and motivate ***RQ2*** and ***RQ3***. In particular, given that around half of our data contain potentially useful information, how can that information be automatically identified and effectively summarized?

IV. Automatic Classification

The second phase of our analysis is concerned with automatically classifying our ground-truth dataset into the different categories of tweets identified earlier. Our research question under this phase (***RQ2***) can be broken down into two sub-questions, first, what classifiers are most effective in the context of Twitter data, and second, what classification features generate the most accurate results.

[1]Data is publicly available at http://seel.cse.lsu.edu/data/re17.zip

A. Classifiers

To answer the first part of ***RQ$_2$***, we investigate the performance of two text classification algorithms, including Naive Bayes (NB) and Support Vector Machines (SVM). SVM and NB have been found to work well with short text. Short-text is a relatively recent Natural Language Processing (NLP) type of text that has been motivated by the explosive growth of micro-blogs on social media (e.g., Tweets and YouTube and Facebook comments) and the urgent need for effective methods to analyze such large amounts of lexically and semantically limited textual data [2], [18], [19]. For instance, Twitter posts are limited to 140 character long messages (*tweets*) and typically contain colloquial terms (e.g., *LOL*, *smh*, *idk*), hyperlinks, Twitter-specific characters such as hashtags (#) and mentions (@), along with phonetic spellings and other neologisms [20]. In detail, NB and SVM can be described as follows:

- **Naive Bayes (NB):** NB is an efficient linear probabilistic classifier that is based on Bayes' theorem [21]. NB assumes the conditional independence of the attributes of the data. In other words, classification features are independent of each other given the class. In the context of text classification, the features of the model can be defined as the individual words of the text. Under this approach, known as the *Bag-of-Words* (BOW), the data is typically represented by a 2-dimensional *word x document* matrix. In the Bernoulli NB model, an entry in the matrix is a binary value that indicates whether the document contains a word or not (i.e., $\{0,1\}$). The Multinomial NB, on the other hand, uses normalized frequencies of the words in the text to construct the *word x document* matrix [22].
- **Support Vector Machines (SVM):** SVM is a supervised machine learning algorithm that is used for classification and regression analysis in multidimensional data spaces [23]. SVM attempts to find optimal hyperplanes for linearly separable patterns in the data and then maximizes the margins around these hyperplanes. Technically, support vectors are the critical instances of the training set that would change the position of the dividing hyperplane if removed. SVM classifies the data by mapping input vectors into an N-dimensional space, and deciding on which side of the defined hyperplane the data instance lies. SVMs have been shown to be effective in domains where the data is sparse and highly dimensional [24].

B. Classification Features

To deal with its unique limited form of text, researchers typically use combinations of Twitter features to help the classifier make more accurate decisions [24], [25], [26]. These features include:

- **Textual Content (BOW)**: Our main classification feature is the words of the tweet. The phrase *Bag-of-Words* (**BOW**), stems from the fact that the text is simply represented as an un-ordered collection of words. Given that Twitter limits messages to 140 characters, a tweet typically has 14 words on average.
- **Text processing**: This set of features includes text reduction strategies such as stemming (**ST**) and stop-word (**SW**) removal. Stemming reduces words to their morphological roots. This leads to a reduction in the number of features (words) as only one base form of the word is considered. Stop-word removal, on the other hand, is concerned with removing English words that are considered too generic (e.g., *the*, *in*, *will*). We further remove words that appear in one data instance (tweet) since they are highly unlikely to carry any generalizable information [2], [27].
- **Sentiment Analysis (SA)**: Sentiment analysis is concerned with determining whether a text conveys positive or negative feelings. Sentiment analysis has been found to play a paramount role in Twitter data analytics [28]. For instance, specific Twitter moods, or sentiments, were found to correlate with public opinion regarding subjects such as consumer confidence and political affiliation, even predicting the movement of the stock market [1], [17]. In our analysis, we assume that a negative sentiment might be associated with a bad experience, such as a system failure or a bad feature. A positive sentiment, on the other hand, might indicate a positive experience [8].

C. Evaluation

To implement NB and SVM, we use Weka [29], a data mining suite that implements a wide variety of machine learning and classification techniques. We also use Weka's built-in stemmer (`IteratedLovinsStemmer` [30]) and stop-word list to preprocess the tweets in our dataset. In our analysis, we use Multinomial NB, which uses the normalized frequency (TF) of words in their documents [22]. Multinomial Naive Bayes is known to be a robust text classifier, consistently outperforming the binary feature model (Multi-variate Bernoulli) in highly diverse, real-world corpora [22]. SVM is invoked through Weka's SMO, which implements John Platt's sequential minimal optimization algorithm for training a support vector classifier [31]. In our analysis, the best results were obtained using the Pearson VII function-based universal kernel (*Puk*) with kernel parameters $\sigma = 8$ and $\omega = 1$ [32]. Universal Kernels are known to be effective for a large class of classification problems, especially for noisy data [33].

