Users – The Hidden Software Product Quality Experts?

A Study on How App Users Report Quality Aspects in Online Reviews

Eduard C. Groen*, Sylwia Kopczyńska†, Marc P. Hauer‡, Tobias D. Krafft‡, and Joerg Doerr*
*Fraunhofer IESE, Kaiserslautern, Germany
Email: {eduard.groen,joerg.doerr}@iese.fraunhofer.de; ORCIDs: 0000-0002-5551-8152; 0000-0003-0048-399X
†Poznan University of Technology, Poznań, Poland
Email: sylwia.kopczynska@cs.put.poznan.pl; ORCID: 0000-0002-9550-3334
‡Technische Universität Kaiserslautern, Kaiserslautern, Germany
Email: {hauer,krafft}@cs.uni-kl.de; ORCIDs: 0000-0002-1598-1812; 0000-0002-3527-1092

***Abstract*—[Context and motivation] Research on eliciting requirements from a large number of online reviews using automated means has focused on functional aspects. Assuring the quality of an app is vital for its success. This is why user feedback concerning quality issues should be considered as well** [Question/problem] **But to what extent do online reviews of apps address quality characteristics? And how much potential is there to extract such knowledge through automation?** [Principal ideas/results] **By tagging online reviews, we found that users mainly write about "usability" and "reliability", but the majority of statements are on a subcharacteristic level, most notably regarding "operability", "adaptability", "fault tolerance", and "interoperability". A set of 16 language patterns regarding "usability" correctly identified 1,528 statements from a large dataset far more efficiently than our manual analysis of a small subset.** [Contribution] **We found that statements can especially be derived from online reviews about qualities by which users are directly affected, although with some ambiguity. Language patterns can identify statements about qualities with high precision, though the recall is modest at this time. Nevertheless, our results have shown that online reviews are an unused Big Data source for quality requirements.**

Index: Terms—crowd based requirements engineering, requirements engineering, non-functional requirements, online user reviews, quality characteristics, quality requirements.

I. Introduction

The requirements engineering (RE) community increasingly sees user feedback on online platforms as a potential source of user requirements. These platforms include app stores like those of Samsung, BlackBerry [1], Google, and Apple [2] [3] [4], social media such as Twitter [5], or issue tracking systems [6]. Manual tagging studies have found online reviews to provide useful information on product features when classified using content categories such as feature requests [3] [5] and general criticism/praise [2] [7]. However, manual tagging is a tedious and time-consuming task. Thus, approaches to automating such analyses are being researched under the name "Crowd-Based RE" [8]. One such method, which we will use in this paper, employs language patterns involving regular expressions to match statements with the specified pattern through automation.

Typically, requirements are distinguished into three categories: functional requirements (FRs), constraints, and quality requirements or non-functional requirements [9]. Concerning the first category, existing FRs can be refined by analyzing online reviews for criticism and praise, while new FRs can be derived from feature requests [3]. Requirements of the second category, constraints, affect and limit projects and software solutions, and are often beyond the user's awareness or explanation capabilities. For example, a user will usually not know about the availability of resources such as budget and manpower, and will not always understand how laws and standards affect software. Hence, there is little chance that user feedback provides useful information on constraints if these are addressed at all. Yet user feedback on the third category, quality requirements, may be of interest. Since users are directly affected by quality characteristics such as usability, performance efficiency, and security, it is probable that online reviews contain statements about product qualities. However, we are aware of only one study that has classified statements according to quality aspects, and this research by Lu and Liang was published nearly simultaneously to this paper [10]. Although their work suggests that quality aspects are indeed addressed by users, it does not systematically address the question to what extent each quality characteristic is addressed.

Research on quality requirements is highly relevant, as they are a crucial aspect of software product development. These requirements are the architectural drivers [11], and not adequately addressing them will likely lead to project failure and high rework cost [12] (see Section V). They also increasingly form a unique selling proposition, especially for consumer products. But getting a complete and correct set of quality requirements is difficult [13] [14]. Some studies have considered bug reports (e.g., [5] [6] [7]), but they identified feature shortcomings at the functional level, and at best they discovered keywords such as "crash" to obtain an indication of poor reliability.

Thus, in this research, we set out to determine whether user feedback can also be a useful source of statements to support the elicitation or prioritization of quality requirements. We formulated the following research questions:

> **RQ1**: What quality aspects of software products are raised by users in app reviews?
>
> **RQ2**: How should language patterns be defined so that quality aspects of software products formulated by users can be automatically detected or extracted?

To answer these questions, we first conducted a study where we manually tagged online reviews for statements about product qualities, which is presented in Section II. In a second study, we performed an automated analysis of online reviews, which is presented in Section III. We discuss potential threats to validity in Section IV. Section V presents related work, and Section VI our conclusions.

II. Study I: Investigation of Online Reviews

A. Method

In order to understand what users say about the quality of apps, we set out to identify statements on product quality by tagging them through content analysis [15].

Step 1: Select and obtain online reviews. To create a dataset with online user reviews, we looked for apps in the five app categories that Pagano and Maalej identified as garnering the most online reviews, which are usually apps with which users interact and build a relationship with [2]. We added the category "Smart Products", for which the owner of a "smart" physical product can download a free app to unlock further functionalities. In each product category, we selected one paid app and a free app, as online reviews of free and paid apps have also been found to differ [2]. We searched for online reviews to the most popular apps available on the international (English-language) version of the Apple App Store, the Google Play Store, and Amazon.com. First, we selected only those apps that are available in all three app stores. As the secondary criterion, each app should have at least a 5% share of reviews for each star on the 5-star rating scale. We then randomly selected one free and one paid application per category. This resulted in our test set from 36 sources ($6\ categories \times 2\ apps \times 3\ app\ stores$), as shown in Table I. The average rating for all apps was 3.5 stars. The online reviews of these products were then crawled using a customized text mining tool.

