Mining Software Features from App Store Artifacts: A Pattern-Based Approach to Feature Extraction and Matching

NaumanIrshadAliShah[*1], Ali Ahmad[2], Muhammad Umer Amir[3]
[1,2,3] Department of Computer Science, University of Central Punjab, Lahore, Pakistan
[*1]naumanirshadalishah@gmail.com, [2]aliahmaddev61@gmail.com, [3]umer.html@gmail.com

Keywords
User Reviews,
App Store Analytics,
Software Feature,
Data Mining,
Data-Driven Requirements,
Feature Extraction,
NLP,
Pattern Recognition

Abstract

This paper presents SAFE, a novel uniform approach to extract app features from single app pages, single reviews and to match them. A main advantage of app stores is that they aggregate important information created by both developers and users. In the app store product pages, developers usually describe and maintain the features of their apps. In the app reviews, users comment these features. Recent studies focused on mining app features either as described by developers or as reviewed by users. However, extracting and matching the features from the app descriptions and the reviews is essential to bear the app store advantages, e.g. allowing analysts to identify which app features are actually being reviewed and which are not. We manually build 18 part-of-speech patterns and 5 sentence patterns that are frequently used in text referring to app features. We then apply these patterns with several text pre- and post-processing steps. A major advantage of our approach is that it does not require large training and configuration data. To evaluate its accuracy, we manually extracted the features mentioned in the pages and reviews of 10 apps. The extraction precision and recall outperformed two state-of-the-art approaches. For well-maintained app pages such as for Google Drive our approach has a precision of 87% and on average 56% for 10 evaluated apps. SAFE also matches 87% of the features extracted from user reviews to those extracted from the app descriptions.

1 INTRODUCTION

App stores are particularly interesting to the software and requirements engineering community for two main reasons. First, they include millions of apps with the possibility to access hundreds of millions of users [1]. Second, aggregate information from customer, business, and technical perspectives [2]. Each app in the store is presented by its own app page. This typically includes the app name, icons, screenshots, previews, and the app description as written and maintained by the developers. Users relay on this information to find and download the app they are looking for. Later, some users review the app and comment on its features and limitations. Some apps might receive even more than a thousand reviews per day [1], some of which include hints and insights for the developers.

Over the past years, we observed a ”boom” of research papers, tools, and projects on app store analytics [3]. Most of these works focus on mining large amount of app store data to derive advice for

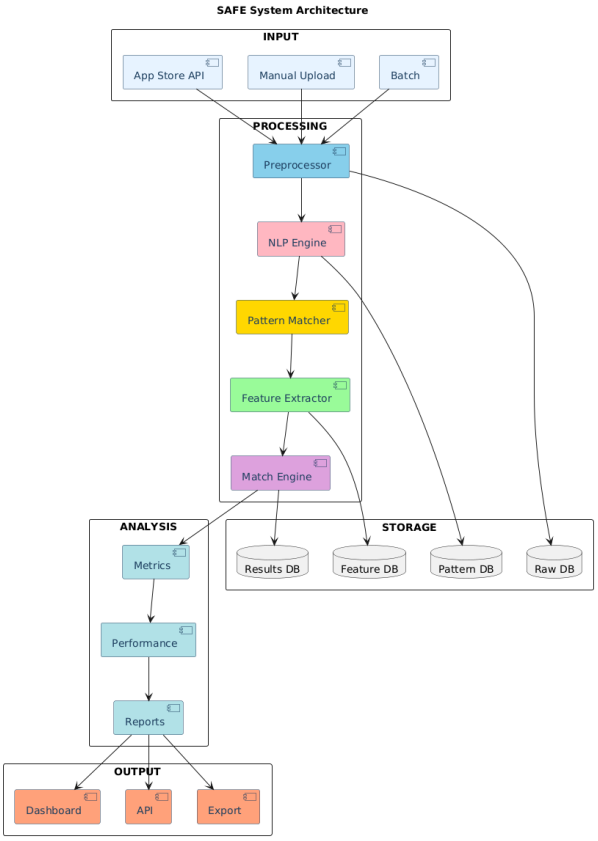

Figure 1: SAFE System Architecture showing the complete pipeline from input to feature extraction and matching

analysts, developers, and users. One popular analytic scenario is to mine app reviews for identifying popular user complaints or requests [4–6]. Another scenario is to mine the app pages [7,8] and their evolution over time [9] to derive recommendations for users or to identify combinations that increase download numbers [10, 11].