Sentiment analysis is performed using Sentistrength [34]. Sentistrength analyzes a document and assigns it two values: a positive sentiment strength and a negative sentiment strength [35], [36]. In particular, the input text is rated by the sentiment content of each word. Most words are neutral, but some words, such as *"loved"* or *"hated"* increase the respective positive or negative sentiment score for their sentence. In our analysis, a text is basically a tweet. For example, the tweet "*@googlechrome really is the best option for someone who enjoys platform agnosticism. All my stuff in @Windows and #OSX*" receives a positive score of 3 and a negative score of 1. The positive score originates from the words *best* (1

point) and *enjoys* (2 points) and the negative score of 1 point is the default score for texts with no negative terminology.

To train our classifiers, we use 10-fold cross validation. This method creates 10 partitions of the dataset such that each partition has 90% of the instances as a training set and 10% as an evaluation set. The benefit of this technique is that it uses all the data for building the model, and the results often exhibit significantly less variance than those of simpler techniques such as the holdout method (e.g., 70% training set, 30% testing set).

Recall, precision, and F-measure are used to evaluate the performance of the different classification techniques used in our analysis. Recall is a measure of coverage. It represents the ratio of correctly classified instances under a specific label to the number of instances in the data space that actually belong to that label. Precision, on the other hand, is a measure of accuracy. It represents the ratio of correctly classified instances under a specific label to the total number of classified instances under that label. Formally, if A is the set of data instances in the data space that belong to the label λ, and B is the set of data instances that were assigned by the classifier to that label, then recall (***R***) can be calculated as $R_\lambda = |A \cap B|/|A|$ and precision (***P***) can be calculated as $P_\lambda = |A \cap B|/|B|$. We also use $F = 2PR/(P + R)$ to measure the harmonic mean of recall and precision.

D. Results and Discussion

The results of our classification process are shown in Table II. In terms of classifiers' accuracy, on average, both SVM and NB were able to achieve competitive results. These results can be explained based on the characteristics of our data space. More specifically, even though Twitter messages are limited in size, the feature space (number of words) is typically very large [24]. This can be attributed to the fact that people use informal language (slang, acronyms, abbreviations) in their tweets. This drastically increases the number of features the classifier needs to process, and also leads the vector representation (BOW) of Twitter messages to be very sparse. While machine learning algorithms tend to over-learn when the dimensionality is high, SVM has an over-fitting avoidance tendency—an inherent behavior of margin maximization which does not depend on the number of features [37]. Therefore, it has the potential to scale up to high-dimensional data spaces with sparse instances. NB tends to be more robust to noise, which seems to work in Twitter data classification, despite the unrealistic conditional independence assumption among classification features [26].

In terms of classification features, the results also show that sentiment scores had almost no impact on performance. This can be attributed to the fact that, unlike political tweets which tend to be very polarized, and typically carry intense emotions [2], software-relevant tweets tend to be neutral. To gain more insight into these results, the boxplots in Fig. 2 show the average combined sentiment score of the different classes of tweets averaged over all our software systems. The figure shows that while bugs tend to be slightly negative (-1), and requirements slightly positive (+1), the difference was not enough to affect the classification accuracy. Furthermore, miscellaneous tweets, while they might carry some extreme emotions, also tend to average out to a neutral state (-1, +1).

TABLE II
SUMMARY OF CLASSIFICATION ACCURACY

	Bug Rep.			User Req.		
Configurations	**P**	**R**	**F**	**P**	**R**	**F**
NB:	0.74	0.77	0.76	0.77	0.58	0.66
NB + ST	0.71	0.80	0.75	0.74	0.58	0.65
NB + SW + ST	0.70	0.75	0.72	0.70	0.54	0.61
NB + Sent.	0.75	0.76	0.75	0.78	0.56	0.65
SVM	0.79	0.75	0.77	0.72	0.60	0.65
SVM + ST	0.78	0.77	0.78	0.72	0.61	0.66
SVM + SW + ST	0.77	0.69	0.72	0.70	0.58	0.63
SVM + Sent.	0.78	0.76	0.77	0.72	0.60	0.66

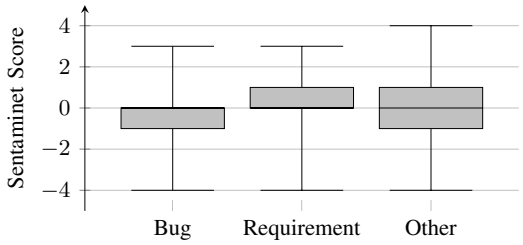

Fig. 2. Sentiment scores for bug reports, user requirements, and other tweets

Our results show that text processing features, such as stemming (**ST**) and stop-word removal (**SW**), produced mixed results for different classifiers. On the one hand, SVM's performance was slightly enhanced when stemming was applied, while NB performance slightly dropped. On the other hand, removing English stop-words seems to have a more noticeable negative impact on the results. In general, stop words seem to add important information value to the classifier. For instance, some of these words (e.g., *would*, *should*, *will*, *don't*, *please*) actually represent distinctive features of user requirements and bug reports (e.g., *"@Snapchat would you please bring the dog filter back?"*). Therefore, removing such words leads to a decline in the classification accuracy.

