SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews

Timo Johann, Christoph Stanik, Alireza M. Alizadeh B. and Walid Maalej
University of Hamburg
Hamburg, Germany
{johann, stanik, 4molla, maalej}@informatik.uni-hamburg.de

***Abstract*—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. In this paper, we propose SAFE, a novel uniform approach to extract app features from the single app pages, the single reviews and to match them. 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.**

***Index Terms*—User Reviews, App Store Analytics, Software Feature, Data Mining, Data-Driven Requirements**

I. 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 [26]. Second, aggregate information from customer, business, and technical perspectives [7]. Each app in the store is presented by its own app page[1]. 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 [26], 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 [22]. Most of these works focus on mining large amount of app store data to derive advice for analysts, developers, and users. One popular analytic scenario is to mine app reviews for identifying popular user complaints or requests [8], [13], [18], and assisting release planning [5], [10], [28]. Another scenario is to mine the app pages [9], [12] and their evolution over time [27] to derive recommendations for users or to identify combinations that increase download numbers [11], [24].

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. The SAFE patterns and the SAFE feature extraction (Section II) represent our **first contribution**.

We implemented our approach and applied it to the descriptions and the reviews of 10 popular apps from the Apple App Store. We also reimplemented two frequently cited state-of-the-art approaches: one focuses on app pages [12], the other on reviews [10]. We evaluated and compared the feature extraction accuracy as well as the matching between the features in the reviews and the descriptions. The evaluation methodology,

[1]https://developer.apple.com/app-store/product-page/

2332-6441/17 $31.00 © 2017 IEEE
DOI 10.1109/RE.2017.71

tools, and dataset (Section III) together with the benchmark results (Section IV) represent our **second contribution**. Our discussion how a uniform app feature extraction can be applied in practice, and what analytic tool vendors and researchers can learn from our research (Section V) represents the **third contribution**. Finally, Section VI summarizes related work and Section VII concludes the paper.

II. 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" [29]. Kang et al. define a feature as "a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems" [14]. 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" [12].

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 following 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 [10], where requirements and features are described and handled differently. Murphy-Hill et al. [23] 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.

A. 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.

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. Figure 1 shows an example of this task: the upper part shows the sentences in the tool including the POS tags; in the lower part, the researcher can enter the POS pattern (that is the list of POS tags) that describe a feature and insert that pattern to the pattern catalogue. In Figure 1, the POS pattern is *VB NN NN* (verb, noun, noun)[2] . It describes the feature *use incognito mode*. Table I 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".

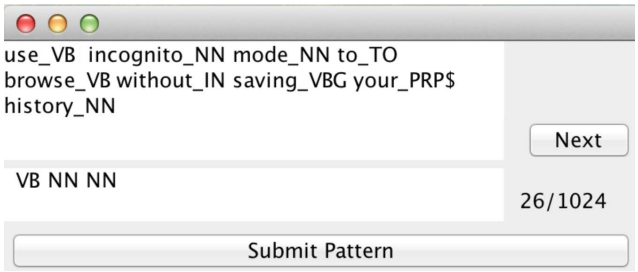

Fig. 1. Tool to identify POS tag patterns used to denote app features.

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

[2]POS tags of the Penn Treebank Project: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html.

TABLE I
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

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 (1). Then SAFE searches for occurrences and the placement of conjunctions within the sentence (2). 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 (3). Finally, the feature identifiers are combined with the remaining list of app features (4).

Case 1 is rather simple. It is identified by the single conjunction “and” as well as the feature identifiers to the left and the right side of the conjunction. SAFE combines the feature identifiers to the app features “send attachments” and “receive attachments”. In **Case 2** SAFE identifies a conjunction on the right side of an enumeration, so that the approach can extract “view documents”, “view PDFs”, “view photos”, and “view videos” as app features. **Case 3** contains two conjunctions and four feature identifiers. Based on the sentence structure, SAFE creates the cross product of the feature identifiers leading to the app features “discuss notes”, “discuss drafts”, “annotate notes”, and “annotate drafts”. For **Case 4** SAFE first identifies two conjunctions. Then, it finds the two feature identifiers “scan” and “comment” besides the first conjunction in the sentence. At the second conjunction, there is an enumeration on the left side and a feature identifier on the right side, which will be combined to a list of the five feature identifiers “paper”, “printed documents”, “business cards”, “handwriting” and “sketches”. In a final step, SAFE combines the first set of feature identifiers “scan” and “comment” with each element of the list resulting in the features “scan paper”, “comment paper”, and so forth. **Case 5** also contains two conjunctions. SAFE looks at the first conjunction and identifies an enumeration on its left and a feature identifier on its right. The second conjunction also has an enumeration on its left and a feature identifier on the right side. The final step combines each feature identifier of both lists to create app features such as “write notebooks” and “collect to-do lists”.