A common and elementary step that these approaches share is the following: how can the app features included in the natural language text be automatically and accurately identified, in particular since the text is unstructured, potentially noisy, and uses different vocabulary. Previous approaches focus on mining the features from a particular artifact in the store, typically either from the pages or from the review. However, app vendors are likely interested in a holistic approach that works for multiple types of artifacts to be able to combine the customer, business, and technical information.

In this paper, we introduce SAFE, a simple, uniform approach to extract and match the app features from the app descriptions (written by developers) and the app reviews (commented by users). Our approach neither requires an a-priori training with a large dataset nor configuration of machine learning features and parameters and works on single text elements such as a single app review. We simply identify, based on a deep manual analysis, a set of general textual patterns that are frequently used in app stores to denote app features. This includes a) 18 part-of-speech patterns (such as Noun_Noun_Noun to represent a feature like ”Email chat history”), b) five sentence patterns (indicating enumerations and conjunctions of features), and c) several noise filtering patterns (such as filtering contact information). After preprocessing the text in a single app description or a review, we apply those patterns and extract a list of features: each consists of two to four keywords. Finally, we match the extracted features from the reviews to those extracted from the descriptions.

2 THE SAFE APPROACH

Features have manifold definitions in Requirements Engineering. According to Wieger and Beatty a feature denotes ”one or more logically related system capabilities that provide value to a user and are described by a set of functional requirements” [12]. Kang et al. define a feature as ”a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems” [13]. Harman et al. focus on apps and state ”a feature to be a property, captured by a set of words in the app description and shared by a set of apps” [8].

Features of a software represent an important way to communicate its capabilities to internal and external stakeholders. Features can help users to understand what the software is about and what they can and can’t do with it. Features are also important for developers and vendors to structure their software projects and artifacts. They are frequently used in release planning and in issue trackers. Features can also be used in requirements and design artifacts and are also often traceable to source code and APIs.

We focus on high-level features that describe the essential functional capabilities of software and which are both understandable to developers and users. For instance, the app Dropbox has the fol-

lowing features described in its app page: ”back up files”, ”send large files”, ”access files online”, ”access files offline”, ”create microsoft office files”, ”edit microsoft office files”, ”share links to files”, and ”includes storage”. In contrast, low-level features contain only a set of words e.g. Internet, Wi-Fi, Connectivity or photo, picture, upload.

There is no uniform way how developers describe the features of their app, but we recognized some patterns. Descriptions often contain continuous texts, which describe features and purposes of an app. In addition, features are often summarized in a bullet point list. In app pages, features from the continuous texts are often repeated within the bullet points.

We exclude games from our analysis (and the evaluation) since we assume that they represent a special type of software [14], where requirements and features are described and handled differently. Murphy-Hill et al. [15] studied game development at Microsoft and found that game developers are ”designing [...] an emotional experience [...] which is very subjective [...] and is an artistic achievement.” The authors gave the example: ”in an e-commerce application, a user has a task to complete that typically takes only a few minutes. In contrast, in some games people play for hours straight on a daily basis over the course of months. As a result, the requirement for games is that the user should be able to stay engaged on multiple timescales, and the mechanism to achieve that will vary from game to game”.

Our approach includes three main types of patterns to extract those features from app pages and app reviews. First, it defines a set of text patterns that are frequently used to describe features. We have identified these patterns based on a manual analysis of real app pages and reviews. Second, we apply these patterns on app pages and app reviews with some pre- and post-processing of the text to identify the features. Finally, we use text similarity algorithms to match the extracted features from multiple app artifacts.

2.1 Identifying the SAFE Patterns

The SAFE patterns consist of two types of text patterns that are frequently used to describe features in app stores: SAFE Part-of-Speech (POS) patterns and SAFE sentence patterns.

2.1.1 SAFE POS Patterns

In a first step, we analyzed app descriptions to identify commonly used POS patterns to denote app features. For this, we crawled the descriptions of 100 apps from the Google Play store. We then POS-tagged the text using the Natural language Tool Kit (NLTK) and extracted all sentences. Afterwards, we rendered each sentence in a tool. One researcher manually looked at the sentences to find those patterns. Table 1 summarizes the most frequent POS patterns in our sample that we use in SAFE. We cut off all POS patterns that occurred less than 10 times. By far the most frequently used SAFE POS patterns include two words. The remaining patterns include up to four words such as ”choose from popular versions”.