In summary, to answer ***RQ₂***, our results show that in the context of software relevant tweets, the textual content of Twitter messages is the only contributing factor to the classification accuracy. Other features that are often used as supplemental attributes to enhance Twitter data are irrelevant. The results also confirm previous findings regarding the suitability of SVM and NB as robust classifiers for Twitter data [26].

V. SUMMARIZATION

The third phase of our analysis is focused on generating succinct summaries of the technically useful software tweets. A summary can be described as a short and compact description that encompasses the main theme of a collection of tweets related to a similar topic [38], [39].

A. Tweets Summarization: A Pilot Study

The summarization task in our analysis can be described as a multi-document summarization problem, where each tweet is considered as a separate document. In general, multi-document summarization techniques can be either extractive or abstractive. Extractive methods select specific documents, or keywords, already present within the data as representatives of the entire collection. Abstractive methods, on the other hand, attempt to generate a summary with a proper English narrative from the documents. Generating abstractive summaries can be a very challenging task. It involves heavy reasoning and lexical parsing to paraphrase novel sentences around information extracted from the corpus [40]. This problem becomes more challenging when dealing with the lexically and semantically limited Twitter messages. Therefore, extractive summarization techniques are typically employed to summarize micro-blogging data [41].

To get a sense of how developers would identify the main topics in a list of software-relevant tweets, we conducted a pilot study using two expert programmers with more than 10 years of programming experience each. Each expert was assigned 4 sets of tweets, including 2 sets of bug reporting tweets and 2 sets of user requirement tweets from 4 different systems, including: *Snapchat*, *Chrome*, *Whatsapp*, and *Windows10*. Their task was to go through each set and identify ten tweets that they thought captured the main topics raised in the set. No time constraint was enforced.

Our experts were interviewed after the experiment. Both of them implied that they initially identified the main topics in the tweets after going through the set once or twice. Once these topics were identified (stood out due to their frequent appearance), they selected tweets that included requests (terms and phrases) related to the main topics identified. Our experts were then provided with a word cloud for each set of tweets they were asked to summarize. Each cloud includes the most frequent 30 terms from each set, where more frequent words are drawn in larger font. Our experts were then asked if they thought these clouds were sufficient to convey the main concerns raised in the set. Both experts implied that they preferred to see full tweet summaries over keyword summaries. In general, word-clouds lack context and structure. In contrast, full tweets have the advantage of being full sentences, and thus, can carry more meaningful information [42], [39]. For example, examining the set of user requirement tweets addressed to *Snapchat* shows that they revolve around three main concerns, including users asking for new *filters* to be added, users complaining about the *auto-play* feature of Snapchat stories, and a few tweets raising *portability* concerns (requesting *Snapchat* to work on other devices and platforms). Table III shows examples of the tweets related to the `automatic play` feature of Snapchat stories and the `filter` feature. Extracting these full tweets gives developers a better idea of what the common user concerns actually are. However, only displaying tags, such as in Fig. 3, might be not as informative.

Based on our observations during our pilot study, in our analysis we examine the performance of frequency-based extractive summarization techniques in summarizing software relevant tweets. These techniques rely on the frequencies of words as an indication of their perceived importance [43]. In other words, the likelihood of words appearing in a human-generated summary is positively correlated with their frequency [44]. Formally, a full-tweet extractive summarization process can be described as follows: given a topic keyword or phrase M and the desired length for the summary K, generate a set of representative tweets T with a cardinality of K such that $\forall t_i \in T, M \in t_i$ and $\forall t_i, \forall t_j \in T, t_i \nsim t_j$. The condition $t_i \nsim t_j$ is enforced to ensure that selected tweets provide sufficiently different information (i.e., are not redundant) [44].