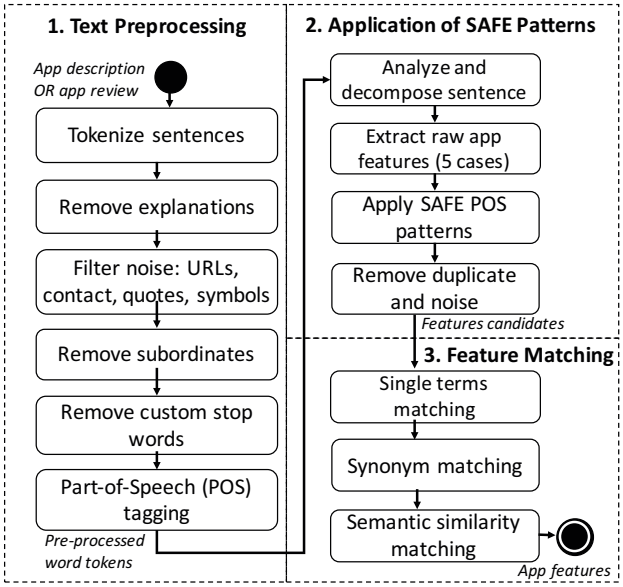

Fig. 2. The main steps of the SAFE approach.

B. Automated Feature Extraction

Figure 2 illustrates the overall steps of SAFE.

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

behaving) as the following sentence shows: “The app let you upload images, so that you don’t have to worry about losing them”. Next, SAFE word-tokenizes each sentence without changing the order of the words. Thereafter, it removes stop words and the name of the app from each tokenized sentence to reduce the number of words that might introduce noise in the feature extraction [10]. The final step of the preprocessing is to attach Part-of-Speech tags to each word token.

2) Application of SAFE Patterns: This part starts with analyzing and decomposing the sentences based on the *conjunctions*, *enumerations*, and *feature identifiers* as described in Section II-A. 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 I 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*).

C. 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 [10] and Li et al. [16].

Overall, SAFE performs three steps (see Figure 2) , each representing 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 ant 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 II.

TABLE II
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

III. EMPIRICAL EVALUATION

We introduce our evaluation goals, evaluation questions, and data used. We then describe the methodology followed.

A. 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[3]. 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. For further analysis, the data was persisted inside a MongoDB database. To enable replication, we publish the dataset on our website[4].

We also reimplemented two recent, frequently cited approaches for extracting features from app store data: the approach of Harman at al. [12] (also used by Al-Subaihin et al. [2]) focuses on app descriptions and the approach by Guzman and Maalej [10] focuses on for app reviews.

[3]https://affiliate.itunes.apple.com/resources/documentation/itunes-store-web-service-search-api

[4]https://mast.informatik.uni-hamburg.de/app-review-analysis

The feature extraction approach by *Harman et al.* includes four main steps [12]. First, it extracts coarse features by analyzing app descriptions of multiple apps to find the part of the description that contains app features. Then, it removes stop words to filter out noise. Afterwards, word frequencies and trigram collocations are computed, which leads to featurelets. Finally, the authors use a greedy clustering algorithm to group featurelets that share at least two words.

The approach of *Guzman and Maalej* focuses on feature extraction on a large set of reviews. The authors gather the title and the comments of each review of an app. Then, they perform several preprocessing steps to extract nouns, verbs, and adjectives, to remove stop words, and to perform lemmatization and group inflected words. Afterwards, the authors perform bigram collocation finding by means of a likelihood ratio test [21]. Thereafter, they filter the collocations by keeping those that appear in at least three reviews while ignoring the order of the words within the collocations. As users might use different terms for the same app feature, the authors group the remaining collocations which have the same synonyms. Finally, they choose the collocation with the highest frequency to be the representation of its group.