2.1.2 SAFE Sentence Patterns

Existing approaches often miss app features, if a single sentence contains a lot of independent features. For example, the sentence: ”send and receive images, videos and sticker” contains the following six app features: ”send images”, ”send videos”, ”send stickers”, ”receive images”, ”receive videos”, and ”receive stickers”. As language has a high ambiguity, we do not cover all possible cases but those we found frequently in the app descriptions. In order to extract all app features we analyze the sentence structure and handle five different sentence patterns illustrated in the following with real examples from app descriptions:

Case 1: ”send and receive attachments”

Case 2: ”View documents, PDFs, photos, and videos”

Case 3: ”Discuss and annotate notes and drafts”

Case 4: ”Use camera capture to easily scan and comment on pieces of paper, including printed documents, business cards, handwriting and sketches”

Case 5: ”Write, collect and capture ideas as searchable notes, notebooks, checklists and to-do lists”

SAFE identifies enumerations (comma separated text), conjunctions, and feature identifiers (words might be used to construct a feature) within each sentence. Then SAFE searches for occurrences and the placement of conjunctions within the sentence.

Table 1: Overview of the SAFE POS Patterns

#	POS Pattern	Freq.	Example
1	Noun Noun	183	Group conversation
2	Verb Noun	122	Send message
3	Adjective Noun	119	Precise location
4	Noun Conjunction Noun	98	Phone or tablet
5	Adjective Noun Noun	70	Live traffic conditions
6	Noun Noun Noun	35	Email chat history
7	Verb Pronoun Noun	29	Share your thoughts
8	Verb Noun Noun	28	Enjoy group conversations
9	Verb Adjective Noun	26	Perform intuitive gestures
10	Adjective Adjective Noun	20	Super bright flashlight
11	Noun Preposition Noun	18	Highlight with colors
12	Verb Determiner Noun	14	Share an image
13	Verb Noun Preposition Noun	14	Use depth of field
14	Adjective Noun Noun Noun	12	Fast system virus scanner
15	Adjective Conjunction Adjective	12	Pre-installed and user-installed
16	Verb Preposition Adjective Noun	11	Choose from popular versions
17	Verb Pronoun Adjective Noun	11	Create your greatest album
18	Noun Conjunction Noun Noun	10	Song and artist album

Afterwards, the text parts of the left and the right side of each conjunction are analyzed to identify the placement of possible enumerations and feature identifiers. Finally, the feature identifiers are combined with the remaining list of app features.

2.2 Automated Feature Extraction

The main steps of SAFE are:

1. Text Preprocessing: The input of the preprocessing is either a single app description or a single review. First, SAFE performs a sentence tokenization. Then, all brackets and the text in between are removed as we observed that app features are prominently stated and that text in brackets often simply contains additional explanations. Next, SAFE filters three types of sentences: 1) sentences that contain URLs and 2) sentences that contain email addresses, as those are used to provide contact information, and 3) sentences that contain quotations, as they are most likely to be quoted reviews. Those types of sentences are removed from the data set. Further, in the remaining sentences such as bullet points and multiple symbols (such as '*' or '#') are removed. These are often used to visually delineate different parts of the descriptions.

Within the clean sentences, SAFE searches for keywords that introduce a subordinate clause. We cut off subordinate clauses, because they do not describe features (what the app is doing) but give reasons of why they are useful (how the app is).

2. Application of SAFE Patterns: This part starts with analyzing and decomposing the sentences based on the conjunctions, enumerations, and feature identifiers. Then, SAFE applies the sentence patterns on the decomposed sentences to extract raw app features. The raw app features as well as the remaining sentences that did not match any sentence patterns represent the input for the next step, in which the SAFE POS patterns from Table 1 are used to extract a list of feature candidates. Finally, SAFE iterates through all extracted feature candidates, removes duplicates and noise such as identical word pairs (e.g. "email email" which matches the SAFE POS pattern Noun Noun).

2.3 Automated Feature Matching

The input of this step is a list of feature candidates extracted from a single app description as well as the extracted feature candidates from the related user reviews. SAFE matches the feature candidates from both, the app description and the related user reviews. The matching is based on a binary text similarity function at sentence level that is inspired by approaches presented by Guzman and Maalej [14] and Li et al. [16].