In our analysis, we investigate the performance of a number of extractive summarization techniques that have been shown to work well in the context of mirco-blogging data on social media [44], [43], [45], [39]. These techniques include:

- **Hybrid Term Frequency (TF)**: Hybrid TF is the most basic method for determining the importance of a tweet. Formally, a word's w_i value to the summary is computed as the frequency of the word in the entire collection of tweets $f(w_i)$ divided by the number of unique words in the collection (N). This *hybrid* modification over classical single-document TF is necessary to capture concerns that are frequent over the entire collection [44]. The probability of a tweet of length n words to appear in the summary is calculated as the average of the weights of its individual words: $\frac{1}{n}\sum_{i=1}^{n} f(w_i)/N$.
- **Hybrid TF.IDF**: Introduced by Inouye and Kalita [44], the hybrid TF.IDF approach is a frequency-based summarization technique that is designed to summarize social media data. Hybrid TF.IDF accounts for a word's scarcity across all the tweets by using the inverse document frequency (IDF) of the word. IDF penalizes words that are too frequent in the text. Formally, TF.IDF can be computed as:

$$TF.IDF = TF(w_i) \times \log \frac{|D|}{|d_j : w_i \in d_j \wedge d_j \in D|} \quad (1)$$

  where $TF(w_i)$ is the term frequency of the word w_i in the entire collection, $|D|$ is the total number of tweets in the collection, and $|d_j : w_i \in d_j \wedge d_j \in D|$ is the number of tweets in *D* that contain the word w_i. The importance of a tweet can then be calculated as the average TF.IDF score of its individual words.

  To control for redundancy, or the chances of two very similar tweets getting selected, before adding a top tweet to the summary, the algorithm makes sure that the tweet does not have a textual similarity above a certain threshold with the tweets already in the summary. Similarity is calculated using the cosine between the vector representations of tweets.
- **SumBasic**: Introduced by Nenkova and Vanderwende [43], SumBasic uses the average term frequency (TF) of tweets' words to determine their value.

TABLE III
EXAMPLE OF REQUIREMENT TWEETS RELATED TO SIMILAR FEATURES FROM SNAPCHAT'S TWITTER FEED

Tweets related to the feature ``filter''	Tweets related to the feature ``automatic play''
"where is my dog filter"	*"change the way you view stories back to the original way?"*
"can you bring back the bunny face filter pls"	*"I do not want to automatically watch people's stories!!"*
"more arty filters like the one today"	*"hate the stories autoplay feature"*

Fig. 3. A word cloud summary of Snapchat's user requirement tweets

However, the weight of individual words is updated after the selection of a tweet to minimize redundancy. This approach can be described as follows:

1) The probability of a word w_i in the input corpus of size N words is calculated as $\rho(w_i) = f(w_i)/N$, where $f(w_i)$ is the frequency of the word in the entire corpus.
2) The weight of a tweet of length n words is calculated as the average probability of its words, given by: $\frac{1}{n}\sum_{i=1}^{|n|}\rho(w_i)$
3) The best scoring tweet is selected. For each word in the selected tweet, its probability is reduced by $\rho(w_i)_{new} = \rho(w_i) \times \rho(w_i)$. This step is necessary to control for redundancy, or minimize the chances of selecting tweets describing the same topic with high frequency words.
4) Repeat from 2 until the required length of the summary is met.

B. Evaluation

We recruited 10 programmers (experts) to participate in our experiment, including 3 graduate students in computer science and 7 industry professionals. Our experts have reported an average of 6 years of programming experience. 5 systems from Table I were randomly selected to conduct our experiment. These systems include: *Chrome*, *Minecraft*, *SnapChat*, *Whatsapp*, and *Windows 10*. Each of our experts was assigned 2 different systems to summarize, such that, each system is summarized by exactly 4 different experts. For each system we provided two sets of tweets, including the set of bug reporting tweets and the set of user requirement tweets. The main task of the expert was to go through each set and identify 10 tweets that they believed captured the common concerns raised in the set. The tweets in each set were randomized ahead of time to avoid any ranking bias (e.g., an expert would always favor tweets from the top of the list). No time constraint was enforced. However, most of our participants responded within a one week period.

The various summarization techniques proposed earlier were then used to generate the automated summaries for the 5 systems included in our experiment. To enhance the quality of the generated summaries, English stop-words were excluded from our frequency analysis. Stemming was also applied to minimize the redundancy imposed by the usage of different variations of words (e.g., *show, showing, shown, and shows*). To assess the quality of these summaries, for each system, we calculate the average term overlap between our experts' selected lists of tweets (reference summaries) and the various automatically generated summaries. Formally, a recall of a summarization technique t is calculated as:

$$Recall_t = \frac{1}{|S|}\sum_{i=1}^{|S|}\frac{match(t, s_i)}{count(s_i)} \quad (2)$$

where S is the number of reference summaries, $match(t, s_i)$ is the number of terms that appear in the reference summary s_i and the automated summary generated by t, and $count(s_i)$ is the number of unique terms in the reference summary s_i. An automated summary that contains (recalled) a greater number of terms from the reference summary is considered more effective [46], [44], [43]. In our analysis, recall is measured over different length summaries (5, 10, 15, and 20 tweets included in the summary).