Step 2: Structure online reviews. We took a stratified random sample from the dataset by randomly selecting two reviews for each rating from the 5-star rating scale, resulting in ten reviews per product per store, or 360 reviews in total. For Cleverbot on Amazon, our dataset included only one 4-star review, which was compensated by adding a randomly selected review from the most frequent adjacent rating, which was a 5-star review. Using OpenNLP (http://opennlp.apache.org), we then split these reviews into separate sentences, which we call statements, for a total of 1,385 statements. The statements were exported to a CSV file, which we loaded into a customized web-based tagging tool to enable the annotators to tag statements independently of one another.

TABLE I
List of apps in our dataset with the number of reviews crawled from three app stores. An asterisk (*) indicates that the app store limited the number of retrievable reviews (Am = Amazon, Ap = Apple App Store, Go = Google Play).

Category	App (Price)	No. crawled reviews		
		Am	Ap	Go
Entertainment	Disney Movies Anywhere	120	2,359	4,217
	Cleverbot ($0.99)	59	1,530	800
Productivity	OneNote	422	6,722	2,959
	Tiny Scan Pro ($4.99)	432	2,140	601
Social Media	TweetCaster	1,200	7,758	4,050*
	TweetCaster Pro ($4.00)	857	724	4,023*
Messaging	Viber	889	67,671	4,306*
	IM+ Pro ($2.99)	132	193	2,215
Games	PAC-MAN 256 – Endless Arcade	698	2,261	4,402*
	Maze Sonic & SEGA All Stars Racing ($2.99)	302	3,548	420
Smart Products	Philips Hue	69	2,469	601
	August Smart Lock	344	290	283
	Total	5,602	97,715	28,877

Step 3: Define tagging categories and a tagging schema. Before the annotators could categorize statements by assigning "tags" in the tagging tool, we had to determine these tags. As we are interested in statements about product qualities, we decided to use the structure of the quality model for "software product quality" proposed in the ISO 25010 standard [16], a commonly used quality standard. This quality model contains eight software product quality characteristics, which became our "main tag". These are in turn subdivided into 31 subcharacteristics, which became our "sub tags". For example, the characteristic "compatibility" has the two subcharacteristics "co-existence" and "interoperability". To the "main tag", we added the option "none" to tag statements that are not about quality or which are too unclear. To get a common understanding of the meaning of the tags, we performed two test runs and discussed the process and the results, based on which we iteratively composed a tagging schema with the definitions from ISO 25010, along with examples and signal words. The annotators were additionally trained on the ISO 25010 characteristics and subcharacteristics.

Step 4: Perform tagging. The tagging was performed individually by five annotators, consisting of two researchers in computer science and psychology, and three social informatics students. After reading each statement, we assigned the best fitting main tag. The sub tag was optional, and was only assigned if the statement could clearly be assigned to a particular subcharacteristic. For example, the statement *"Very ugly emoticons!!!"* was assigned "usability" as a main tag, and "user interface aesthetics" as a sub tag. The statement *"Doesn't even load for me anymore."* was assigned only the main tag "reliability", because it is unclear what the cause of the problem is. Additionally, if a statement could clearly be assigned to more than one characteristic, the annotators were allowed to provide more than one tag, along with their associated sub tags. Cases where this happened included statements that had independent clauses, such as two sentences that are joined by the word "but" or "and", that lacked punctuation so that the sentence splitter failed to split them, or that simply covered multiple aspects, e.g., "Nice interface and loads quickly." was assigned "usability" with the subtag "user interface aesthetics", and "performance" – "time behavior".

Step 5: Reconcile and process tagging results. After completing the tagging of the entire sample, one annotator exported the tagged statements to an Excel file. A statement was readily assigned a tag if at least three of five annotators agreed, or if the two researchers assigned the same tag. Statements that could not be unambiguously assigned one main tag were reconciled by the two researchers. They analyzed each of these statements, considered the tags assigned, and together made a decision on the definite main tag and the appropriate sub tag(s). Based on this process, we built up our ground truth.

Step 6: Analyze data. We analyzed the data resulting from the content analysis by calculating sums and frequencies, making comparisons per app (between app stores), per grouping (between categories), by price (paid vs. free), and by rating (1 to 5 stars). The results are presented in the next section.

B. Results

We analyzed 360 reviews consisting of 1.368 statements, which took each annotator approximately 10 hours to tag. A total of 163 (45.3%) reviews were tagged as containing information on quality characteristics (see Table II). They were made of 263 statements (19.0% of all statements), with two statements per review on average (i.e., the title and one sentence from the review text), and ranging from only a title to a review with a title and three sentences. Most statements were tagged with one characteristic and one subcharacteristic; 32 reviews (36 statements) were assigned two characteristics, 4 reviews (4 statements) were assigned three characteristics, and 9 reviews (9 statements) were assigned two subcharacteristics within the same characteristic.

1) ***Frequency of characteristics:*** In Table II, we sorted the characteristics and their subcharacteristics by their frequency. Users reported most frequently on three characteristics, with "usability" being the most prevalent. In turn, the subcharacteristic "operability" was discussed most often by users, in 32 (56.1%) of the 57 reviews on "usability". We additionally identified 15 statements (S) in 14 reviews (R)[1] on "usability" in general (e.g., "the most intuitive app"), while zero statements addressed the "usability" subcharacteristics "appropriate recognizability", "user error protection" or "accessibility". "Reliability" was identified as the second most frequent characteristic. Users predominantly addressed issues regarding "fault tolerance" (32 R; 60.4% of all reviews on reliability), but "maturity" was never addressed. Interestingly, for every product, at least one review addressed "reliability", though there are only 9 reviews that regard the two most frequent characteristics "usability" and "reliability". Interestingly, the third most frequently found characteristic was "portability", with the majority of the reviews (38 R; 79.2%) addressing "adaptability", which gave an idea about the environment (e.g., operating systems, Internet browsers) in which users used or would like to use the app. We concluded that we could not distinguish well between the subcharacteristics of "security", because in most cases these subcharacteristics already determine the cause or culprit. As a result, without understanding the background of the issue, one would have to guess which subcharacteristic could apply. For example, the statement "Can't even get in" may suggest either an "authentication" or "authorization" issue. Moreover, we did not identify any statement about the characteristic "maintainability" in our test set.