B. Evaluation Method

To evaluate the accuracy of SAFE and compare it with the two other approaches, we first created three evaluation sets: for the extraction from app descriptions, for the extraction from app reviews, and for the feature matching. Then we instructed human coders to verify the results of the three approaches. The creation of the evaluation sets are described in each of the following subsections.

The evaluation was performed in two parts. First, we evaluated the app feature extraction on app descriptions and then on app reviews. The benchmark approaches had been replicated to the best of our knowledge. As the approach of Harman et al. focuses on app descriptions, we used it as a benchmark to evaluate the feature extraction from app descriptions. Likewise, as the Guzman and Maalej approach was developed for app reviews we applied their approach as a benchmark for extracting features from reviews. In the following we explain both procedures.

1) App Descriptions: For the creation of the evaluation set, two authors independently and manually extracted app features from the description of each app. We compared the extracted app features of both authors and agreed what features should be in the evaluation set. In case of disagreements a third person was involved. However, this was only necessary in 2 cases.

We then extracted the app features using Harman's et al. approach and SAFE and checked the results with the manual extraction. For this, we implemented a coding tool shown on Figure 3. The tool shows the app description on the left side and asks two independent coders to carefully check the manual set and state, on a binary scale, if the extracted app features are part of the app description. The coding tool displays one extracted app feature at the time. We randomized the extracted app features of both approaches to reduce possible bias that might be introduced if the results of each approach would be shown one after another. Finally, we use this evaluation set to calculate precision, recall, and the f-measure by comparing the coding results with the evaluation set.

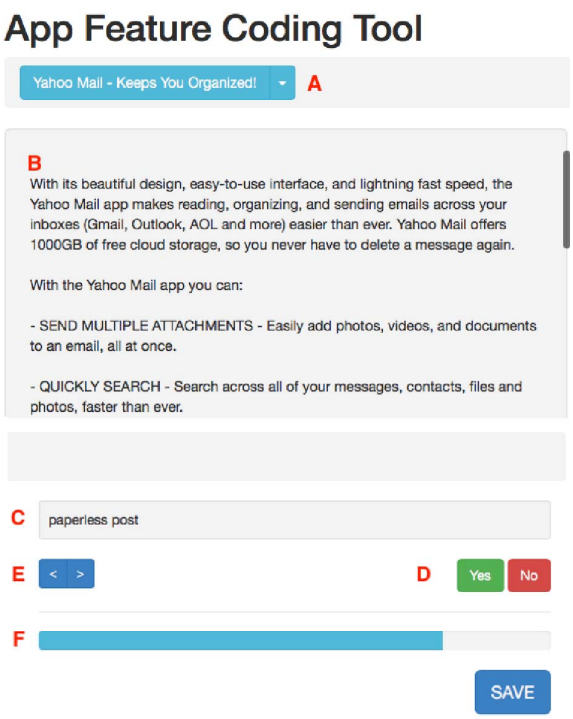

Fig. 3. Coding tool for evaluating the feature extraction from app descriptions. A: The app to be coded. B: The raw app description. C: A feature extracted by an automated approach (i.e. either SAFE or Harman et al.). D: Code if C is a feature and part of the description. E: Navigation. F: Progress bar.

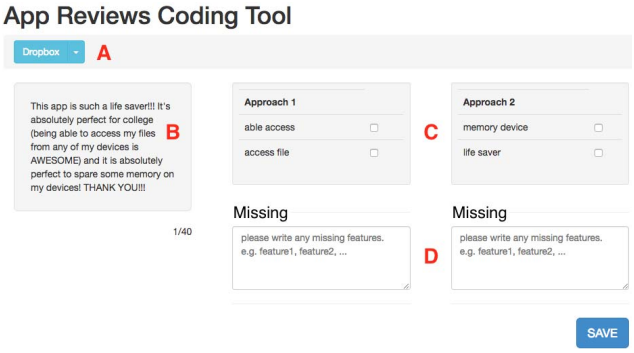

Fig. 4. Coding tool for evaluating the feature extraction from reviews. The app to be coded. B: The raw app review. C: The features extracted by both approaches (SAFE and Guzman and Maalej with random order). A checked box marks a correctly identified feature. D: Text field to add features that the approach might have missed.

For the manual feature extraction we agreed on the following rules:

1) Focus on functional aspects and omit quality aspects: "Quickly search emails" becomes "search emails".
2) Focus on the essential feature: "Upload your family

photos" becomes "upload photos".