C. Results and Discussion

The recall of the different summarization techniques is shown in Fig. 4 and Fig. 5. Randomly generated summaries (tweets were selected randomly from each set using the `.NET Random` class) were used to compare the performance of our proposed methods. Our results show that all methods outperformed the random baseline. On average, SumBasic was more successful than hybrid TF.IDF and TF in summarizing the common concerns found in software-relevant tweets. TF achieved the poorest performance, suggesting that redundancy

control is important in order to achieve comprehensive summaries. The results also show that TF was only slightly outperformed by hybrid TF.IDF. Due to the limited nature of the documents in our corpus (i.e., individual tweets), the IDF part seems to have a limited impact on the results as it is typically dominated by TF.

The better performance of SumBasic in comparison to hybrid TF.IDF can be explained based on their underlying redundancy control mechanisms. SumBasic tends to be more forgiving for redundant terms as it avoids excessive redundancy while also allowing common words to occasionally repeat in the summary. This can be useful in cases where there is relatively high redundancy in the data. Hybrid TF.IDF, by enforcing a similarity threshold on the entire tweet rather than individual words, can exclude important concerns from the summary. More specifically, changing the redundancy threshold can lead to extreme changes in the summaries (excluding a large number of tweets or hardly any). Finding an optimal threshold that works for all cases can be an exhaustive task especially that different datasets might require different thresholds. For instance, the best hybrid TF.IDF recall in our analysis was observed at similarity thresholds of 0.65 for bug reports and 0.50 for user requirements.

It is important to point out that more computationally expensive techniques such as Latent Dirichlet Allocation (LDA) [47] and cluster-based summarization have been employed in the literature to summarize Twitter data [39]. However, due to the lack of semantic structure in Twitter posts, such complex relational models were reported to be ineffective in capturing topical information. Furthermore, such techniques typically require heavy calibration of multiple parameters in order to generate decent results. This limits the practicality of such techniques and their ability to produce meaningful summaries [42], [38]. Frequency-based techniques, on the other hand, are computationally inexpensive and relatively easier to implement and calibrate. This aspect can be crucial to achieve an easy transfer of our research findings to practice.

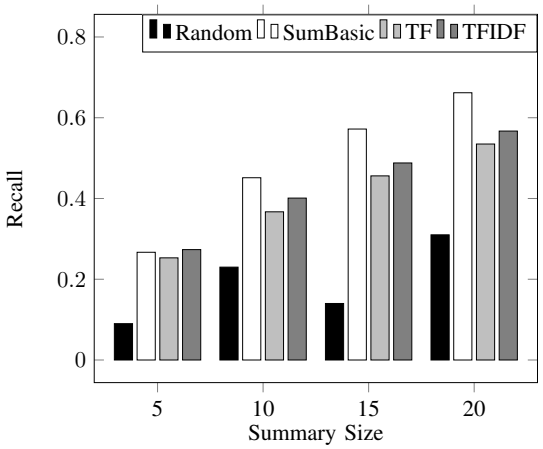

Fig. 4. Performance of different summarization techniques over bug reporting tweets measured at different length summaries (5, 10, 15, 20)

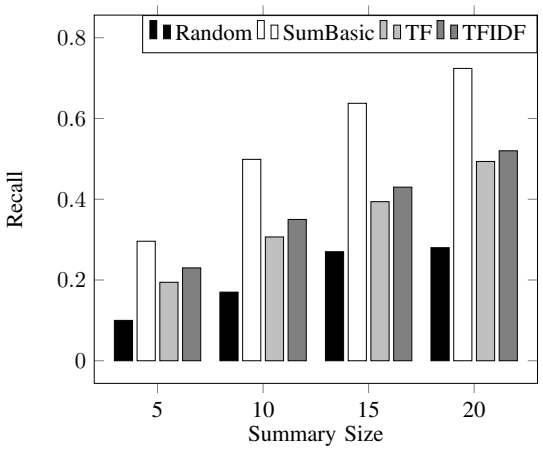

Fig. 5. Performance of different summarization techniques over user requirements tweets measured at different length summaries (5, 10, 15, 20)

VI. Threats to Validity

The study presented in this paper has several limitations that might affect the validity of the results [48]. A potential threat to the proposed study's internal validity is the fact that human judgment is used to classify and summarize our sample tweets and prepare our ground-truth dataset. This might result in an experimental bias as humans tend to be subjective in their judgment. However, it is not uncommon in text classification to use humans to manually classify the data, especially in social media classification [14], [15]. Similarly, evaluating machine-generated against human-generated summaries is a standard evaluation procedure. Therefore, while the subjectivity and bias threats that stem from using humans are inevitable, they can be partially mitigated by using multiple judges at different levels of expertise.