2) ***Comparison of app stores:*** Amazon provided the longest reviews with 480 statements, of which 20.4% (R: 62; S: 98) contained statements on quality aspects. Apple had slightly shorter reviews, resulting in 465 statements, of which 25.8% (R: 66; S: 120) were on quality aspects, though the reviews that address quality aspects were longer on average. Google had the shortest reviews, amounting to a total of 413 statements, of which only 10.9% (R: 35; S: 45) contained quality-related statements (see also Table II). Interestingly though, the reviews from Google provided a nearly even distribution of the reviews across characteristics (avg. 5, median 7, min. 3, max. 11). We also found differences in the emphasis on quality characteristics. The most frequently addressed quality with Apple was "reliability" (R: 30; S: 45) followed by "usability" (R: 28; S: 28) and "portability" (R: 20; S: 29); the reviews on Amazon focused on "usability" (R: 29; S: 36) and "portability" (R: 21; S: 25), and those from Google mainly discussed "reliability" (R: 11; S: 15), followed by "usability" (R: 7; S: 9), "portability" (R: 7; S: 8), and "compatibility" (R:7; S:7).

3) ***Comparison of apps:*** Table III shows that OneNote received a fairly average total number of statements (119 from all three app stores together), but garnered the most quality-related reviews with 80.0% (R:24; S:47/119), followed by TweetCaster with 56.7% (R:17; S:20/128) and IM+ Pro with 53.3% (R:16; S:27/129). August Smart Lock received the longest reviews overall with 209 statements, but only 8.6% of these (R:13; S:18/209) were about quality. CleverBot had the lowest share of quality-related statements with 3.1% (R: 3; S:3/97) (see Table III). The three most frequently identified quality characteristics overall are usually also the most often found quality characteristics per app. One exception is IM+ Pro, for which "compatibility" was mentioned most frequently (R:8), as users reported on the advantage of interoperability with many instant messaging services, or some unsuccessful connections between them.

4) ***Comparison of app categories:*** The product category "productivity" (see Table III) had the highest share of the quality-related reviews and statements (R: 39; S: 65/202, 32.2%), by which it accounts for 24.0% of all reviews and 25.0% of statements that are quality-related. The lowest-

[1]In the following paragraphs we will report data about both the number and the percentage of reviews and statements. For the sake of brevity, we denote reviews by R, and statements by S.

TABLE II
NUMBER OF QUALITY-RELATED REVIEWS AND STATEMENTS PER QUALITY CHARACTERISTIC IN TOTAL AND BY APP STORE (AM = AMAZON, AP = APPLE APP STORE, GO = GOOGLE PLAY). TOTALS DIFFER FROM SUMMED VALUES AS MULTIPLE TAGS COULD BE ASSIGNED TO ONE REVIEW.

Characteristics and subcharacteristics	Total Reviews (R)	Total Statements (S)	Am	Ap	Go
Usability	**57 (35.0%)**	**75 (28.5%)**	**29; 38**	**21; 28**	**7; 9**
Operability	32 (19.6%)	42 (16.0%)	15; 22	15; 18	2; 2
UI Aesthetics	13 (7.9%)	14 (5.3%)	2; 2	9; 10	2; 2
Learnability	7 (4.3%)	8 (3.0%)	7; 8	0; 0	0; 0
General	14 (8.6%)	15 (5.7%)	9; 9	1; 1	4; 5
Reliability	**53 (20.2%)**	**76 (28.9%)**	**12; 16**	**30; 45**	**11; 15**
Fault tolerance	32 (19.6%)	41 (15.6%)	8; 9	18; 24	6; 8
Recoverability	7 (4.3%)	8 (3.0%)	1; 1	4; 4	2; 3
Availability	4 (2.5%)	5 (1.9%)	0; 0	3; 4	1;1
General	22 (13.5%)	50 (19.0%)	4; 7	15; 17	3; 3
Portability	**48 (29.4%)**	**60 (22.8%)**	**21; 24**	**20; 29**	**7; 7**
Adaptability	38 (23.3%)	38 (14.4%)	14; 14	19; 19	5; 5
Installability	10 (6.1%)	10 (3.8%)	7; 7	3; 3	0; 0
Replaceability	2 (1.2%)	3 (1.1%)	0; 0	2; 3	0; 0
General	9 (5.5%)	9 (3.4%)	3; 3	4; 4	2; 2
Compatibility	**27 (16.6%)**	**31 (11.8%)**	**11; 11**	**9; 13**	**7; 9**
Interoperability	22 (13.5%)	26 (9.9%)	10; 10	8; 12	4; 4
Co-existence	1 (0.6%)	1 (0.4%)	0; 0	0; 0	1; 1
General	4 (2.5%)	4 (1.5%)	1; 1	1; 1	2; 2
Performance efficiency	**21 (12.9%)**	**30 (11.4%)**	**7; 10**	**9; 14**	**5; 6**
Time behavior	14 (8.6%)	17 (6.5%)	4; 4	5; 7	5; 6
Resource utilization	4 (2.5%)	8 (3.0%)	2; 5	2; 3	0; 0
General	5 (3.1%)	5 (1.9%)	1; 1	4; 4	0; 0
Security	**15 (9.2%)**	**19 (7.2%)**	**5; 5**	**5; 6**	**5; 8**
Functional suitability	**14 (8.6%)**	**16 (6.1%)**	**1; 1**	**10; 12**	**3; 3**
Functional correctness	9 (5.5%)	9 (3.4%)	1; 1	6; 6	2; 2
General	6 (3.7%)	7 (2.7%)	0; 0	5; 6	1; 1
Total	163 (100.0%)	263 (100.0%)	62; 98	66; 120	35; 45

scoring category was "entertainment" (R: 17; S: 27/181, 14.9%). Although in this category, Disney Movies Anywhere received a higher share of matched statements, CleverBot provided very few matches, causing a low overall score for this category. Paid apps received longer reviews on average, but fewer reviews and fewer statements about paid apps mentioned quality aspects (R: 61, 37.0%; S: 115/724, 15.9%) compared to free apps (R: 102, 63.0%; S: 145/644; 22.5%). The distribution of reviews and statements across subcharacteristics was similar to the patterns observed across the characteristics. The exception was that free apps had a larger relative percentage of reviews and statements on "security" (R: 7.9% vs. 1.2%; S: 7.9% vs. 3.7%), whereas paid apps had a higher share of statements (but not reviews) on "performance efficiency" (13.1% vs. 6.7%) and "functional suitability" (8.0% vs. 2.8%).