Overall, SAFE performs three steps, each repre-

senting a different approach to identify matching features, starting with the most restrictive approach. First, SAFE performs the matching on a single term level. This means, that two feature candidates are matching if each single term of them is equal, ignoring the order of the terms. For instance, ”send email” and ”email send” are considered matching features. Second, SAFE uses the synonym sets of the words captured from WordNet to perform the second step of finding matching features. In this approach two app features are considered a match, if their terms are synonyms. For instance, ”take photo” and ”capture image” represent a match. However, the accuracy of this step declines when the number of words of both candidate features is not equal. Hence, we also use a semantic similarity matching at sentence level introduced by Yuhua Li et al. [16] as our third step. This approach incorporates two similarity metrics: on the one hand, the semantic similarity employing statistical characteristics of a lexical corpus that can be inferred as a domain knowledge base; on the other hand, the order similarity of word order vectors extracted from the sentences. Since we assume that the ordering of the words does not affect the meaning of the features, we skip the word order similarity metric.

Finally, to deal with the candidate features that have unequal number of words, we utilized cosine similarity calculated between two feature candidates through the semantic similarity algorithm introduced by Li et al. [16]. We set the threshold of the similarity to .7 based on experiments we performed while creating the feature matching approach. Correlation factors between .7 and 1 are considered significant. Some examples of the calculated similarities of candidate features are shown on Table 2.

Table 2: Cosine Similarity of Exemplary Candidate Features

Feature 1	Feature 2	Cosine Similarity
compose new email	create new email	0.85
taking image	capture image	0.73
send attachments	send attached files	0.63
add image	delete image	0.38
send new email	receive emails	0.35

3 EMPIRICAL EVALUATION

3.1 Evaluation Goals and Data

We implemented and tested SAFE as a python library and conducted an empirical evaluation to assess the approach effectiveness and accuracy to extract and match features. We focused on the following evaluation questions:

1. How precise is SAFE when extracting features from app descriptions and from app reviews?
2. How does SAFE perform compared to other state-of-the-art approaches?
3. How accurate can SAFE match the extracted features from the app descriptions and the reviews?

For the evaluation, we created a dataset by crawling descriptions and reviews of 10 apps from the Apple App Store. We selected the top five free and top five paid apps (January 2017) from the category Productivity of the Apple App Store. We chose the storefront of the United States to obtain reviews in English. For each app, we gathered its description page using the iTunes Search API. From the app description page, we extracted metadata consisting of 44 values, such as the name, version, description, category, or price. The app reviews were programmatically scraped using a self-developed tool, which accesses an internal iTunes API. A review consists of 10 values, including the reviewer name, title, description, rating, and date.

We also reimplemented two recent, frequently cited approaches for extracting features from app store data: the approach of Harman et al. [8] (also used by Al-Subaihin et al. [17]) focuses on app descriptions and the approach by Guzman and Maalej [14] focuses on app reviews.

3.2 Evaluation Results

3.2.1 Feature Extraction from the App Descriptions

We evaluated the features extracted from the descriptions by our approach and by the approach of Harman et al. SAFE extracts features from the whole description, whereas Harman et al.’s approach only uses texts from the bullet lists of the app page. Table 3 shows the results. The evaluation is based on ten apps and contains 197 features (approximately 20 per app). The Harman et al.’s approach extracted

208 and SAFE 161 feature candidates.

SAFE achieved an overall average precision of .559 and a recall of .434. Respectively, our implementation of the Harman et al. approach achieved an average precision of .278 and a recall of .292. This results in an increase of about 100% in precision and 48% in recall between both approaches. The results are highly dependent on the analyzed app. Both approaches have the highest precision for the Google Drive app and the lowest for the Printer Pro app. Highest recalls are achieved for Fantastical 2 (SAFE) and Cloud App Mobile (Harman). The low performance for the Printer Pro app is because the verb "print", which is needed to construct app features in this case, occurred outside of the bullet list in the description.

3.2.2 Feature Extraction from the App Reviews

Similarly, we evaluated the feature extraction from reviews. Users also describe misbehavior of apps which are syntactically very similar to a feature description. For example "The app deleted all my notes". We used the SAFE approach and an implementation of the Guzman and Maalej [14] approach to extract features from reviews. Table 4 shows the results for the extractions.