3) Remove the benefit gained from a feature: "Undo Send, to prevent embarrassing mistakes" becomes "undo send".
4) "Open, edit, and save documents" is split to "open documents", "edit documents", and "save documents".

2) App Reviews: For the evaluation of the feature extraction from the reviews, we selected the 80 most recent app reviews of five apps from our data set. We compared the feature extraction by Guzman and Maalej and SAFE. As the approach of Guzman and Maalej expects a big set of app reviews, we feeded it with a random sample of 5000 reviews for each app (except for one app, which only had a total of 3,742 app reviews). In contrast, SAFE expects single app reviews as input.

We also adjusted the coding tool to match the review evaluation, as shown in Figure 4. Other than for the app description, the creation of the ground truth was during the coding by manually adding missing features. On the left side of the tool, a single app review is displayed. On the right side, all extracted app features from both approaches are listed randomly. Two independent coders have a binary choice to check if a feature was correctly extracted. To reduce a possible coder bias, the order of the approaches is randomized for each app review. In addition, the coder added the app features each approach missed by writing them down in the appropriate text box below the correlating approach.

C. Feature Matching

We evaluated the feature matching by using the true positive features extracted from the app descriptions and the true positives from the app reviews. We took the extracted app features from the five apps we used for coding the reviews. We then run the feature matching of SAFE and created a table containing the results. Table III shows a simplified example of the evaluation. It includes the app features extracted from the reviews, the app features extracted from the description, as well as the matching result of SAFE and a column in which the coder evaluates the result. Two human coders evaluated the matching results. During the evaluation, the coders also had access to the list of all true positive extracted app features from the descriptions. This enabled the coders to verify if app features for which SAFE could not find a match, really did not contain any.

There are three main use cases for the matching: First, we can identify app features users refer to. Second, we can use this information to identify app features that users refers to but are not described in the app page. Finally, we might be able to extract app feature requests from mismatches. As soon as all the extracted app features of a review have been evaluated, the coder will be asked if the approaches missed any feature. This additional step allows us to calculate the recall, because we have not created an evaluation set for the app reviews.

IV. Evaluation Results

We separately evaluated the feature extraction of the SAFE approach for descriptions and reviews, as well as the feature matching. We compared the results of the feature extraction with two recent approaches. We chose the approach of Harman et al. [12], which was designed for feature extraction from descriptions. For feature extraction from reviews we chose the approach of Guzman and Maalej [10]. We calculated precision and recall for feature extraction from descriptions and reviews as well as for the feature matching as in 1 and 2.

TABLE III
Simplified example of the feature matching evaluation.

Feature (review)	Feature (description)	Matching Result	Coder's Statement
send email	send emails	match found	True
month view	week view	match found	False
retrieve images	retrieve pictures	no match	False
email notification		no match	True

$$Precision = \frac{tp}{tp + fp} \tag{1}$$

$$Recall = \frac{tp}{tp + fn} \tag{2}$$

For the feature extraction steps the precision is the fraction of retrieved features that are part of our manual evaluations set. Recall is the fraction of features that are part of our evaluation set and are retrieved by the respective approach. The true positives (tp) are the number of features that are correctly extracted by the approaches (those that are part of the evaluation set). The false positives (fp) are the number of features that are extracted by the approaches but are not considered as a feature (not part of the evaluation set). The false negatives (fn) are the features that are not extracted by the respective approach, but are part of the evaluation set.

A. 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 IV shows the results. The evaluation is based on ten apps (see Section III-B) and contains 197 features ($\sim$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 (standard derivation of precision: .21 and recall: .12 for SAFE; .15/.13 for Harman et al.). 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.

TABLE IV
EVALUATION RESULTS FOR APP DESCRIPTIONS. # MEANS COUNT, P MEANS PRECISION, R MEANS RECALL. MAX VALUES IN A COLUMN ARE IN BOLD.