In our experiment, there were minimal threats to construct validity as the standard performance measures (Recall, Precision), which are extensively used in related research, were used to assess the performance of different methods.

Threats to external validity impact the generalizability of results [48]. A potential threat to our external validity stems from the fact that our dataset is limited in size and was generated from a limited number of software systems and tweets. To mitigate this threat, we ensured that our dataset was compiled from a wide variety of application domains. Furthermore, we used randomization to sample 400 tweets from the set of tweets collected for each system. While considering all the data instances in the analysis might enhance the validity of our results, manually analyzing such large amounts of data can be a tedious and error-prone task, and as a result, research on Twitter data analytics is often conducted using partial datasets. Other threats might stem from the tools we used in our analysis. For instance, we used Weka as our machine learning and classification platform and Sentistrength was used to tag our tweets' sentiment. Nonetheless, such tools have

been extensively used in the literature and have been shown to generate robust results across a plethora of applications. Furthermore, using such publicly available benchmark tools enables other researchers to replicate our results.

Finally, it is unclear whether our approach will be as successful for systems that are less widely used. More specifically, there is no guarantee that such systems will have enough tweets that offer meaningful data for developers. In such cases, other sources of user feedback, such as app store reviews and online surveys, can be used to paint a full picture. Another concern stems from the fact that Twitter is often used for advertisement purposes by software vendors. Many tweets can be simply marketing messages from the vendor or competitors, rather than feedback from users. Therefore, there is no guarantee that the technically informative tweets captured by our approach are indeed from the right audience and not just containing publicity which accidentally describes as a requirement or a bug report.

VII. Related Work

In software engineering, the research on mining microblogging services has focused on the way developers use such platforms to share and exchange software development information. For instance, Bougie et al. [14] manually analyzed 600 tweets from three different software engineering communities. The results showed that developers' tweets tend to include more conversation and information sharing in comparison to non-technical populations of Twitter users.

Tian et al. [15] manually analyzed a sample of 300 developer tweets in various software engineering communities. The results showed that such tweets commonly contain job openings, news, questions and answers, or links to download new tools and code. In a follow-up study, Prasetyo et al. [49] investigated the feasibility of automatically classifying tweets as relevant and irrelevant to software developers in engineering software systems. The authors used SVM to classify the data in [15]. The results showed that 47% of the classified tweets were found to be relevant to developers.

Singer et al. [13] surveyed 271 and interviewed 27 active GitHub developers about using Twitter in their development and interaction activities. The authors reported that developers use Twitter mainly to stay aware of industry changes, for learning, and for building relationships. Sharma et al. [12] proposed an approach to help developers to identify software relevant tweets. Individual tweets were assigned a relevance probability based on their similarity to a language model generated from a subset of posts from *StackOverflow*. Evaluating the proposed approach over a random sample of 200 tweets showed improvement over previous models that use classification and keyword dictionaries.

Initial exploratory work on the value of Twitter data for requirements engineers was proposed by Guzman et al. [4]. The authors manually analyzed and classified a sample of 1,000 tweets to determine the usage characteristics and content of software-relevant tweets. The results showed that software tweets contained useful information for different groups of technical and non-technical stakeholders. While our work builds upon this work, in our analysis we only focused on feedback dedicated to the technical stakeholders of the system (developers) by limiting data collection to tweets directly addressed to the Twitter accounts of our sample systems. This constraint enabled us to obtain higher accuracy levels. For instance, qualitative analysis of our 4,000 tweets showed that around 50% of our data contained useful technical feedback, in comparison to only 19% in [4]. We were also able to achieve an average classification F_1 of 72% using SVM, in comparison to an F_1 of 48% achieved in [4].

VIII. Conclusions and Future Work

This paper presents a three-fold procedure aimed at leveraging Twitter as a main source of technical software information. These phases include data collection, classification, and presentation. Our analysis is conduced using 4,000 tweets sampled from tweets addressed to 10 software systems from a broad range of application domains. A manual qualitative analysis was conducted to determine the information value of sampled tweets. Our results showed that around 50% of these tweets contained useful technical data that can be translated into actionable bug reports and user requirements (***RQ1***). Our analysis also showed that SVM and NB can be effective in capturing and categorizing technically useful tweets. In terms of classification features, our results showed that in the context of software-relevant tweets, sentiment analysis seems to have no impact on performance, while text reduction strategies had a conflicting impact on the classification accuracy (***RQ2***).

Technically informative tweets were then summarized using multiple automated summarization algorithms. These algorithms, including hybrid TF, hybrid TF.IDF and SumBasic, are known for their simplicity (implementation, calibration, and computation overhead) and decent performance in the context of social media data. A human experiment using 10 programmers was conducted to assess the performance of the different summarization techniques. The results showed that the summarization algorithm SumBasic was the most successful in recalling majority of the common concerns raised in software-relevant tweets (***RQ3***).