*5) **Comparison of ratings**:* As illustrated by Table IV, there were more quality-related reviews with a negative rating (1 and 2 stars) than with a positive rating (4 and 5 stars): 120 vs. 62. With respect to "reliability", reports on the app crashing resulted in low ratings (1 or 2 stars). Reviews regarding "usability" could provide both positive feedback (e.g., "easy to set up lists") and negative feedback (e.g., "it is difficult to navigate"). It seems that people are more inclined to writing a review when there is an issue with the quality of an app.

C. Discussion

1) Software quality subcharacteristics identified in user feedback: In this study, we were able to ***identify which quality aspects users most often mention in online reviews about apps***. Although the statements of users often contained ambiguities, which made the tagging results not as clear-cut, we did find quite a consistent pattern, namely that particular product quality characteristics and their subcharacteristics were reported more frequently than others. This pattern was consistent across apps, app categories, app stores, and price groups. ***"Reliability" and "usability" were found most often***, and they together accounted for over half of all reviews and statements on quality, though users rarely address both characteristics in one review. "Maintainability" received zero statements. Thus, it appears that users will primarily report on runtime qualities by which they are directly affected, either positively or negatively. On the other hand, we found that they do not, and in many cases cannot, contribute useful input on the maintainability of a product. This means that users have gathered expertise on particular app qualities because of their personal experience, providing reliable firsthand reports from

TABLE III
NUMBER OF QUALITY-RELATED REVIEWS FOR APPS VS. QUALITY CHARACTERISTICS.

Category	App	Usability	Reliability	Portability	Compatibility	Perf. Efficiency	Security	Func. Suitability	Total
Entertainment	Cleverbot	0	1	0	0	0	0	2	3
	DisneyMovies	3	3	9	0	2	1	1	19
Games	PacMan	2	6	1	0	2	1	0	12
	SonicSega	3	4	8	1	1	1	2	20
Messaging	IM+Pro	3	2	3	8	4	1	2	23
	Viber	3	4	2	2	2	2	1	16
Productivity	OneNote	9	6	6	6	2	6	1	36
	TinyScanner	6	7	3	0	3	0	4	23
Smart Products	PhilipsHue	7	5	8	3	0	0	1	24
	SmartLock	7	3	5	1	4	2	0	22
Social Media	TweetCaster	6	8	2	2	0	1	0	19
	TweetCasterPro	8	4	1	4	1	0	0	18
	Total	57	53	48	27	21	15	14	

using the app. However, it also means that user feedback will not provide information on (sub)characteristics that are not as visible to the users, so that mining online reviews may be less suited for helping to determine how well those quality aspects are being fulfilled.

TABLE IV
NUMBER OF QUALITY-RELATED REVIEWS FOR RATINGS VS. QUALITY. CHARACTERISTICS.

Rating	Usability	Reliability	Portability	Compatibility	Perf. Efficiency	Security	Func. Suitability	Total
1	10	18	13	4	4	2	3	54
2	17	16	9	7	6	5	6	66
3	11	13	12	7	5	3	2	53
4	11	3	9	4	5	4	2	38
5	8	3	5	5	1	1	1	24

Every characteristic also had at least one subcharacteristic that was either never mentioned, or in the case of "compatibility" and "portability", was mentioned in just one and two reviews, respectively. This pattern could be consistently observed across all apps considered, which shows that users generally report on particular quality aspects more than on others. Moreover, it means that ***statements are often best considered on a subcharacteristic level***, though users do not address all subcharacteristics. Although we only assigned a subcharacteristic if a statement could be clearly assigned to one, every characteristic had ***one predominant subcharacteristic*** that accounted for over half the tags of that characteristic. The four most common ones we found are "operability", "adaptability", "fault tolerance" and "interoperability". The only exceptions to this finding are: "maintainability" was never addressed, and it was not possible to assign subcharacteristics of "security" without knowing, for example, what caused a particular login problem.

2) What we can learn from user feedback: "Usability" is the only quality characteristic that users report on both ***positively and negatively in nearly equal numbers***; the difference in the percentage of the reviews rating 1–2 vs. 4–5 stars is approx. 10%. Reports on "operability" either confirm or deny ease of use, whereas a lack of such statements might indicate improvement potential. We also found indications that online reviews may inform among others UX designers about the elements that can spoil a positive user experience through statements such as "way to export documents isn't terribly obvious", or "Would you please provide us with the ability to have the links inside of the tweet highlighted".

Statements about "reliability" and its prevalent subcharacteristic "fault tolerance" are almost *exclusively negative* and indicative of an error occurring. Such reviews seem to be a good source of test cases (e.g., by providing attributes of the testing environment) and bug reports, conform research on using bug reports for RE [6]. Most apps appear to have at least one situation in which faults are handled improperly (e.g., crashing without informing the user of what happened; not supporting the user in finding a workaround or in reporting bugs). These kinds of reviews should be monitored with care, as they seem to have the greatest overall influence on an app's rating, followed by reviews on "performance efficiency". The latter can help to propose new or refine existing quality requirements on the one hand, and, on the other hand, help to create acceptance test cases for functions that are expected to finish faster or smoother. This seems to be especially important for apps that users expect to work efficiently, such as instant messengers or games. Statements on "adaptability" could be useful for deriving the environments (operating systems, devices) in which the app is used or in which users would like to use them. "Interoperability" provides insights into the need for or the ability to transmit data or commands across devices, or to identify which services should cooperate with one another. There is a similar potential for formulating or improving quality requirements for other (sub)characteristics, although the information available from user feedback in online reviews will be more limited.