App Name	# Reviews	Evaluation Set	Approaches							
			Harman et al. [12]				SAFE			
		Feature #	#	P	R	F_1	#	P	R	F_1
Forest: Stay focused, be present	913	13	15	.266	.286	.275	13	.462	.400	.429
Yahoo Mail - Keeps You Organized!	29,186	34	25	.400	.294	.339	19	.737	.389	.509
Printer Pro - Print documents, photos, emails	6,377	11	2	.000	.000	-	14	.214	.250	.231
Gmail - email by Google: secure, fast & organized	57,212	24	19	.474	.391	**.429**	14	.714	.400	.513
Google Drive - free online storage	21956	17	10	**.500**	.357	.417	8	**.875**	.389	.538
CloudApp Mobile	111	26	41	.317	**.464**	.377	18	.722	.481	.578
Google Docs	18,766	12	13	.231	.231	.231	9	.667	.462	.545
Dropbox	12,563	9	6	.333	.200	.250	10	.300	.300	.300
Fantastical 2 for iPhone - Calendar and Reminders	3,724	30	65	.123	.258	.167	46	.500	**.697**	**.582**
iTranslate Voice - Speak & Translate in Real Time	11,834	21	12	.333	.211	.258	10	.500	.278	.357
Overall		197	208	.278	.292	.27	161	.559	.434	.458

B. 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 [10] approach to extract features from reviews. Table V 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.

In contrast to descriptions, reviews often contain slang, typos or incorrect grammar and thus represent a more challenging input for feature extraction. The average recall of .709 means that SAFE does not miss too many features. In practice it might be more relevant to extract most of the features than the reduction of the noise.

TABLE V
EVALUATION RESULTS FOR APP REVIEWS. F MEANS NUMBER OF ACTUAL FEATURES, # MEANS TOTAL FEATURES EXTRACTED, P MEANS PRECISION, R MEANS RECALL.

App Name	Manual #F	Approaches							
		Guzman&Maalej [10]				SAFE			
		#	P	R	F_1	#	P	R	F_1
Yahoo	46	147	**.260**	**.283**	**.271**	74	.238	**.761**	.363
Gmail	53	158	.224	**.283**	.250	104	.247	.736	.370
Dropbox	52	154	.237	.280	.257	90	**.260**	.727	**.383**
Fantastical	65	196	.189	.274	.224	146	.230	.652	.340
iTranslate	28	89	.189	.250	.215	60	.213	.679	.325
Overall means	244	744	.218	.276	.244	474	.239	.709	.358

C. 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 as shown in Equation 3. SAFE had a promising accuracy of .870.

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{3}$$

V. DISCUSSION

We discuss how researchers and tool vendors can use SAFE and similar approaches to help software engineering teams analyze and exploit app store data. We then discuss how SAFE can be combined with other approaches to overcome its limitations. Finally, we discuss the threats to validity of our research and the measures we used to mitigate these threats.

A. Usage Scenarios for App Feature Extraction

Feature extraction is an elementary step for implementing many app store analytic scenarios in practice [17], [19]. 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 [10]. Moreover, app store analytics approaches that aim at classifying the reviews as informative or not [5] and those that aim at classifying feedback as bug reports, feature requests, rating, and user experience [18] 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).

Such a difference can reveal two insights: either users apply a different vocabulary or the features discussed in the reviews are missing in the app. In the first case, developers and users might learn how to communicate better and bridge their "vocabulary gap". In the second case, developers can easily get the list of suggested features or at least those that are associated with their apps but not included in them: a useful information for requirements identification, scoping, and prioritization. As apposed to analytics approaches that allow identifying reviews with feature requests (e.g. by Iakob and Harrison [13] or by Maalej et al. [18]), feature extraction allows also to **filter the relevant review parts** (e.g. sentences), which include the relevant information – something particularly useful as the quality and structure of the reviews are very heterogeneous. Relevant insightful information about the feature request might be in the first or in the last sentence of a review. This will also enable the clustering and aggregation of artifacts (such as comments) based on features as well as identifying feature duplicates in these artifacts.

Finally, feature extraction is the preliminary underlying step in many if not all **feature recommendation and optimization** scenarios [11], [24], [27]. Feature recommendations might target both users and developers. For users, feature extraction can help building recommender models that learn dependencies of reported and used features. From the app usage, such models can also derive which features users are interested in. When combining features mentioned in the pages of many apps (e.g. apps within the same category), a predictor model could derive the optimal combination of features, and answer questions such as "which dependencies of features get the best ratings". This might also help identifying which features are missing in a particular app release from the "feature universe".

B. 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 fives feature 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 (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.