The line of work in this paper will be expanded along several directions as follows:

- Data collection: A main part of our future effort will be devoted to collecting larger datasets from a more diverse set of software systems. More data will enable us to conduct in depth analysis of software users' tweeting patterns, and thus draw more robust conclusions.
- Analysis: Our future work will include experimenting with more advanced text classification and summarization techniques to achieve higher levels of accuracy.
- Tool support: A working prototype that implements our findings in this paper will be developed. This prototype will enable developers to classify and summarize tweets related to their systems in an effective and accurate manner.

ACKNOWLEDGMENT

This work was supported by the Louisiana Board of Regents Research Competitiveness Subprogram (LA BoR-RCS), contract number: LEQSF(2015-18)-RD-A-07.

REFERENCES

[1] J. Bollen, H. Mao, and X. Zeng, "Twitter mood predicts the stock market," *Journal of Computational Science*, vol. 2, no. 1, pp. 1–8, 2011.

[2] M. Conover, B. Goncalves, J. Ratkiewicz, A. Flammini, and F. Mencze, "Political polarization on twitter," in *AAAI Conference on Weblogs and Social Media*, 2011, pp. 89–96.

[3] P. Dodds, K. Harris, I. Kloumann, C. Bliss, and C. Danforth, "Temporal patterns of happiness and information in a global social network: Hedonometrics and Twitter," *PLoS ONE*, vol. 6, no. 12, 2011.

[4] E. Guzman, R. Alkadhi, and N. Seyff, "A needle in a haystack: What do twitter users say about software?" in *International Requirements Engineering Conference*, 2016, pp. 96–105.

[5] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta, "Release planning of mobile apps based on user reviews," in *International Conference on Software Engineering*, 2016, pp. 14–24.

[6] R. Basole and J. Karla, "Value transformation in the mobile service ecosystem: A study of app store emergence and growth," *Service Science*, vol. 4, no. 1, pp. 24–41, 2012.

[7] M. Bano and D. Zowghi, "A systematic review on the relationship between user involvement and system success," *Information and Software Technology*, vol. 58, pp. 148–169‘, 2015.

[8] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *Requirements Engineering Conference*, 2013, pp. 125–134.

[9] N. Chen, J. Lin, S. Hoi, X. Xiao, and B. Zhang, "Ar-miner: mining informative reviews for developers from mobile app marketplace," in *International Conference on Software Engineering*, 2014, pp. 767–778.

[10] G. Carreńo and K. Winbladh, "Analysis of user comments: An approach for software requirements evolution," in *International Conference on Software Engineering*, 2013, pp. 343–348.

[11] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? On automatically classifying app reviews," in *Requirements Engineering Conference*, 2015, pp. 116–125.

[12] A. Sharma, Y. Tian, and D. Lo, "Nirmal: Automatic identification of software relevant tweets leveraging language model," in *International Conference on Software Analysis, Evolution, and Reengineering*, 449–458, p. 2015.

[13] L. Singer, F. Filho, and M. Storey, "Software engineering at the speed of light: How developers stay current using Twitter," in *International Conference on Software Engineering*, 2014, pp. 211–221.

[14] G. Bougie, J. Starke, M. Storey, and D. German, "Towards understanding Twitter use in software engineering: Preliminary findings, ongoing challenges and future questions," in *International Workshop on Web 2.0 for Software Engineering*, 2011, pp. 31–36.

[15] Y. Tian, P. Achananuparp, I. Lubis, D. Lo, and E. Lim, "What does software engineering community microblog about?" in *Working Conference on Mining Software Repositories*, 2012, pp. 247–250.

[16] "Twitter developer API," https://dev.twitter.com/, accessed: August, 15th.

[17] B. O'Connor, R. Balasubramanyan, B. Routledge, and N. Smith, "From tweets to polls: Linking text sentiment to public opinion time series," in *AAAI Conference on Weblogs and Social Media*, 2010, pp. 122–129.

[18] M. McCord and M. Chuah, "Spam detection on Twitter using traditional classifiers," in *international conference on Autonomic and trusted computing*, 2011, pp. 175–186.

[19] A. Wang, "Don't follow me: Spam detection in Twitter," in *International Conference on Security and Cryptography*, 2010, pp. 1–10.

[20] L. Squires, "Enregistering internet language," *Language in Society*, no. 39, pp. 457–492, 2010.

[21] P. Langley, W. Iba, and K. Thompson, "An analysis of Bayesian classifiers," in *Artificial Intelligence*, 1992, pp. 223–228.

[22] A. McCallum and K. Nigam, "A comparison of event models for Naive Bayes text classification," in *AAAI - Learning for Text Categorization*, 1998, pp. 41–48.