Both approaches achieved a comparably low precision whereas the SAFE approach has a significant higher recall (.709) compared to the approach of Guzman and Maalej (.276), which is a difference of about 157%. The low precision of the approaches can be traced back to the fact that user reviews are very noisy and do not always refer to features as it is often the case in app descriptions.

3.2.3 Matching Features in the Descriptions and the Reviews

Finally, we report the results of our evaluation on the matching of the extracted features. The evaluation of the app reviews showed that in total 178 app features had been extracted correctly by SAFE from reviews. The data set contains these 178 true positives. The feature matching approach of SAFE uses the data set to look for matches in the true positive extracted app features from the app descriptions. SAFE could identify 27 matches of which the coders found that 19 are correct. SAFE was able to achieve a recall of .560 and a precision of .704 to correctly identify matches. On the other hand, SAFE's precision to identify non-matches correctly is .900. Additionally, we calculated the accuracy as it also contains the true negatives and the false negatives. SAFE had a promising accuracy of .870 as shown in Table 5.

4 DISCUSSION

4.1 Usage Scenarios for App Feature Extraction

Feature extraction is an elementary step for implementing many app store analytic scenarios in practice [**?**, 18]. Perhaps the most intuitive scenario is monitoring app features' health, that is to continuously identify and measure which app features are mentioned in user comments, how frequent, and which sentiments are associated with which feature [14]. Moreover, app store analytics approaches that aim at classifying the reviews as informative or not [**?**] and those that aim at classifying feedback as bug reports, feature requests, rating, and user experience [6] would profit from organizing the results based on app referred features. For instance, bug reports associated with certain features might help developers to faster trace the bugs to the relevant software components or to find other bug reports related to the same feature which include additional information for the bug reproduction.

The next analytic scenario enabled by feature extraction is the identification of the features' delta and new insights, that is identifying the differences between features described by developers and those described in the user reviews. The features' delta can even distinguish between different reviews (and thus users') subpopulations, such as features mentioned by certain users, in certain regions, stores or channels (e.g. forum or social media).

4.2 Using SAFE in Practice

SAFE has one major advantage since it is uniform: that is, it can be applied on multiple artifacts, in particular on app descriptions and app reviews. SAFE automatically identifies app features in the descriptions and maps those to the app features in the reviews. It identifies five features sets:

- App features mentioned in the description
- App features mentioned in the reviews
- App features mentioned in the description and in the reviews (intersection)
- App features mentioned only in the description

Table 3: Evaluation Results for App Descriptions

App Name	Reviews	Eval Set	Harman et al.				SAFE			
			#	P	R	F1	#	P	R	F1
Forest	9,131	13	15	0.266	0.286	0.275	13	0.462	0.400	0.429
Yahoo Mail	29,186	34	25	0.400	0.294	0.339	19	0.737	0.389	0.509
Printer Pro	6,377	11	2	0.000	0.000	-	14	0.214	0.250	0.231
Gmail	57,212	24	19	0.474	0.391	0.429	14	0.714	0.400	0.513
Google Drive	21,956	17	10	0.500	0.357	0.417	8	**0.875**	0.389	0.538
CloudApp Mobile	11,126	41	64	0.317	0.464	0.377	18	0.722	0.481	0.578
Google Docs	18,766	12	13	0.231	0.231	0.231	9	0.667	0.462	0.545
Dropbox	12,563	9	6	0.333	0.200	0.250	10	0.300	0.300	0.300
Fantastical 2	3,724	30	65	0.123	0.258	0.167	46	0.500	**0.697**	0.582
iTranslate Voice	11,834	21	12	0.333	0.211	0.258	10	0.500	0.278	0.357
Overall	-	**197**	**208**	**0.278**	**0.292**	**0.271**	**161**	**0.559**	**0.434**	**0.458**

Table 4: Evaluation Results for App Reviews

App Name	Manual #F	Guzman & Maalej				SAFE			
		#	P	R	F1	#	P	R	F1
Yahoo	46	147	0.260	0.283	0.271	74	0.238	**0.761**	0.363
Gmail	53	158	0.224	0.283	0.250	104	0.247	0.736	0.370
Dropbox	52	154	0.237	0.280	0.257	90	0.260	0.727	0.383
Fantastical	65	196	0.189	0.274	0.224	146	0.230	0.652	0.340
iTranslate	28	89	0.189	0.250	0.215	60	0.213	0.679	0.324
Overall	**244**	**744**	**0.218**	**0.276**	**0.244**	**474**	**0.239**	**0.709**	**0.357**