Nevertheless, we think that the achieved accuracy of SAFE – even if it outperforms other research approaches – is not good enough to be applied in practice. We think that a hybrid approach (a simple, pattern and similarity based as SAFE together with a machine learning approach) is probably the most appropriate. For instance, machine learning can be used to pre-filter and classify reviews before applying SAFE on them. Indeed, we think that this is the main reason why our accuracy values were rather moderate: because we were very restrictive in the evaluation and applied SAFE on "wild, random" reviews.

SAFE can also be extended by a model that is trained from multiple pages and multiples users. This might e.g. be used to support users to improve their vocabulary by auto-completing feature terms.

Also, developers and analysts should be able to correct the extracted features e.g. following a critique-based recommender model, or simply building a supervised classifier based on SAFE and continuously improving the classifier model based on developers' feedback (identifying false positives and true negatives). This can either be used to persist the list of actual app features or to train a classifier to learn or adjust patterns per app.

C. Limitations and Threats to Validity

As for every study that includes manual coding, the coders in our evaluation might have done mistakes a) indicating wrong features in the descriptions and reviews or b) wrongly assessing the extraction of the SAFE or the two reference approaches. To mitigate this risk, we conducted a careful peer-coding based on a coding guide (e.g. stating what is a feature) and using a uniform coding tool. Therefore, we think that the reliability and validity of the app description and feature matching evaluations – both involving peer-coding – is rather high. For the reviews, we decided to have one single coder for two reasons. First, the large number of reviews would require more resources. Second, the coders were well trained from the first phase of coding app descriptions. Still in future, manually pre extracting review features, as we did for the features from app descriptions, would certainly improve the reliability. As we share the data, we encourage follow researchers to replicate and extend the evaluation.

Concerning the sampling, our evaluation relies only on Apple App Store data and apps from one single category. While this helped concentrating the effort and to learn the evaluation apps, it certainly limits the external validity of our results. It is, however, important to mention that the SAFE patterns, the core "idea" of the approach, were identified from Google Play Store data of 100 apps from multiple categories. Since SAFE also works on Apple Store data, we feel comfortable to claim that our approach is generalizable to other apps, except for games which we ignored on purpose (see Section II).

As for the accuracy and benchmark values, we refrain from claiming that these are exact values and we think that they

are rather indicative. We think that the order of magnitude of precisions and recalls calculated, as well as the differences between the approaches is significant. We took special careful measures to conduct a fair and non-biased comparison. For instance, during the evaluation of the extraction, the tool randomly showed the extracted results in the same layout, without mentioning which approach extracted which features.

Finally, when reimplementing the reference approaches we might have done mistakes. We tried to conduct the comparison as fair as possible and implemented all steps described in the papers to the best of our knowledge. In some cases (e.g. list of stop words), we tried to rather be spacious and fine-tuned the reference approach to have results comparable to those reported on the papers.

VI. Related Work

A. App Store Analysis

App stores contain important information for app developers, analysts, and other software stakeholders [20] [25]. App store analysis is thus a growing research field. Martin et al. [22] conducted a systematic literature review on app store analysis. The earliest reported study dated to the year 2010. Martin et al. [22] defined 7 subfields of app store analysis: API Analysis, Feature Analysis, Release Engineering, Review Analysis, Security, Store Ecosystem, and Size & Effort Prediction. 127 technical and non-technical studies were considered relevant. Our study can be classified into the fields of Feature Analysis as well as Review Analysis and is – to the best of our knowledge – the first that combines these two fields.

Indeed, most promising app store analytics approaches proposed so far are rather app independent by nature. For instance, classifying reviews in bug reports, feature request, user experience, or ratings [18] is semantically independent from the single apps and is rather related to general software engineering activities and concepts (bugs, enhancement, etc.). Approaches that tried to analyze the app problem domain rather focus on identifying topics [8] (which can be everything) or filtering informative reviews [5], [28].

The most related approaches to our are those of Harman et al. [12] and Guzman and Maalej [10]. The latter focused on feature extraction from user reviews and applied a sentiment analysis to find out how users like a certain feature. The main difference to our approach is that we do not need a large data input to train a model. The approaches are also intended for either user reviews or app descriptions but not for both. In contrast, we also enable the matching of features mentioned in the reviews and the descriptions.