Our results are in line with the findings of Pagano and Maalej [2], whose statistical analyses of English-language app store reviews revealed that users provide more RE-relevant user feedback for apps they interact and build a form of relationship with. The deviating pattern of comments on Cleverbot reveals that the awkward conversations with this AI bot prevented users from building up such a relationship. Pagano and Maalej also found differences in the user feedback to free vs. paid apps, as well as differences related to how many stars the user rated the app. We only found an effect of rating on the characteristic "reliability", which was reported considerably more often in reviews with a lower rating. Quality aspects were reported less often in reviews about paid apps, even though these reviews were longer on average, but the frequency with which characteristics were addressed was similar.

3) Beyond software quality: We identified one review that addressed the experience with the support provided to the product, rather than with the product quality itself. This finding creates a new research direction. It would be valuable to investigate what requirements users of software products have towards the services that are offered, such as troubleshooting with the help of customer support, and how this affects the perceived quality of the product.

III. Study II: Automated Extraction of Quality-Related Statements

A. Method

The goal of Study II was to thoroughly analyze quality-related user feedback to discover language patterns and determine if they can be identified through automated means **(RQ2)**, comparable to the way Panichella et al. [4] identified and implemented 246 recurring linguistic patterns for feature requests. In our study, we focused on statements about "usability" – the characteristic we identified as the most frequently addressed in the online reviews in Study I. Our research procedure was executed according to the following steps:

Step 1: Identify linguistic structures and define language patterns. We assessed and compared the 75 statements from the 57 reviews that were tagged as "usability" in Study I, to identify signal words and recurring expressions by which users describe this characteristic. For statements using distinct words, or for statements with similar structures, we defined language patterns. The notation of these language patterns is an extension of typical regular expressions, in which we group categories of keywords into word lists (e.g., "EN_Persons" represents any of the words "I", "you", "we", "us", . . .). A word list name acts as a placeholder. Our analysis tool cycles through all the words from a word list to find a matching pattern in the statements in our data.

For example, our results included such statements as: "The app is easy to navigate.", and "This thing works like champ, easy to customize and configure.". These are two statements that both share the words "easy to", followed by a verb. Based on this, the following language pattern on "operability" could be created in the form of a regular expression: *"(?<!EN_Negations)(that |)(easy)(to |)(navigate |customize)"* (a simplified version of language pattern ID 3). Our language patterns categorize statements along two dimensions. The first dimension concerns the type of statement, i.e., whether a statement is positive, negative, or requesting. The second dimension determines the requirement type, i.e., whether the statement addresses a functional or quality aspect (in this study, "usability" and its subcharacteristics). To ensure a clear distinction between positive and negative statements, the patterns often include operators that forbid negations such as "not". In the case of the above language pattern, this prevents from matching a negative statement like "not (that) easy to navigate". Then, the pattern was expanded with additional keywords from other reviews about "operability" so that it can match among others "simple to use".

Step 2: Evaluate the language pattern syntax. Using an online regular expression tester, we determined improvements to the language patterns based on the following rules:

- a pattern is incorrectly constructed if it does not match any statement;
- a pattern may be too restrictive if it matches one or a few statements;
- a pattern is too general if it matches statements not belonging to the (sub)characteristic it should identify;
- a pattern is too inaccurate if it cannot correctly differentiate between requirement types.

Since the regular expression tester we used did not support word lists, for the testing we replaced the placeholders with all words in the corresponding word lists, separated by the"OR" operator.

Step 3: Refine the patterns. To test the language patterns, we performed a tool-based pattern matching using the test set of 360 reviews from Study I. Based on this, we were able to measure the precision and recall on this test set. After evaluating the language patterns, we improved them in three ways: we resolved (syntax) errors, identified ways to combine similar patterns, and created positive expressions as inverse patterns to negative ones, and vice versa. For example, "A very intuitive, helpful app" and "Not very intuitive" can be recognized by a variant of the language pattern ID 6, *"(?i)(?<!EN_Negation)(EN_Emphasis)(intuitive)"*, if the operator *"(?<!EN_Negation)"* is changed to *"(EN_Negation)"*, making it possible to match the negation.

The process of evaluating and refining the language patterns was repeated until they matched (only) the statements tagged as "usability" in our test set (i.e., correct syntax, high precision and high recall). We calculated precision as "true positives/matched statements", where the matched statements represented the total of all true positives (TPs) and false positives (FPs).

Step 4: Evaluate the complete dataset. The resulting patterns were used to evaluate the complete dataset of 132,194 statements from our full dataset (see Table I). We then assessed the results to find FPs (wrong subcharacteristic, wrong requirement type, or both). The language patterns with a high FP rate (>50%) were refined in further iterations (if possible) and tested on the complete dataset again. In the results, we compared the automated results to the tagging results on a per-statement level, as language patterns match individual statements rather than reviews.

B. Results

We could define a language pattern for 39 (52.0%) of the statements that were tagged as relating to "usability" in Study I; 23 (43.4%) about "operability", 7 (13.2%) about "UI aesthetics", 5 (9.4%) about "learnability", 3 (5.7%) about both "operability" and "learnability", and one (1.9%) about both "operability" and "UI aesthetics". Surprisingly, the 15 statements on general "usability" did not provide any useful linguistic pattern. The commonalities in the structure of the statements allowed us to cover them using 16 language patterns (see Table V), of which 14 matched statements on "operability" (patterns 1-9, 11, 13-15), one on "learnability" (pattern 10), and one on "UI aesthetics" (pattern 12).

The remaining 36 (48.0%) statements did not provide a pattern for which a language pattern could be constructed, including all 15 statements (5.0%) on general "usability", 13 about "operability", and 6 about "UI aesthetics". These included vaguely formulated or nested statements, such as "One thing that bugs me right away, is all your blocked users show up right in your profile". Other statements provided too little informational content to reliably find statements on "usability" with a language pattern. For example, the pattern in the statement "What a terrible game, look, layout." is "what a terrible...", which would uncover negative statements about many aspects that are not about "usability". Alternatively, it would be possible to search for keywords like "layout", but this, in turn, makes the requirement type classification difficult.