Table 5: Feature Matching Performance

Metric	Value
Precision (Matches)	0.704
Recall (Matches)	0.560
Precision (Non-matches)	0.900
Accuracy	0.870

(probably unpopular features)

- App features mentioned only in the reviews (probably feature requests)

Another major advantage of SAFE is that it does not require a training or a statistical model and works ”online” on the single app pages and the single reviews. This enables processing the pages and the reviews immediately during their creations. This also reduces the risk for overfitting a particular statistical model to certain apps, domains, or users.

5 CONCLUSION

In this paper, we presented a simple approach for feature extraction (SAFE) from both app pages and user reviews. We call it simple because it neither requires an a priori training with a large dataset nor a configuration of machine learning features and parameters. The SAFE approach performs a comprehensive preprocessing of the natural language input, applies Part-of-Speech and sentence patterns to extract human readable features and finally matches the features from app reviews to the features extracted in the related app page. On app descriptions, we obtained a precision of up to 88% with an average of 56% and a recall of 70% with an average of 43%. For unfiltered user reviews, we had an average precision of 24% but could achieve a recall of 71%. Features extracted from user reviews are thus still noisy but the relatively high recall confirms that we catch the prevailing majority of features discussed by the users. The feature matching approach

performed three steps, consisting of a single term matching, a synonym matching, and a semantic similarity matching achieving an accuracy of 87%.

Our approach is an enabler for multiple app store analytics scenarios like monitoring app features' health, the identification of features' delta for gaining new insights, and feature recommendation and optimization. Therefore, SAFE gives analysts a feature-based perspective on their apps.

Even if the results are encouraging, future research and a fine tuning of SAFE is needed. A subsequent machine learning approach trained by developers can further improve the results.

ACKNOWLEDGMENT

This research is part of the OPENREQ project, funded by the Horizon 2020 program of the European Union. Project Id: 732463.

References

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

[2] A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang, "App store analysis: Mining app stores for relationships between customer, business and technical characteristics," Research Note RN/14/10, UCL Department of Computer Science, 2014.

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

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

[5] L. V. Galvis Carreño and K. Winbladh, "Analysis of user comments: an approach for software requirements evolution," in *ICSE '13 Proceedings of the 2013 International Conference on Software Engineering*, 2013, pp. 582-591.

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

[7] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 1025-1035.

[8] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: Msr for app stores," in *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, 2012, pp. 108-111.

[9] F. Sarro, A. A. Al-Subaihin, M. Harman, Y. Jia, W. Martin, and Y. Zhang, "Feature lifecycles as they spread, migrate, remain, and die in app stores," in *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, 2015, pp. 76-85.

[10] N. Hariri, C. Castro-Herrera, M. Mirakhorli, J. Cleland-Huang, and B. Mobasher, "Supporting domain analysis through mining and recommending features from online product listings," *IEEE Transactions on Software Engineering*, vol. 39, no. 12, pp. 1736-1752, 2013.

[11] M. Nayebi and G. Ruhe, "Trade-off service portfolio planning- a case study on mining the android app market," *PeerJ*, e1671, 2015.

[12] K. Wiegers and J. Beatty, *Software Requirements (Developer Best Practices)*, 3rd ed. Microsoft Press, 2014.

[13] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and P. A. Spencer, "Feature-oriented domain analysis (FODA) feasibility study," Technical Report CMU/SEI-90-TR-21, CMU, 1990.

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

[15] E. Murphy-Hill, T. Zimmermann, and N. Nagappan, "Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development?" in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 1-11.

[16] Y. Li, D. McLean, Z. A. Bandar, J. D. O'Shea,

and K. Crockett, ”Sentence similarity based on semantic nets and corpus statistics,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 8, pp. 1138-1150, 2006.

[17] A. A. Al-Subaihin, F. Sarro, S. Black, L. Capra, M. Harman, Y. Jia, and Y. Zhang, ”Clustering mobile apps based on mined textual features,” in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2016, pp. 38:1-38:10.

[18] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe, ”Toward data-driven requirements engineering,” *IEEE Software: Special Issue on the Future of Software Engineering*, vol. 33, no. 1, pp. 48-54, 2016.