B. Feature Extraction, Mining, and Recommendation

Baker et al. carried out a systematic literature review on feature extraction approaches from natural language requirements [4]. They identified 13 articles that use feature extraction approaches. However, these are not related to feature extraction from app pages and user reviews in app stores. Our approach is highly tightened to these artifacts with the intention to support app store analytic scenarios as discussion in Section V.

Feature mining and recommendation are often discussed in the context of domain analysis [3]. The majority of related work focused on requirements specification or other project internal documentation. These kinds of documents are already semi formatted and are not comparable to descriptions and reviews in app stores.

Studies that explicitly focus on product descriptions and reviews are yet limited, pursue different goals, or are focused on specific tasks. Hariri et al. [11] rely on previous work of Dumuitru et al. [6] and focus on software product descriptions, but not on the app stores. Harman et al. [12] presented a feature extraction approach from app store descriptions. Their approach aimed more at the clustering and is relevant, e.g., for comparison purposes. They consider elements such as near, wifi, hotspot as a feature, whereas our approach is looking for fine-grained features such as *upload pdf file* or *edit photos*.

Acher et al. [1] build a feature model, based on features described in a table structure. In app stores, descriptions are not predefined in formal or semi formal structures, the approach cannot be applied directly. Similarly, Wang et al. [30] present a feature recommendation approach from online software repositories, like app stores and base their approach on feature topic models which cannot be applied on single product descriptions.

Lulu and Kuflik [15] presented an approach to cluster apps based on their functionality and enable users to find a desired app faster. The authors propose an unsupervised machine learning approach that extracts function keywords found in app descriptions, blogs, and articles and processes those to build the clusters. As the goal of the authors was to create clusters, they do not extract a set of human readable app features but rather a set of function keywords from multiple sources.

VII. 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.

VIII. Acknowledgement

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

References

[1] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire. On extracting feature models from product descriptions. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, pages 45–54. ACM, 2012.

[2] 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*, ESEM '16, pages 38:1–38:10. ACM, 2016.

[3] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler. An exploratory study of information retrieval techniques in domain analysis. In *12th International Software Product Line Conference*, pages 67–76, 2008.

[4] N. H. Bakar, Z. M. Kasirun, and N. Salleh. Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review. *Journal of Systems and Software*, 106:132–149, 2015.

[5] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang. AR-miner: Mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 767–778. ACM, 2014.

[6] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 181–190. ACM, 2011.

[7] 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. Reseach Note RN/14/10, UCL Department of Computer Science, 2014.

[8] 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*, pages 582–591. IEEE Press, 2013.

[9] 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*, ICSE 2014, pages 1025–1035. ACM, 2014.

[10] 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*, pages 153–162, 2014.

[11] 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*, 39(12):1736–1752, 2013.

[12] 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*, pages 108–111, 2012.

[13] 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*, pages 41–44. IEEE, 2013.

[14] 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.

[15] D. Lavid Ben Lulu and T. Kuflik. Functionality-based clustering using short textual description: Helping users to find apps installed on their mobile device. In *Proceedings of the 2013 International Conference on Intelligent User Interfaces*, IUI '13, pages 297–306. ACM, 2013.

[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*, 18(8):1138–1150, 2006.

[17] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik. On the automatic classification of app reviews. *Requirements Engineering*, 21(3):311–331, 2016.

[18] 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*, pages 116–125, 2015.

[19] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe. Toward data-driven requirements engineering. *IEEE Software: Special Issue on the Future of Software Engineering*, 33(1):48–54, 2016.

[20] W. Maalej and D. Pagano. On the socialness of software. In *Proceedings of 9th DASC*. IEEE Computer Society, 2011.

[21] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[22] 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.

[23] 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*, pages 1–11. ACM, 2014.

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

[25] D. Pagano and B. Brügge. User involvement in software evolution practice: A case study. In *Proceedings of the 35th ICSE*, 2013.

[26] D. Pagano and W. Maalej. User feedback in the appstore: An empirical study. In *21st IEEE International Requirements Engineering Conference*, pages 125–134. IEEE, 2013.

[27] 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. *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, pages 76–85, 2015.

[28] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. Release planning of mobile apps based on user reviews. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 14–24. ACM, 2016.

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

[30] Y. Yu, H. Wang, G. Yin, and B. Liu. Mining and recommending software features across multiple web repositories. In *Proceedings of the 5th Asia-Pacific Symposium on Internetware*, pages 1–9. ACM, 2013.