Before we made improvements to our language patterns, we tested them against our complete dataset. This resulted in 35,940 matched statements (see Table V). However, the language pattern 9 matched 34,286 (95.4%) of these statements. It was found to be erroneous and did not allow us to deal with variations in the syntax well, therefore we excluded it from further iterations. After refining the other 15 language patterns, they matched 1,654 statements, which we could then assess on their correctness. From these results, we identified 1,528 (92.4%) TPs, and 126 (7.6%) FPs. The TPs matched 1,519 statements (99.4%) about "operability", 7 (0.5%) about "learnability", and 2 (0.1%) about "UI aesthetics". This shows that our language patterns found a larger share of statements on "operability" than we did in our tagged set.

Upon analysis of the FPs, we found that 87 (69.0%) of the 126 statements marked as a FP did address "usability", but that they were either negative statements about "usability" while the language pattern intended to find positive statements, or vice versa. For example, the language pattern *"(?i)(?<!EN_Negations)(EN_Emphasis)(intuitive)"* (language pattern ID 6) should find positive statements, but did not exclude "...settings does not seem very intuitive"). Only 27 (21.4%) of the FPs did not match the characteristic "usability" or a subcharacteristic.

TABLE V
RESULTS FROM OUR ANALYSIS OF THE 16 LANGUAGE PATTERNS OVER THE COMPLETE DATASET.

Language Pattern ID	Matched Statements	TPs	FPs	Precision
1	17	13	4	76.5%
2	3	2	1	66.6%
3	1,320	1,301	19	98.6%
4	31	31	0	100.0%
5	55	54	1	98.2%
6	20	14	6	80.0%
7	146	61	85	42.5%
8	10	10	0	100.0%
9	34,268*	–	–	–
10	14	7	7	50.00%
11	1	1	0	100.0%
12	2	2	0	100.0%
13	4	4	0	100.0%
14	1	1	0	100.0%
15	20	17	3	85.00%
16	10	10	0	100.0%
Total	1,654	1,528	126	92.4%

*)The pattern was too general and was omitted from further iterations and total counts.

C. Discussion

1) Linguistic structure of statements: In this study, we assessed the statements tagged as "usability" and found that ***about half the statements could be matched through language patterns*** **(RQ2)**. For example, many users describe their experience with the app's operability trough variations of word combinations including "(not) easy to ...", "...(not) very intuitive / user-friendly", and "would like to be able to ...". However, the patterns shall be carefully constructed. The first two given examples show a clear valence (either positive or negative), while the third shows a request for quality. This means that statements with a negative valence should require the combination of a negation and a positive word (e.g., "not easy"), or the use of a negative word (e.g., "hard"), while the same pattern could forbid those words to find statements with a positive valence. Making a distinction in valence is important when using the statements to elicit requirements because finding either many positive or many negative statements about "usability" have different implications for defining requirements. Furthermore, identification of requesting statements about qualities (i.e., "quality requests") might help to improve the software product by fixing some inadequacies or introducing new qualities. Manual extraction of quality-related statements from our full dataset would be very time-consuming. Instead, ***it was possible to construct language patterns to find statements*** following particular language patterns in a larger dataset, as detailed in Section III-A.

2) Language pattern derivation: The process of ***formulating language patterns required several iterations***, often because even ***small syntax errors can greatly compromise results***. Negatively phrased statements were sometimes more difficult to identify since in many cases adding a negating word is not enough. Nevertheless, the experience with language patterns shows promising results. Especially because patterns for positive statements can be easily adapted to find negative statements and vice versa. Instead of using language patterns, it is possible to query for particular keywords (such as the combination "easy to use"), which could reveal many relevant statements. However, such query would not allow automatically determining the statement type, e.g., if it is positive (e.g., "very easy to use"), negative (e.g., "not easy to use") or requesting (e.g., "please make it more easy to use"). For instance, the keyword-based classification by Yang and Liang did not distinguish between the quality (sub)characteristics [17]. Thus, we argue that more intricate patterns are needed.

3) Language pattern precision: The results contained many TPs, though ***we also identified several FPs***, many of which did address "usability" but had the wrong valence or subcharacteristic. Moreover, ***several matched statements contained ambiguities*** (e.g., vague formulations), meaning we could not always identify the result as a TP with certainty. When in doubt, we classified such a statement as a FP. We found that the precision of our language patterns is higher than the results of comparable studies that used keyword-based approaches (e.g., [17]) and vector-based classification techniques (e.g., [10]). This means that when performing an analysis with these language patterns, nearly all results are appropriate.

Despite the high precision, we also recognized that compared to the manual tagging, ***the 16 language patterns covered only a limited portion of all relevant statements*** on "usability". In Study I (see Section II), we found 75 statements on "usability" from 1,368 statements. Assuming that the proportion of statements will scale, we can infer that our full dataset (consisting of 384,510 statements) theoretically contains 20,518 statements on "usability", of which we covered only 1,654, or 8.1%, but using only a set of 16 language patterns. Such low recall can be explained by the focus of our study. We limited ourselves only to create language patterns that matched the tagged statements to investigate whether the tagging results could contribute to finding other statements on "usability". This way we could show that it is possible to find other statements with exactly those patterns.

There are several ways to increase recall, though some of these approaches are less systematic. One could logically theorize about similar patterns or wordings not found in the data. This could greatly improve the effectiveness of the existing language patterns. Another possibility is to construct additional language patterns. However, the frequency of patterns follows a power law distribution, as could also be seen with our results; a few language patterns are able to identify many statements that follow a commonly found pattern, while the majority of patterns are found less often. Additional language patterns will likely cover fewer statements, or may become very complex. Especially nested statements, in which a word combination about "usability" is separated by words or subordinate clauses, are difficult to identify in this way. Moreover, we could not create a language pattern for all statements tagged. Another approach, which would be more systematic, is by taking a top-down approach, where possible keywords and expressions relating "usability" would be elicited in an expert workshop. A query based on the workshops results could give rise to identifying new patterns that make use of such keywords and expressions. However, the ambiguous formulations of users will likely always prevent automation from finding some statements that human annotators would classify as "usability".