[23] C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[24] T. Joachims, "Text categorization with suport vector machines: Learning with many relevant features," in *European Conference on Machine Learning*, 1998, pp. 137–142.

[25] B. Sriram, D. Fuhry, E. Demir, H. Ferhatosmanoglu, and M. Demirbas, "Short text classification in Twitter to improve information filtering," in *International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2010, pp. 841–842.

[26] S. Wang and C. Manning, "Baselines and bigrams: Simple, good sentiment and topic classification," in *Annual Meeting of the Association for Computational Linguistics*, 2012, pp. 90–94.

[27] J. Grimmer and B. Stewart, "Text as data: The promise and pitfalls of automatic content analysis methods for political texts," *Political Analysis*, vol. 21, pp. 267–297, 2013.

[28] A. Pak and P. Paroubek, "Twitter as a corpus for sentiment analysis and opinion mining," in *Language Resources and Evaluation Conference*, 2010, pp. 1320–1326.

[29] "Weka 3: Data mining software in java," http://www.cs.waikato.ac.nz/ ml/weka/, accessed: August, 15th.

[30] J. Lovins, "Development of a stemming algorithm," *Mechanical Translation and Computational Linguistics*, vol. 11, pp. 22–31, 1968.

[31] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods - Support Vector Learning*, B. Schoelkopf, C. Burges, and A. Smola, Eds. MIT Press, 1998.

[32] B. Üstün, W. Melssen, and L. Buydens., "Facilitating the application of support vector regression by using a universal pearson vii function based kernel." *Chemometrics and Intelligent Laboratory Systems*, vol. 81, pp. 29–40, 2006.

[33] I. Steinwart, "On the influence of the kernel on the consistency of support vector machines," *Journal of Machine Learning Research*, vol. 2, pp. 67–93, 2001.

[34] "Sentistrength," http://sentistrength.wlv.ac.uk/, accessed: August, 15th.

[35] D. Vilares, M. Thelwall, and M. Alonso, "The megaphone of the people? Spanish SentiStrength for real-time analysis of political tweets," *Journal of Information Science*, vol. 41, no. 6, pp. 799–813, 2015.

[36] F. Nielsen, "A new ANEW: Evaluation of a word list for sentiment analysis in microblogs," in *ESWC2011 Workshop on 'Making Sense of Microposts': Big things come in small packages*, 2011, pp. 93–98.

[37] P. Brusilovsky, A. Kobsa, and W. Nejdl, Eds., *The Adaptive Web: Methods and Strategies of Web Personalization*. Springer-Verlag, 2007.

[38] C. Llewellyn, C. Grover, and J. Oberlander, "Summarizing newspaper comments," in *International Conference on Weblogs and Social Media*, 2014, pp. 599–602.

[39] E. Khabiri, J. Caverlee, and C. Hsu, "Summarizing user-contributed comments," in *AAAI Conference on Weblogs and Social Media*, 2011, pp. 534–537.

[40] U. Hahn and I. Mani, "The challenges of automatic summarization," *Computer*, vol. 33, no. 11, pp. 29–36, 2000.

[41] J. Nichols, J. Mahmud, and C. Drews, "Summarizing sporting events using twitter," in *ACM International Conference on Intelligent User Interfaces*, 2012, pp. 189–198.

[42] E. Barker, M. Paramita, A. Funk, E. Kurtic, A. Aker, J. Foster, M. Hepple, and R. Gaizauskas, "What's the issue here?: Task-based evaluation of reader comment summarization systems," in *International Conference on Language Resources and Evaluation*, 2016, pp. 23–28.

[43] A. Nenkova and L. Vanderwende, "The impact of frequency on summarization," Microsoft Research, Tech. Rep., 2005.

[44] D. Inouye and J. Kalita, "Comparing Twitter summarization algorithms for multiple post summaries," in *International Conference on Social Computing (SocialCom) and International Conference on Privacy, Security, Risk and Trust (PASSAT)*, 2011, pp. 298–306.

[45] E. Momeni, C. Cardie, and M. Ott, "Properties, prediction, and prevalence of useful user-generated comments for descriptive annotation of social media objects," in *AAAI Conference on Weblogs and Social Media*, 2013, pp. 390–399.

[46] C. Lin, "Rouge: A package for automatic evaluation of summaries," in *Workshop on Text Summarization Branches Out*, 2004, pp. 74–81.

[47] D. Blei, A. Ng, and M. Jordan, "Latent Dirichlet Allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[48] A. Dean and D. Voss, *Design and Analysis of Experiments*. Springer, 1999.

[49] P. Prasetyo, D. Lo, P. Achananuparp, Y. Tian, and E. Lim, "Automatic classification of software related microblogs," in *International Conference on Software Maintenance*, 2012, pp. 596–599.