IV. Threats to Validity

We discuss the validity of our studies and results using the guidelines from Wohlin et al. [18].

Conclusion validity: To mitigate the fishing threat, we employed a software tool for tagging that presented only the reviews' contents to the annotators, involved five annotators, and defined a detailed tagging procedure. Then, to increase the reliability of the results (i.e., the reliability of the measures, the heterogeneity of annotators, and the subjectivity of the tagging), we developed a detailed tagging schema based on the structure of the ISO 25010 standard, and we performed a pilot study. We also discussed any inconsistencies and doubts among the annotators to ensure that we provided each review with the most suitable tag we could. Moreover, we reported both the number of reviews and the number of statements to provide better insights into our findings on the user feedback.

Construct validity: Another issue that might have threatened our results concerns the way the reviews are written. The language is mostly informal, the knowledge of the users varies greatly and many users write poorly or ambiguously. This also compromised the certainty with which the matched statements in Study II could be marked as TPs. In case of doubt, such statements was marked as FPs.

Internal validity: Obviously, there could be some variation in user feedback that is related to a certain (type of) app. Thus, we based our results on a stratified sample (see Table I) and included different apps (product, category, cost) from the three most popular app stores.

External validity: Although our conclusions are based on a sample of reviews for 12 apps (Study I on 360 reviews, Study II on 131,928 reviews), to increase the ability to generalize the results, we used quite a wide variety of apps regarding category and cost. The obtained online reviews come from three different international app stores, and we assessed these across all possible ratings. Of course, it should be noted that the findings might not be generalizable to other, less interactive products, as some may show entirely different patterns. Concerning the possibility of automating the extraction of quality-related statements, the conclusions are restricted to only statements about the characteristic “usability” in English.

V. Related Work

Since quality requirements are a crucial in specifying software [19] but getting a complete and correct set of quality requirements is difficult [13], many researchers and practitioners have been working on methods for elicitation and specification of quality requirements (e.g., [11], [20]). Such methods use various techniques to assist those who elicit and specify requirements. These techniques include goal modeling and noting down requirements patterns like in NFR Framework [19], PABRE [21], or templates like NoRTs [22] are suggested. The data on quality aspects extracted from online reviews can be combined with such methods as used as prompts or help with prioritization of requirements.

Combining data extracted from online reviews with existing methods may help make them less costly. A good example is the “NFR method” of Fraunhofer IESE [14] [23], an industry-validated approach to eliciting a complete set of quality requirements. This method assumes a backlog of experience-based artifacts to identify potentially relevant quality characteristics. This backlog is modified in several discrete steps to tailor it to the specificities of the project (i.e., the domain in which the quality requirements need to be elicited). This process is work- and cost-intensive and requires domain experts who typically have very limited availability. One way to reduce the manual effort required to perform the NFR method is to automate some of its steps. Such an approach using automation would make use of text mining to identify statements about relevant product quality characteristics provided by crowd members. The type of statements indicates relevant characteristics (for the current system development task, but also for the experience-based artifacts). The number and quality of these statements can then assist in assigning the priority of the characteristics which is important for focusing the limited NFR elicitation effort.

Analyzing existing textual documents to identify potential requirements has a long tradition in Requirements Engineering. Since the early 2000s, research has been adapting computational linguistic techniques to analyze user feedback for RE [24]. In product-line engineering, document analysis during scoping has proven to contribute to a better understanding of the domain better identification of product features [25]. Analysis of legal texts has been used to extract relevant requirements for software products in regulated environments [26]. Recently, various works have been published on analyzing app reviews to extract requirements related information, (e.g., by [5], [6], [7]). To our best knowledge, only research that has analyzed existing online reviews from the perspective of quality requirements, is a recent study by Lu and Liang [10]. They manually tagged 4,000 statements from online reviews on two apps using as tags some quality characteristics (“reliability”, “usability”, “portability”, and “performance efficiency”), along with “functional requirements” and “others”. In total, they tagged 1,259 statements (31.4%) as relating to one of the four quality characteristics, whereas this was only 19.0% in our work. They found that a classifier using the Augmented User Reviews – Bag-of-Words technique combined with the Bagging machine learning algorithm provides the best results. The authors report a weighted average precision of 0.714, and a weighted average recall of 0.723, though for only the quality characteristics, the weighted precision is 0.654, and the weighted recall is 0.505. In earlier work, they iteratively derived keywords by analyzing reviews about iBooks, based on which a classifier could identify functional and non-functional requirements [17].

With Crowd-based RE (CrowdRE), text mining techniques have been adapted to derive statements about requirements in natural language texts [8]. To this end, the checklists and templates used in the NFR-method are replaced by or at least augmented with keywords and queries that make use of those keywords to identify statements from texts (e.g., from crawled product reviews in app stores). But the use of Crowd-based Requirements Engineering to derive relevant product characteristics is not limited to smartphone apps in the above mentioned NFR method. Browsing a catalog of Non-functional Requirements Templates (NoRTs) [22] could also be supported with knowledge about which qualities are regarded as more important. Also, some prompts could be provided regarding how to tailor NoRTs to get quality requirements that suit potential users. Goal-oriented approaches could also benefit from CrowdRE, as the found characteristics would augment the goal models, typically as soft-goals or quality goals. The proximity of statements could be used to provide hints for relationships in goal models (e.g., [20]).

VI. Conclusion

Online user reviews are increasingly being considered as a source of requirements (see [8] for a review), but the focus has so far been on functional requirements. To assess whether they could also contribute to NFR elicitation, we analyzed the contents of online reviews about apps for statements about software product quality in two related studies according to the characterization of the ISO 25010 standard. By doing this, we sought to answer two research questions: (RQ1) what quality aspects are raised by users?, (RQ2) How should language patterns be defined so that automation can uncover quality aspects formulated by users?.

Through two consecutive studies, we identified statements about software product quality in online reviews that may provide a valuable source of information to formulate or improve quality requirements. Currently, such statements are not being considered in existing research on eliciting requirements. From the outcomes of our studies, we gather that the user feedback in online reviews can provide relevant information on app quality, but only on aspects by which users are affected.

Regarding the question what quality aspects are raised by users **(RQ1)**, we found that they especially contribute valuable information on the subcharacteristics they encounter during runtime. By further assessing the characteristic “usability”, we found that particular words and word combinations are typical for the characteristic "usability", making it possible to efficiently identify and classify statements on quality through automation by searching for these patterns **(RQ2)**. The outcomes of these searches can support the elicitation of requirements. Beyond just measuring the degree to which users perceive "usability", specific requests and problems with the app (e.g., regarding "operability") can be identified. Understanding how users perceive the quality of an app, what problems they encounter and what requests they have, can help in formulating new quality requirements or improving existing ones. We therefore argue that ***online reviews should be considered more seriously as an elicitation source for quality requirements***.

End users form a primary target group of apps, and we found that in online reviews, they report on their firsthand experience. Although these reports are limited to that what the end user can perceive (excluding "maintainability", among others), online reviews should be viewed as a valuable source of expert knowledge on quality issues affecting the end user.

Future work should determine if the distribution pattern of characteristics that we found when answering **(RQ1)** also holds for apps in other categories or for other products (e.g., software-intensive systems). To improve the generalizability of our work, a manual tagging study could also be performed on a larger share of reviews of each application, include products with more extreme rating patterns, and assess apps or software products in other categories than the ones we chose. The set of language patterns regarding "usability" should be refined and expanded, while we will investigate how users formulate other characteristics to determine how these can be identified through automation **(RQ2)**.

This work has analyzed English-language online reviews from the international app stores. Although the country of origin is often not revealed on such platforms, app stores in other languages can often be pinpointed to more precise regions our countries, based on which linguistic and demographic differences could be assessed. Additional analyses on the relationship of the results to other attributes (e.g., the number of app users) could help us gain a better understanding of the significance and background of our results. Based on our results, mechanisms could be investigated that encourage users to express their user feedback on quality in a more structured way, along with approaches to help app developers extract actionable information from user feedback.

Acknowledgment

The research described in this paper was performed in the project Opti4Apps (grant no. 02K14A182) of the German Federal Ministry of Education and Research. We would like to thank Sarah Dossinger, Sadaf Alvani and Saman Ardalan for their valuable contributions to this research, and Sonnhild Namingha from Fraunhofer IESE for reviewing this article.

References

[1] 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," *Requirements Engineering Conference*. IEEE, 2015, pp. 76–85.

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

[3] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *Requirements Engineering Conference* IEEE, 2014, pp. 153–162.

[4] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "How can i improve my app? classifying user reviews for software maintenance and evolution," in *Software maintenance and evolution (ICSME), Intl Conference on*. IEEE, 2015, pp. 281–290.

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

[6] T. Merten, M. Falis, P. Hübner, T. Quirchmayr, S. Bürsner, and B. Paech, "Software feature request detection in issue tracking systems," in *Requirements Engineering Conference (RE)*. IEEE, 2016, pp. 166–175.

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

[8] E. C. Groen, N. Seyff, R. Ali, F. Dalpiaz, J. Doerr, E. Guzman, M. Hosseini, J. Marco, M. Oriol, A. Perini, and M. Stade, "The Crowd in Requirements Engineering: The Landscape and Challenges," *IEEE Software*, 2017, to appear.

[9] K. Pohl, *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated, 2010.

[10] M. Lu and P. Liang, "Automatic Classification of Non-Functional Requirements from Augmented App User Reviews", *Intl. Conf. on Evaluation and Assessment in Software Engineering*, pp. 344–353, 2017.

[11] D. Ameller, C. Ayala, J. Cabot, and X. Franch, "Non-functional requirements in architectural decision making," *IEEE software*, vol. 30, no. 2, pp. 61–67, 2013.

[12] L. M. Cysneiros and J. C. S. do Prado Leite, "Using uml to reflect non-functional requirements," in *Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research*. IBM Press, 2001, p. 2.

[13] L. Chung and J. C. S. do Prado Leite, "On non-functional requirements in software engineering," in *Conceptual modeling: Foundations and applications*. Springer, 2009, pp. 363–379.

[14] J. Doerr, D. Kerkow, T. Koenig, T. Olsson, and T. Suzuki, "Non-functional requirements in industry-three case studies adopting an experience-based nfr method," in *Requirements Engineering Conference*. IEEE, 2005, pp. 373–382.

[15] K. A. Neuendorf, *The content analysis guidebook*. Sage, 2016.

[16] ISO, "ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models", 2010.

[17] H. Yang and P. Liang, "Identification and Classification of Requirementsfrom App User Reviews", *International Conference on Software Engineering and Knowledge Engineering*, pp. 7–12, 2015.

[18] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[19] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach," *IEEE Transactions on software engineering*, vol. 18, no. 6, pp. 483–497, 1992.

[20] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*. Springer Science & Business Media, 2012, vol. 5.

[21] S. Renault and O. Méndez Bonilla and J. Franch Gutiérrez, and M. C. Quer Bosor, "A Pattern-based Method for building Requirements Documents in Call-for-tender Processes", in *IJCSA*, vol. 6, 2009.

[22] S. Kopczyńska and J. Nawrocki, "Using non-functional requirements templates for elicitation: A case study," in *Requirements Patterns (RePa), 2014 IEEE 4th International Workshop on*. IEEE, 2014, pp. 47–54.

[23] J. Doerr, *Elicitation of a complete set of non-functional requirements*. Fraunhofer-Verlag, 2011.

[24] J. N. och Dag, B. Regnell, P. Carlshamre, M. Andersson, and J. Karlsson, "A feasibility study of automated natural language requirements analysis in market-driven development," *Requirements Engineering*, vol. 7, no. 1, pp. 20–33, 2002.

[25] I. John, *Pattern-based documentation analysis for software product lines*. University of Kaiserslautern, 2010.

[26] P. N. Otto and A. I. Antón, "Addressing legal requirements in requirements engineering," *Requirements Engineering Conf.*. IEEE, 2007.